Logic as a Query Language: from Frege to XML

Victor Vianu

U.C. San Diego

Logic and Databases: a success story

- FO lies at the core of modern database systems
- Relational query languages are based on FO:

  SQL, QBE

- More powerful query languages (all the way to XML) are based on extensions of FO
- Foundations lie in classical logic

  FO : Frege relational algebra : Tarski

Outline

- Reasons for the success of FO as a query language in relational databases
- Databases, XML, and the Web:

  the role of logic and automata theory

Why is FO so successful as a query language?

- easy to use syntactic variants

  SQL, QBE

- efficient implementation via relational algebra

  amenable to analysis and simplification

- potential for perfect scaling to large databases

  very fast response can be achieved

  using parallel processing

A relational database:

frequents

drinker	bar
Joe	King's
Joe	Molly's
Sue	Molly's
....	

serves

bar	beer
King's	Bass
King's	Bud
Molly's	Bass
....	

- logically a finite first-order structure

Find the drinkers who frequent some bar serving Bass

FO:

$\{d\text{:drinker} \mid \exists\, b\text{:bar}\ (\text{frequents}(d,b) \wedge \text{serves}(b, \text{Bass}))\}$

QBE:

frequents

drinker	bar
d	b

serves

bar	beer
b	Bass

answer

drinker
d

Find the drinkers who frequent some bar not serving Bass

FO:

$$\{d\text{:drinker} \mid \exists\, b\text{:bar}\ (\text{frequents}(d,b) \wedge \neg\ \text{serves}(b,\ \text{Bass}))\}$$

QBE:

frequents	drinker	bar
	d	b

serves	bar	beer
¬	b	Bass

answer	drinker
	d

- Naïve implementation: nested loops

$\{d\text{:drinker} \mid \exists\, b\text{:bar}\ (\text{frequents}(d,b) \wedge \text{serves}(b, \text{Bass}))\}$

for each drinker

for each bar

check the pattern

Number of checks: $|\text{drinkers}| \times |\text{bars}|$

Roughly n^2: unacceptable for large databases!

- Better approach: relational algebra

Relational algebra operations

- union, difference

- selection σ $\quad \sigma_{\text{beer} = \text{Bass}}(\text{serves}) =$

bar	beer
King's	Bass
Molly's	Bass
....	

- projection π $\quad \pi_{\text{bar}}(\text{serves}) =$

bar
King's
Molly's
......

- join $\bowtie$ frequents $\bowtie$ serves

frequents

drinker	bar
Joe	King's
Joe	Molly's
Sue	Molly's
…………	

serves

bar	beer
King's	Bass
King's	Bud
Molly's	Bass
…………	

frequents $\bowtie$ serves

drinker	bar	beer
Joe	King's	Bass
Joe	King's	Bud
Joe	Molly's	Bass
Sue	Molly's	Bass
…………		

Relational algebra queries

Find the drinkers who frequent some bar serving Bass

$$\pi_{\text{drinker}} \left(\sigma_{\text{beer = Bass}} \left(\text{frequents} \bowtie \text{serves} \right) \right)$$

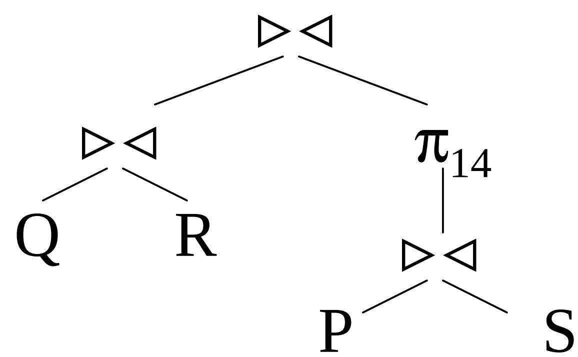
drinker
Joe
Sue
…..

drinker	bar	beer
Joe	King’s	Bass
Joe	Molly’s	Bass
Sue	Molly’s	Bass
…………….		

drinker	bar	beer
Joe	King’s	Bass
Joe	King’s	Bud
Joe	Molly’s	Bass
Sue	Molly’s	Bass
…………….		

Theorem: Relational algebra and FO are equivalent

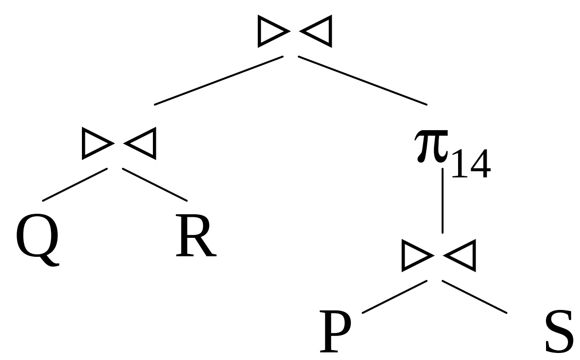
Journey of a Query

SQL	select … from … where
Relational Algebra	$\pi_{13}(P \bowtie Q) \bowtie \ldots$
Query Rewriting	$\pi_{14}(P \bowtie S) \bowtie Q \bowtie R$
Query Execution Plan	
Execution	

Physical Level

Journey of a Query

SQL	select … from … where
Relational Algebra	$\pi_{13}(P \bowtie Q) \bowtie \ldots$
Query Rewriting	$\pi_{14}(P \bowtie S) \bowtie Q \bowtie R$
Query Execution Plan	
Execution	



Physical Level

- rewriting rules for algebra queries

$\pi_{drinker}(\sigma_{beer = Bass}(\text{ frequents } |\times| \text{ serves }))$

- efficient algorithms for individual operations

Indexes: special "directories" to data

cost: roughly n (log n)

much better than n^2 for large databases!

- rewriting rules for algebra queries

$$\pi_{drinker}(\sigma_{beer = Bass}(\text{frequents} \bowtie \text{serves})) \quad \Longrightarrow$$

$$\pi_{drinker}[\text{frequents} \bowtie \pi_{bar}(\sigma_{beer = Bass}(\text{serves}))]$$

- efficient algorithms for individual operations

Indexes: special “directories” to data

cost: roughly n (log n)

much better than n^2 for large databases!

Most spectacular: theoretical potential for perfect scaling!

- perfect scaling: given sufficient resources, performance does not degrade as the database becomes larger
- key: parallel processing
- cost: number of processors polynomial in the size of the database (FO $\subseteq$ AC_0)
- role of algebra: operations highlight parallelism

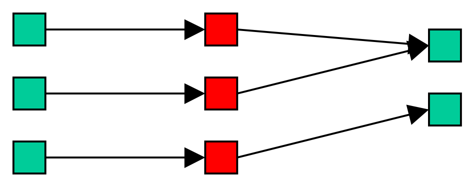
Each algebra operation can in principle be implemented very efficiently in parallel

Example: projection

π_{bar} (serves)

serves

bar	beer
King’s	Bass
King’s	Bud
Molly’s	Bass
……………	

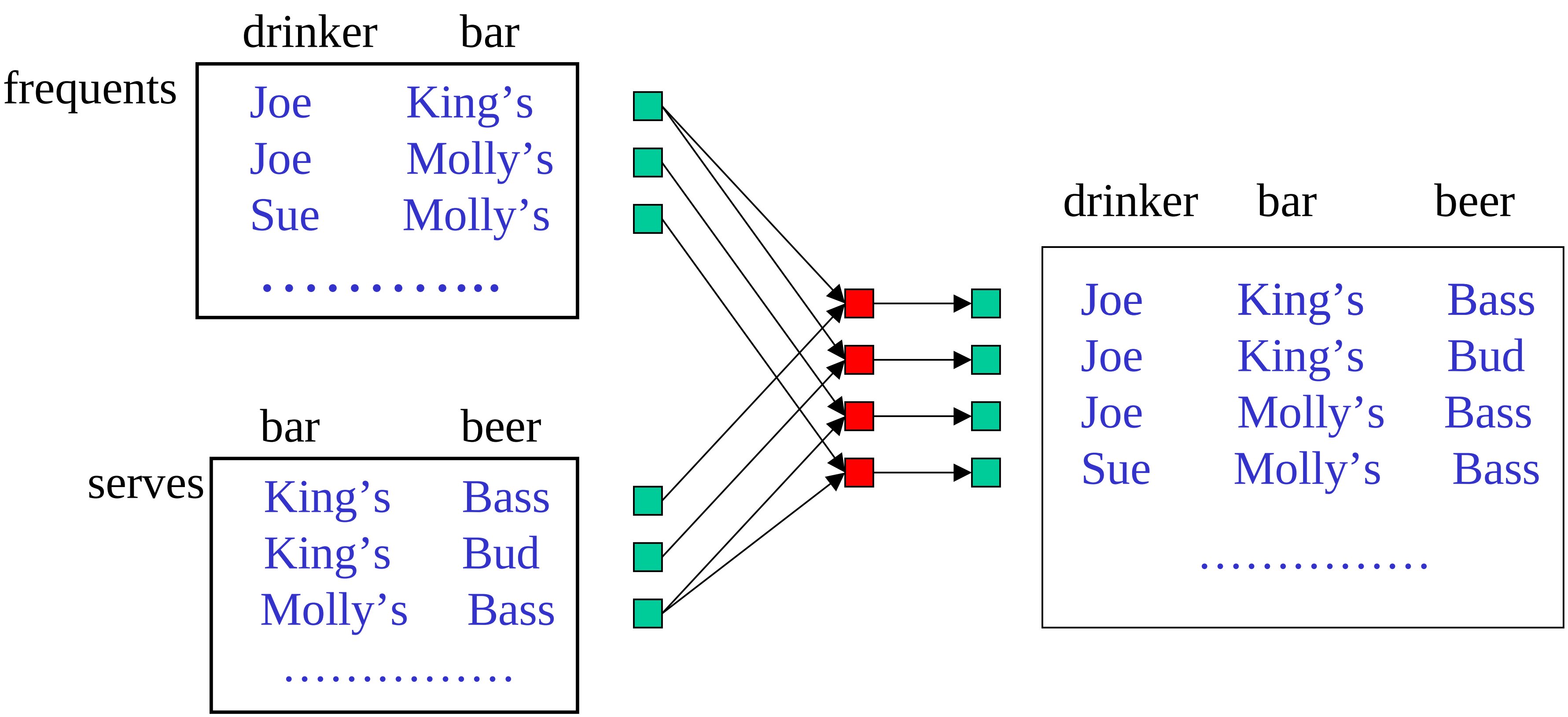


bar
King’s
Molly’s
……

Constant parallel time!

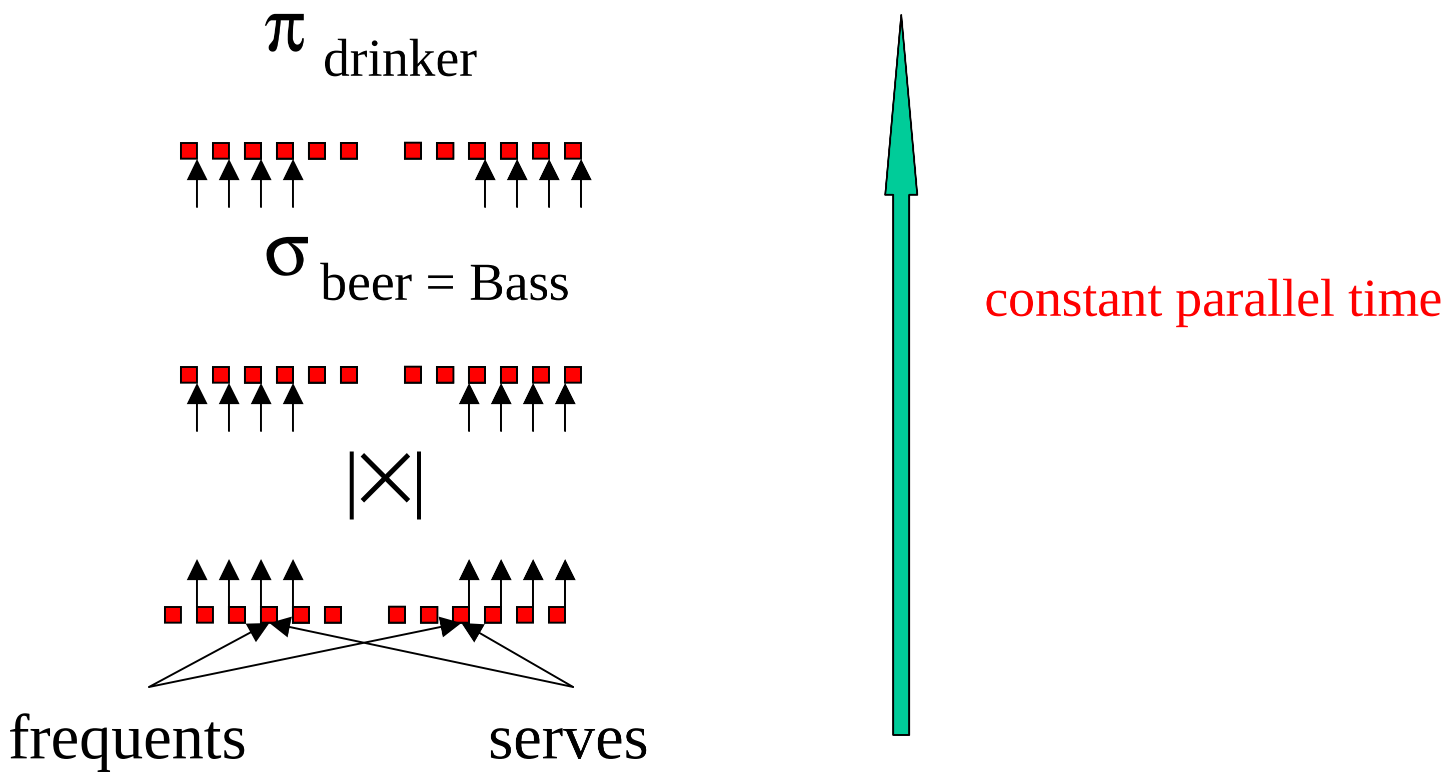
Another example: join

frequents |X| serves



Every relational algebra query takes constant parallel time!

$$\pi_{\text{drinker}}(\sigma_{\text{beer = Bass}}(\text{ frequents} \bowtie \text{serves }))$$



Summary so far:

- Keys to the success of FO as a query language:

  -ease of use

  -efficient implementation via relational algebra

- Constant parallel complexity:

  the full potential of FO as a query language remains

  yet to be realized!

Beyond the basics

- Recursive extensions of FO

  Datalog, fixpoint logics
- Intimate connection to finite-model theory

  see Phokion and Ron’s talks
- Other uses of logic in classical db theory:

  dependency theory: see Jeff’s talk

  incomplete information: see Serge’s talk

  deductive databases, knowledge bases

Databases, XML, and the Web

- relations replaced by trees (XML data)
- structure described by flexible schemas (e.g., DTDs)

Again, logic provides the foundations:

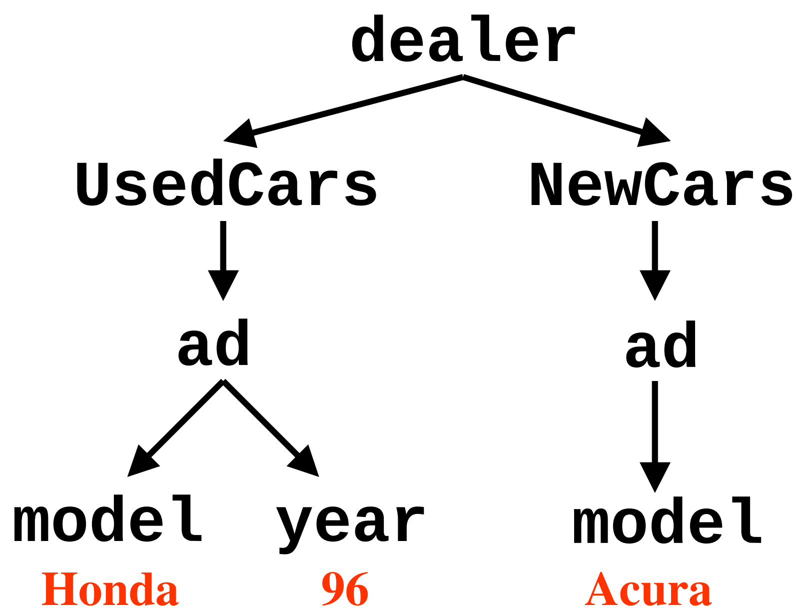
- DTDs are equivalent to tree automata (MSO on trees)
- XML queries are essentially tree transducers
- Can use automata and logic to understand semantics and expressiveness, perform static analysis

Most XML query languages are extensions of SQL

- implementation based on same paradigm
- uses extensions of relational algebra
- query optimization builds upon relational techniques

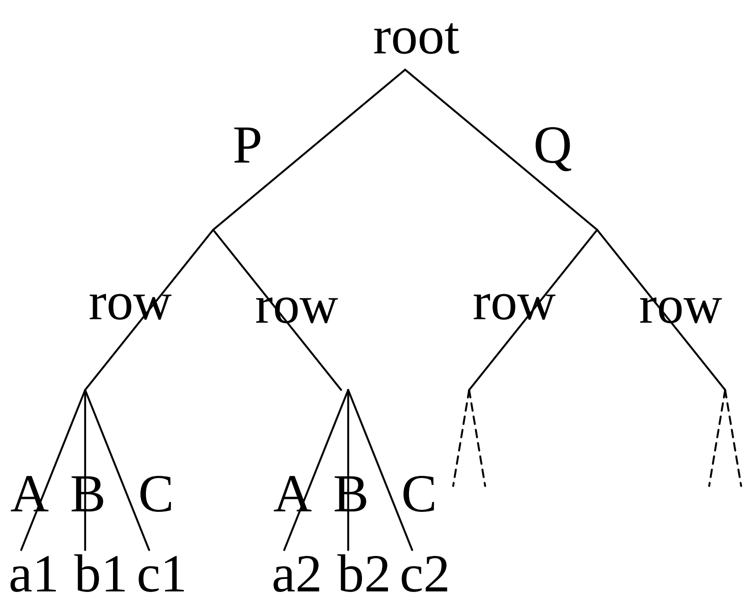
XML and DTDs

```
<dealer>
 <UsedCars>
  <ad>
   <model>Honda</model>
   <year>96</year>
  </ad>
 </UsedCars>
 <NewCars>
  <ad>
   <model>Acura</model>
  </ad>
 </NewCars>
</dealer>
```



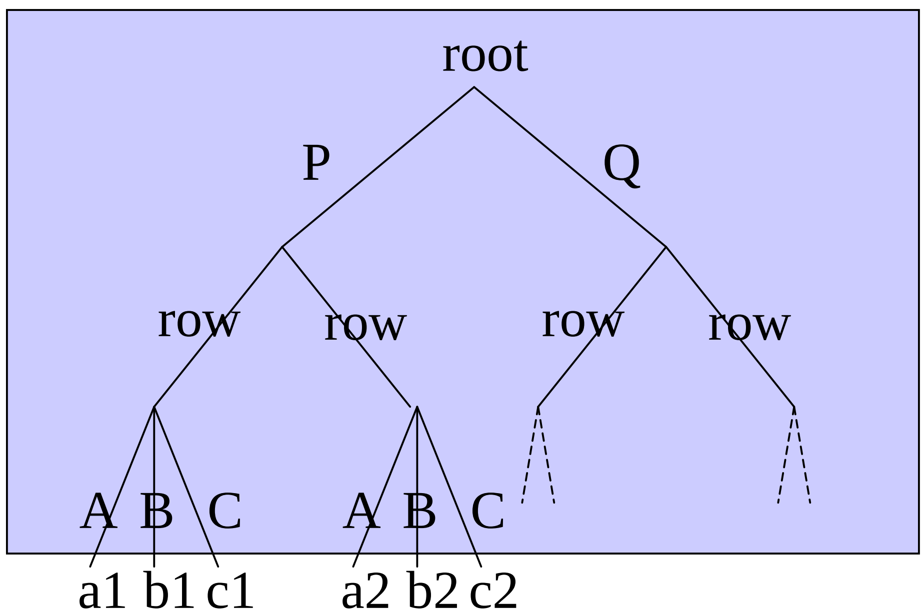
P		
A	B	C
a1	b1	c1
a2	b2	c2

Q	
C	D
c2	d2
c3	d3



P		
A	B	C
a1	b1	c1
a2	b2	c2

Q	
C	D
c2	d2
c3	d3



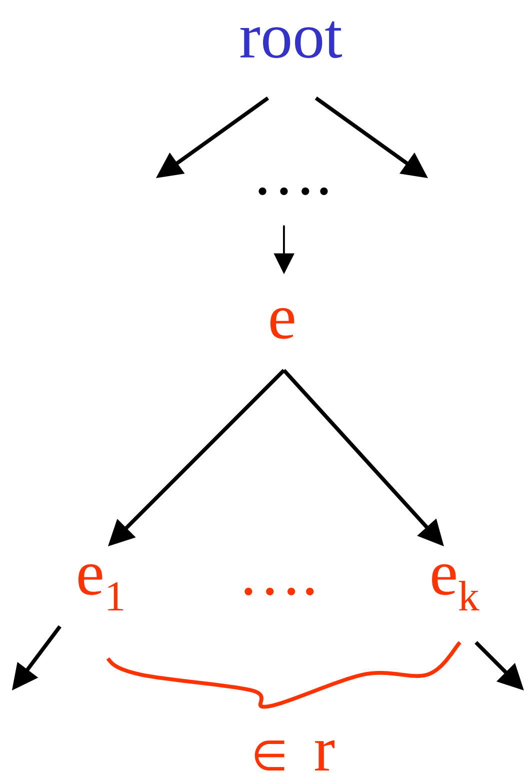
Data Type Definition (DTD)

Σ: alphabet of element names, root $\in \Sigma$

set of rules:

$$\underbrace{e}_{\text{element name}} \longrightarrow \underbrace{r}_{\text{regular expression over } \Sigma}$$

Documents satisfying a DTD



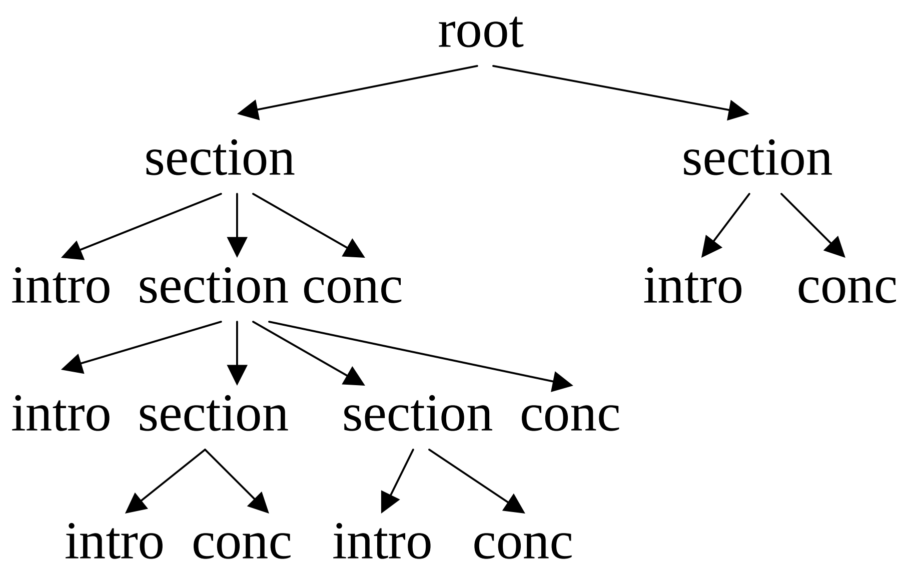
$e \longrightarrow r$

Set of trees satisfying DTD d: **T(d)**

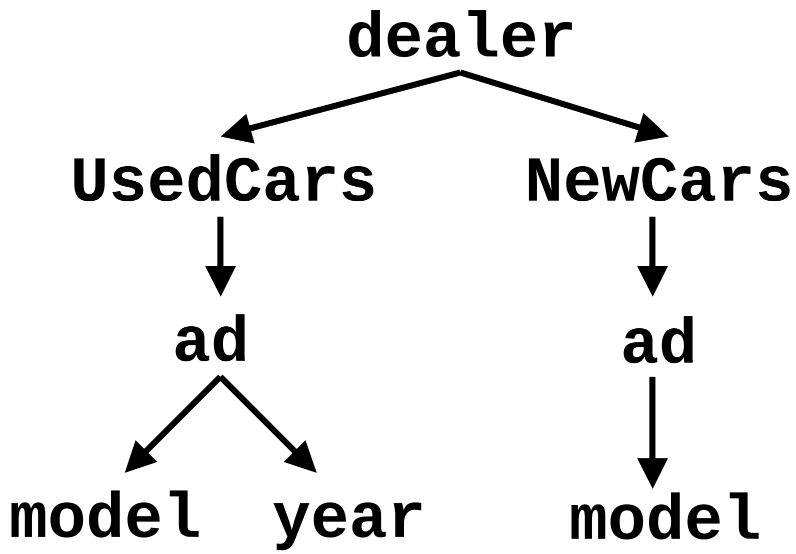
Example

A DTD and a tree satisfying it:

```
    root    →  section*;
 section    →  intro, section*,conclusions;
```

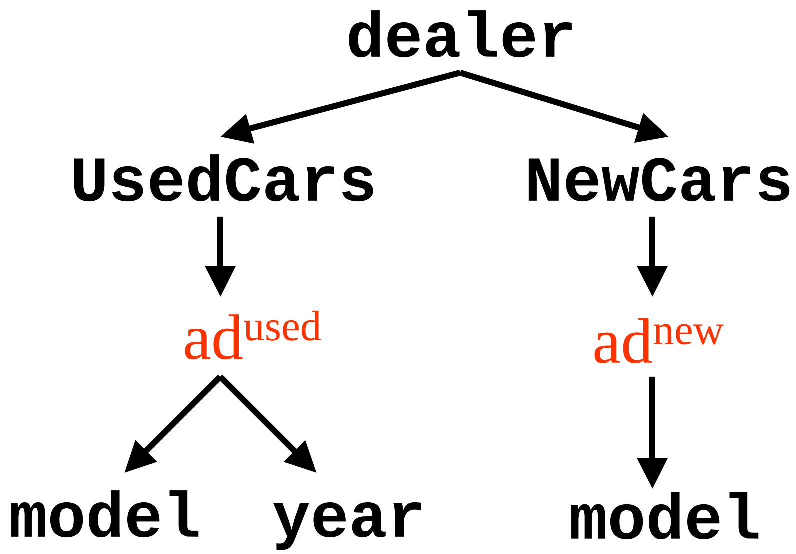


Specialization



ad has different structure in different contexts

Specialization



ad has different structure in different contexts

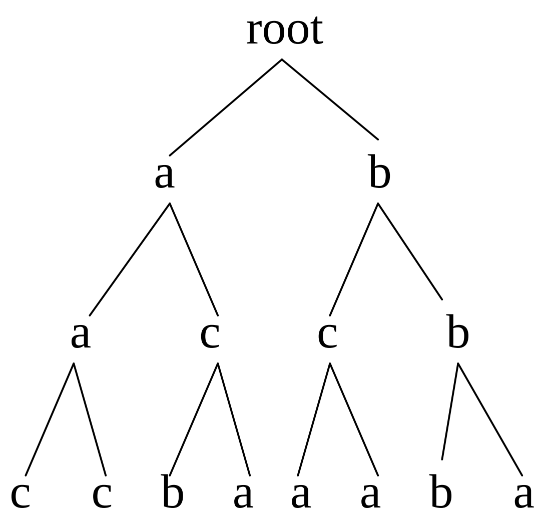
Basic questions on XML schemas

- Validation: how hard?
- Expressive power: what can be defined?
- Closure properties:

  union, intersection, difference
- Complexity of manipulations

Tool: powerful connection to tree automata!

Tree automata

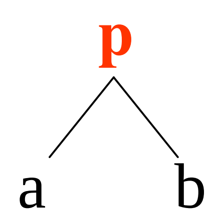


States: **p,q,r, …**

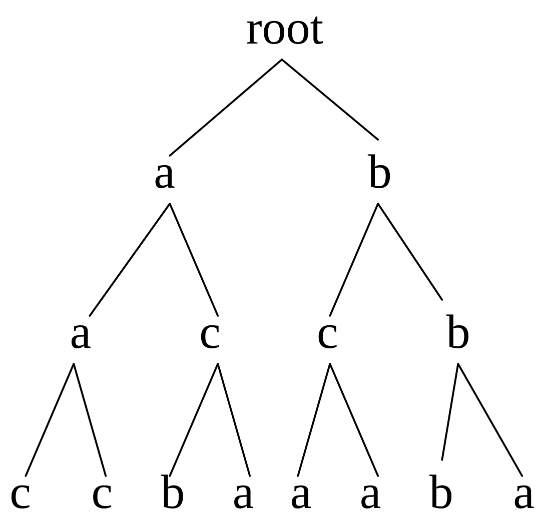
Start state: $\mathbf{q_0}$

Final state: $\mathbf{q_f}$

Transitions:



Tree automata

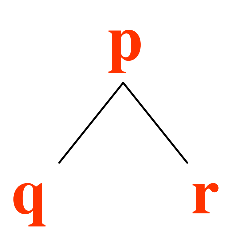


States: **p,q,r, …**

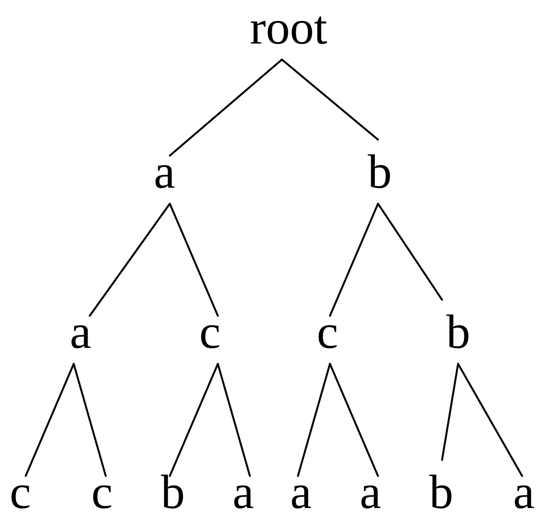
Start state: $\mathbf{q_0}$

Final state: $\mathbf{q_f}$

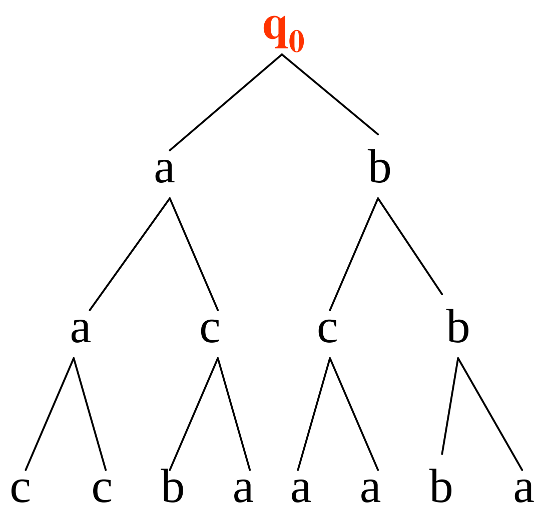
Transitions:



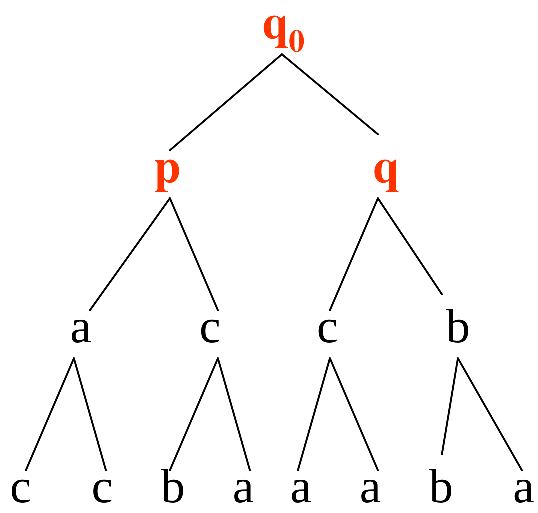
Tree automaton computation



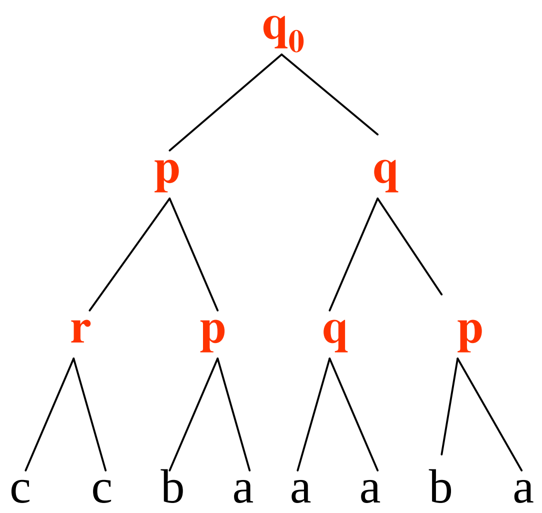
Tree automaton computation



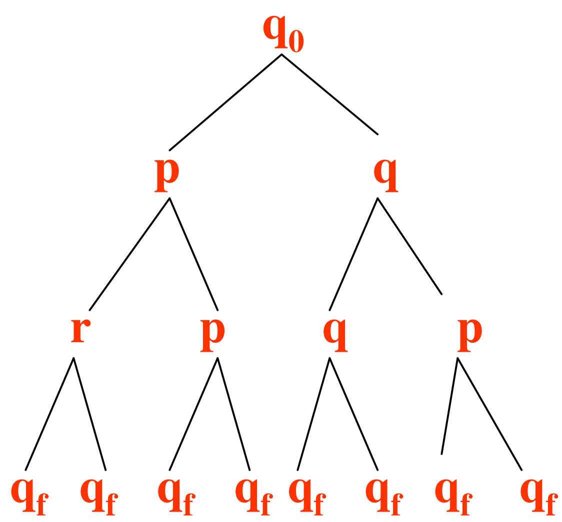
Tree automaton computation



Tree automaton computation



Tree automaton computation

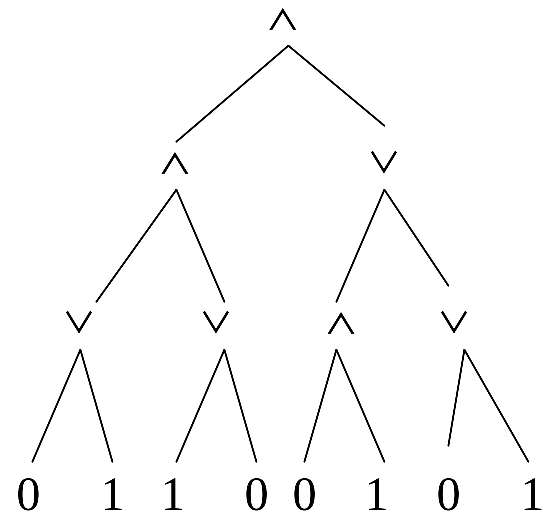


Acceptance: there is a computation such that all leaves are labeled $\mathbf{q_f}$

Set of accepted trees: regular tree language

Examples

- regular: set of trees representing Boolean circuits evaluating to true



- not regular: equal numbers of a and b nodes

Regular: definable in Monadic Second-Order Logic

Theorem: DTDs with specialization define precisely the regular tree languages

Benefits:

- Closure properties:
  union, intersection, complement
- Algorithms for validation wrt DTD,
  testing inclusion of DTDs, computing DTDs
  for union or intersection, etc.
- Static analysis

Query languages for XML

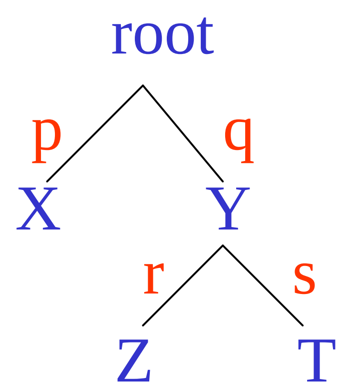
- One paradigm (XML-QL, Lorel, XMAS, etc)
  - extract bindings for variables using patterns
  - construct answer from bindings
- Another paradigm (XSLT, UnQL)
  - structural recursion
- Most recent: XQuery

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

pattern:



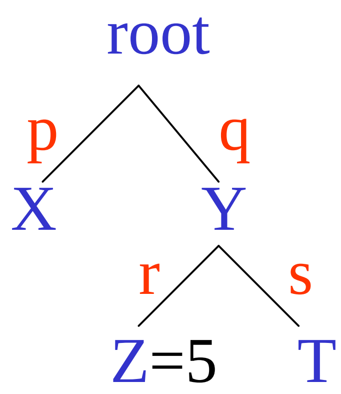
p,q,r,s: regular path expressions

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

pattern:



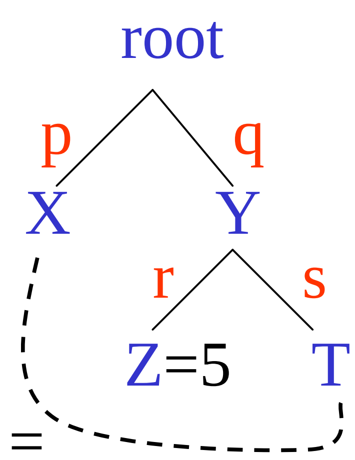
p,q,r,s: regular path expressions

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

pattern:



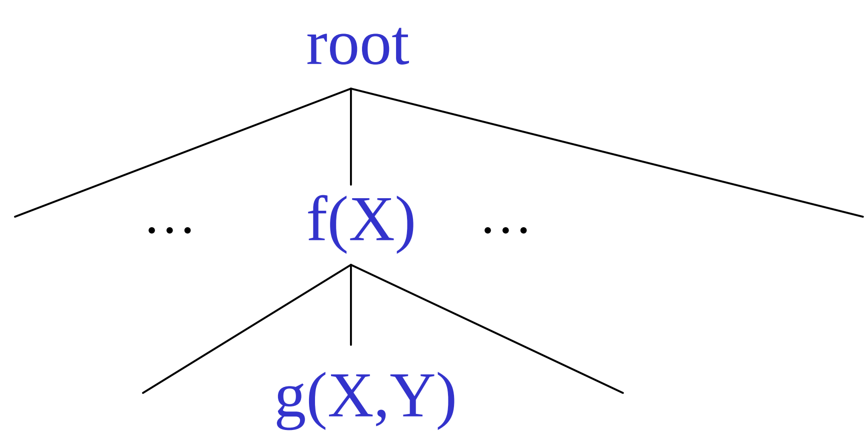
p,q,r,s: regular path expressions

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

answer:



f(X), g(X,Y)

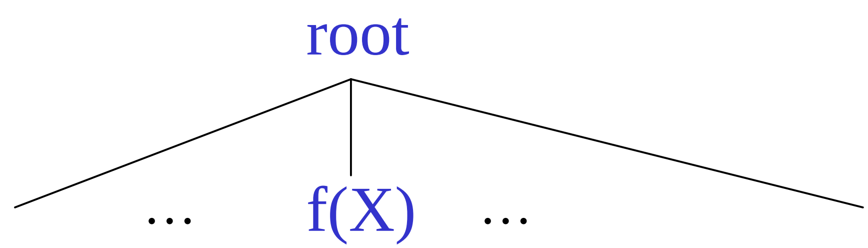
Skolem functions

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

answer:



f(X), g(X,Y)

Skolem functions

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

answer: root

f(X_1) …f(X_i) …f(X_n)

f(X), g(X,Y)

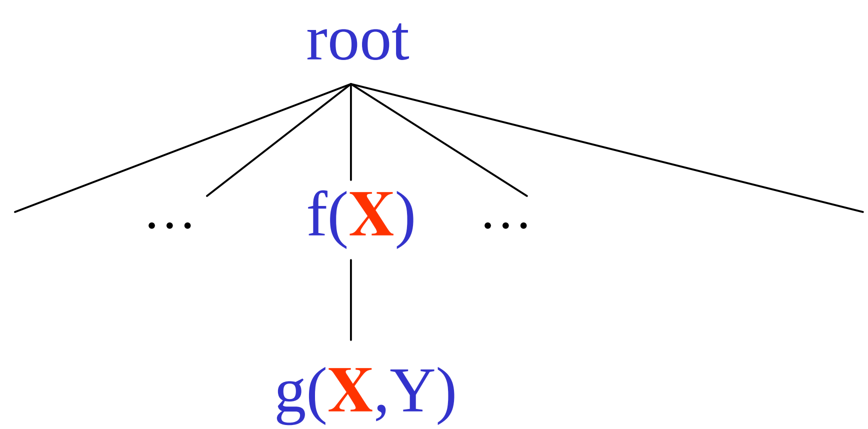
Skolem functions

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

answer:



f(X), g(X,Y)

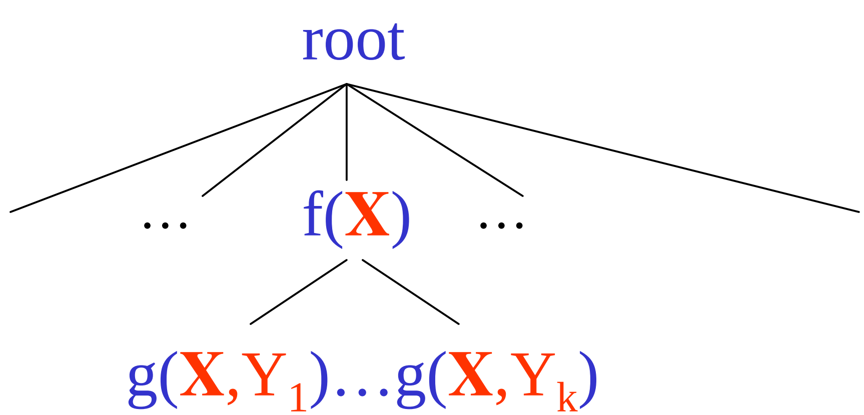
Skolem functions

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

answer:



f(X), g(X,Y)

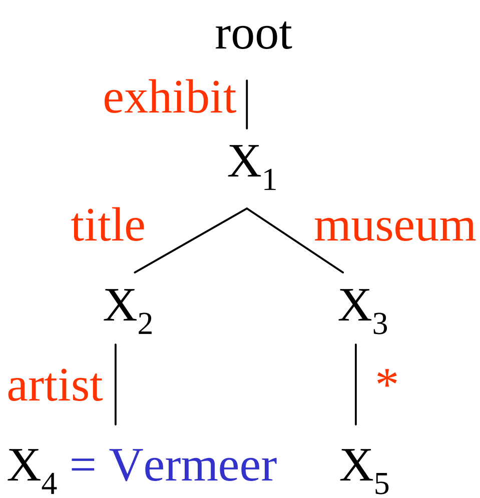
Skolem functions

Example: art exhibits

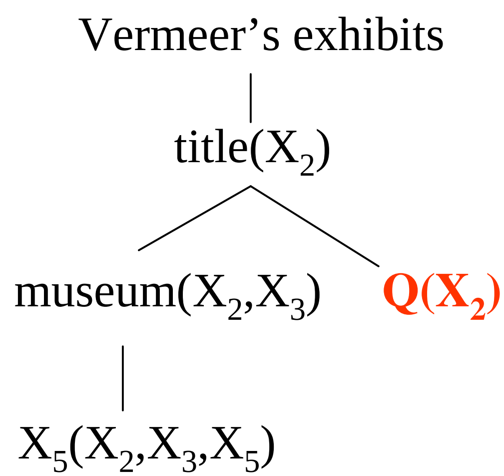
DTD:

root $\rightarrow$ exhibit*
exhibit $\rightarrow$ title. museum*. review*
title $\rightarrow$ artist*
museum $\rightarrow$ address. dates._*

WHERE

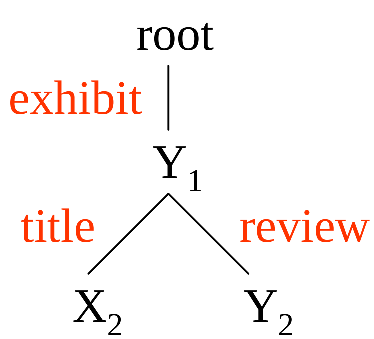


CONSTRUCT

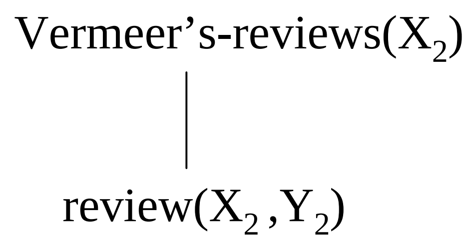


Nested query **$Q(X_2)$:**

WHERE



CONSTRUCT



How are XML queries different?

- Ordered tree data
- Limited recursion (path expressions)
- Limited negation (path expressions)
- Constructed answers
- Expressiveness: no natural yardstick for "completeness"

  not even closed under composition!

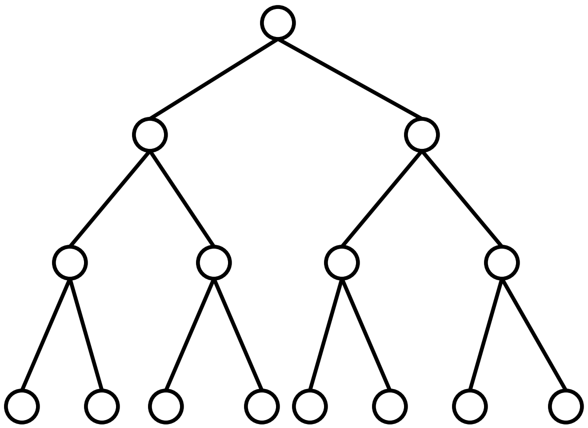
A unifying formalism: tree transducers

k-pebble transducers

[Milo+Suciu+V.-]

- subsume core of usual XML languages
- useful for static analysis

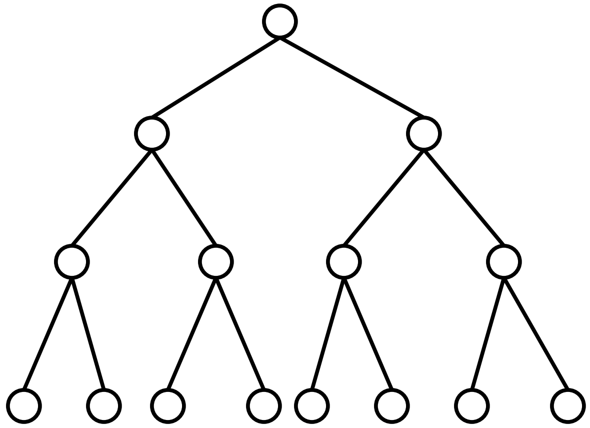
k-pebble transducers



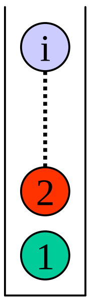
k pebbles: 

- – States, pebbles, transition rules
- – Control: alternating

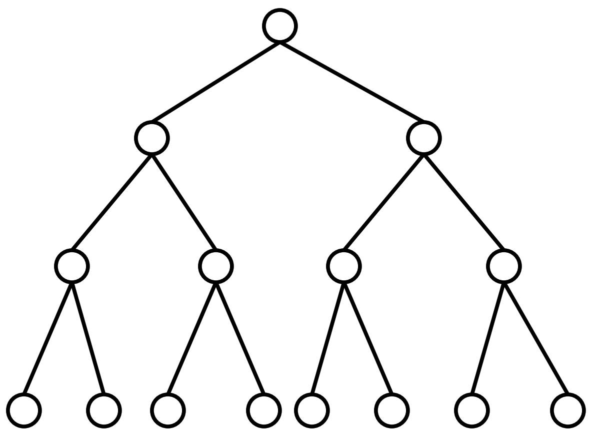
k-pebble transducers



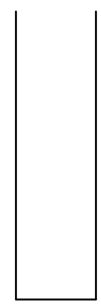
stack discipline for pebbles



k-pebble transducers

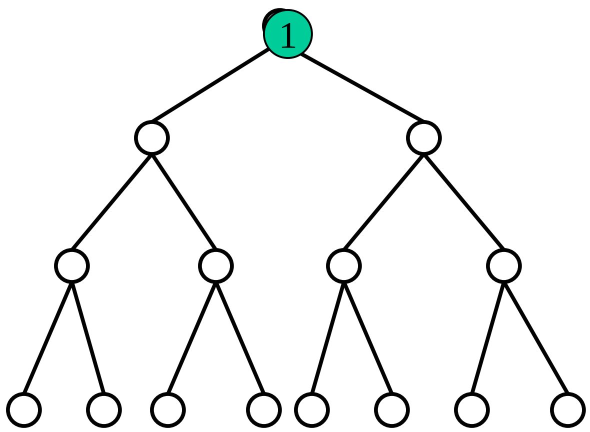


stack discipline for pebbles

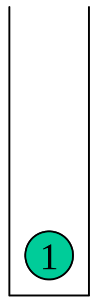


3 pebbles:

k-pebble transducers

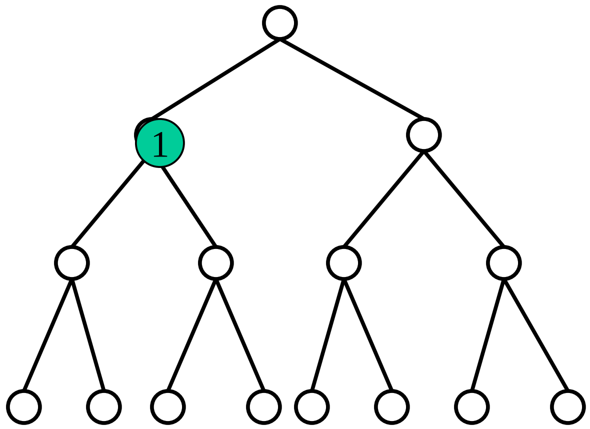


stack discipline for pebbles

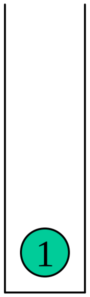


3 pebbles: 1 2 3

k-pebble transducers

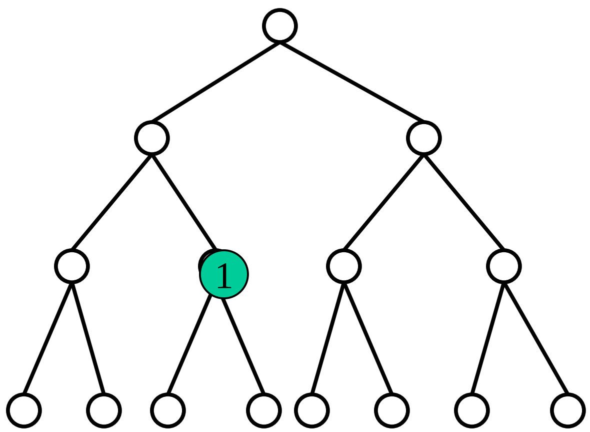


stack discipline for pebbles

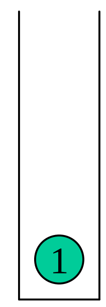


3 pebbles: 

k-pebble transducers

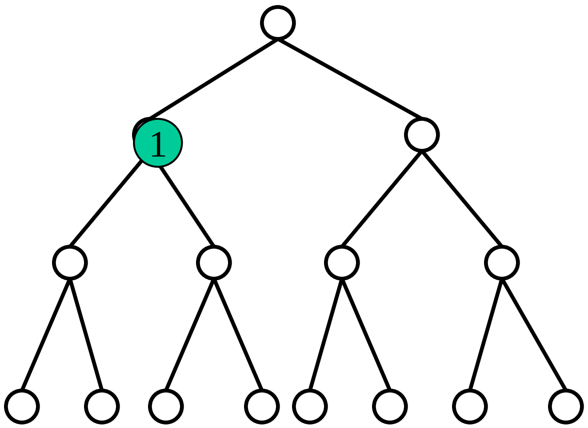


stack discipline for pebbles

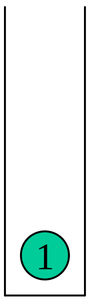


3 pebbles: 1 2 3

k-pebble transducers

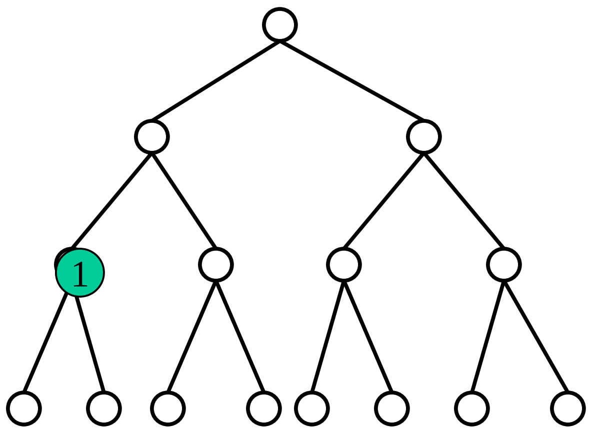


stack discipline for pebbles

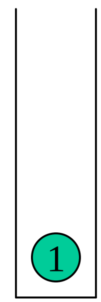


3 pebbles: 1 2 3

k-pebble transducers

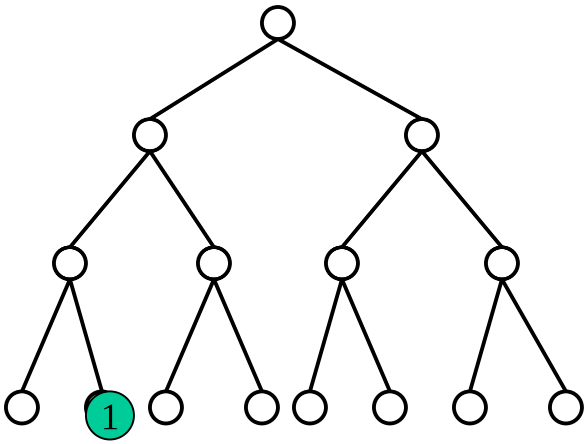


stack discipline for pebbles

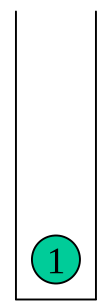


3 pebbles: 

k-pebble transducers

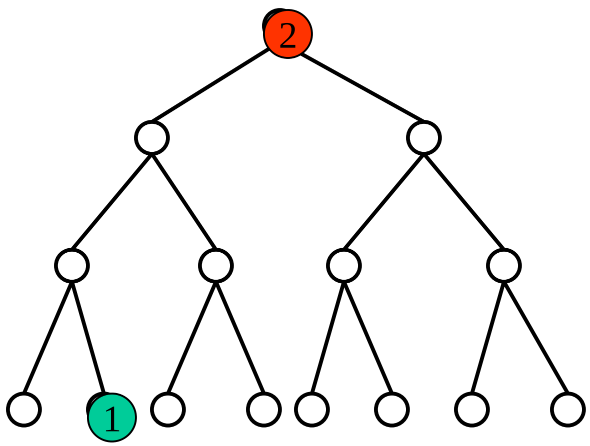


stack discipline for pebbles

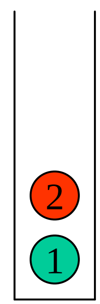


3 pebbles: 1 2 3

k-pebble transducers

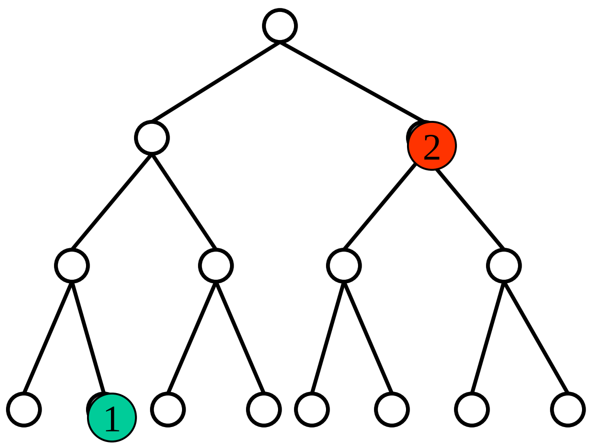


stack discipline for pebbles

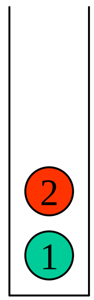


3 pebbles: 1 2 3

k-pebble transducers

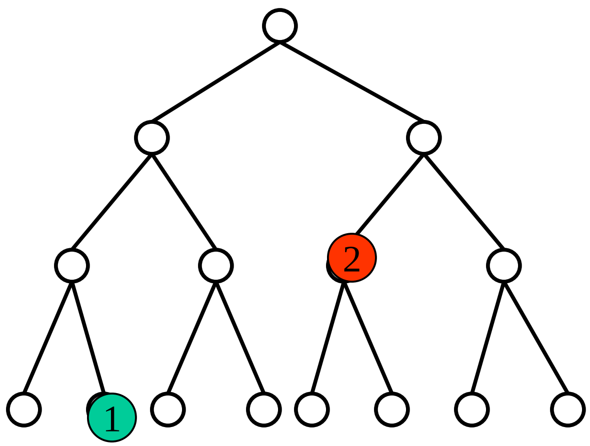


stack discipline for pebbles

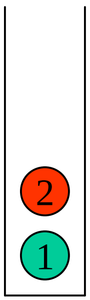


3 pebbles: 1 2 3

k-pebble transducers

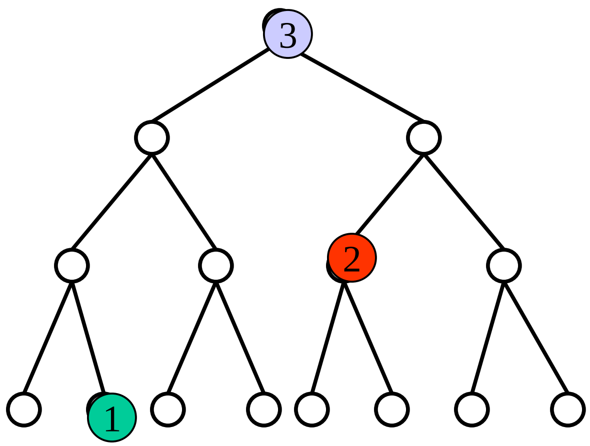


stack discipline for pebbles

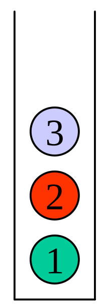


3 pebbles: 

k-pebble transducers

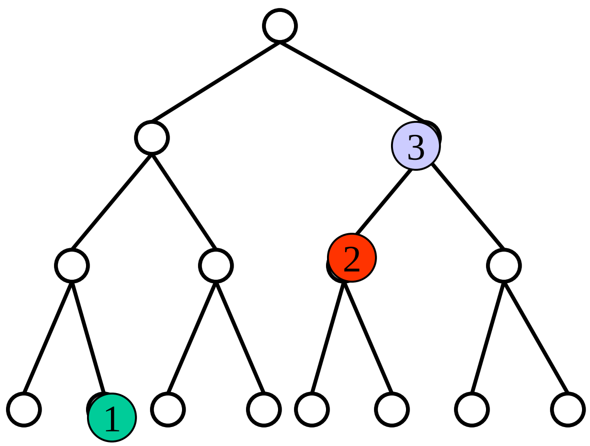


stack discipline for pebbles

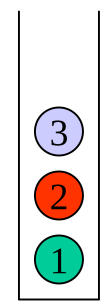


3 pebbles:

k-pebble transducers

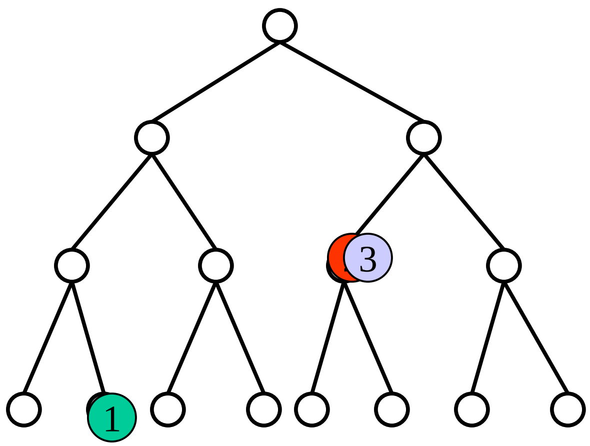


stack discipline for pebbles

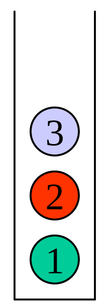


3 pebbles: 

k-pebble transducers

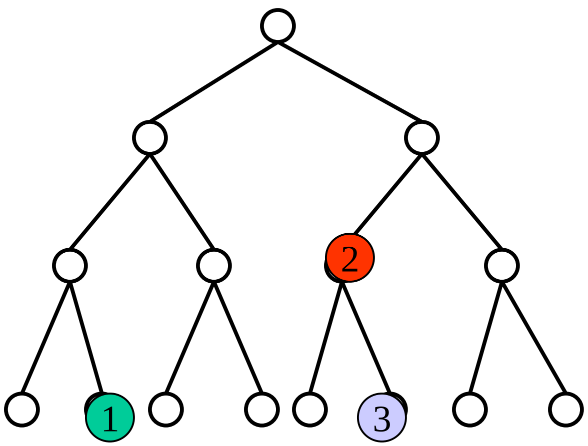


stack discipline for pebbles

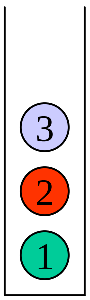


3 pebbles: 

k-pebble transducers

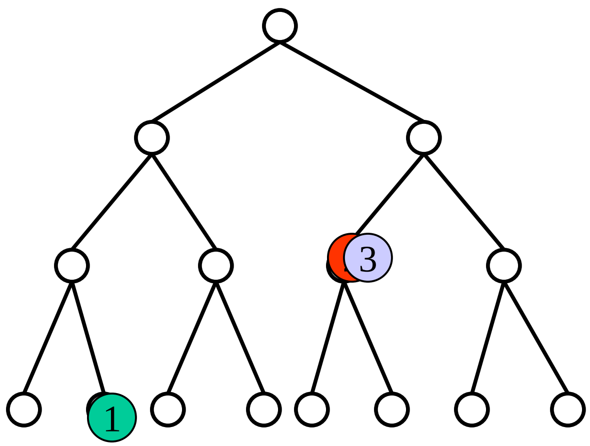


stack discipline for pebbles

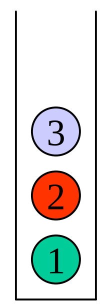


3 pebbles: 1 2 3

k-pebble transducers

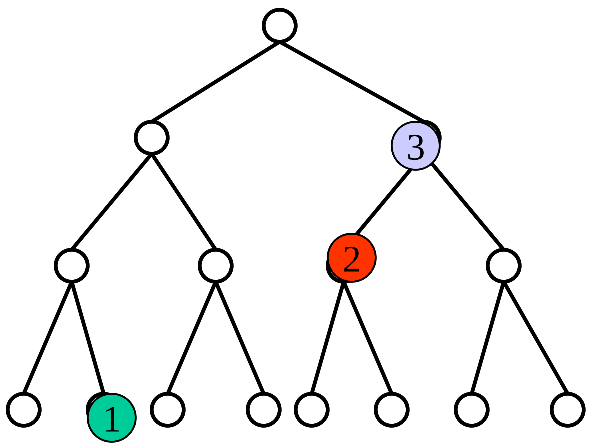


stack discipline for pebbles

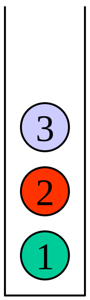


3 pebbles:

k-pebble transducers

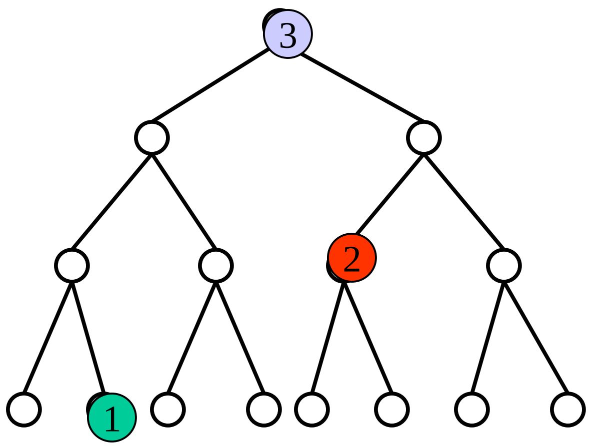


stack discipline for pebbles

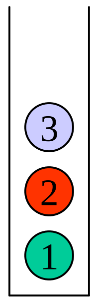


3 pebbles: 

k-pebble transducers

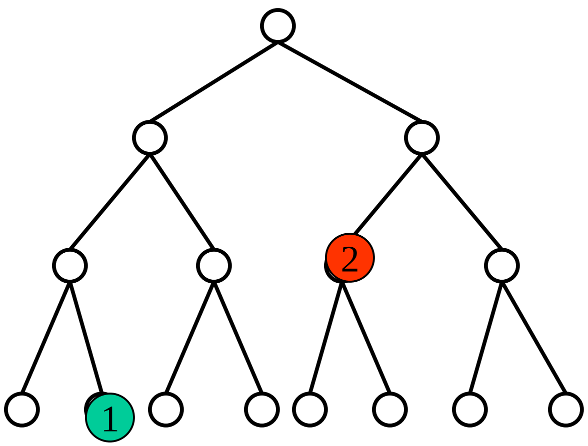


stack discipline for pebbles

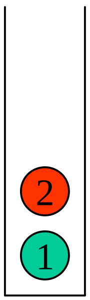


3 pebbles: 1 2 3

k-pebble transducers

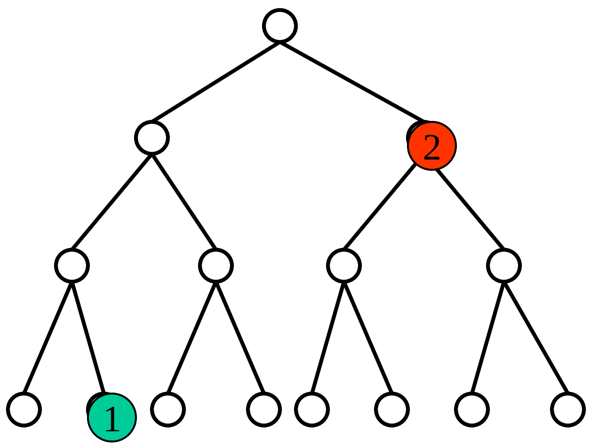


stack discipline for pebbles

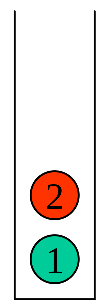


3 pebbles: 

k-pebble transducers

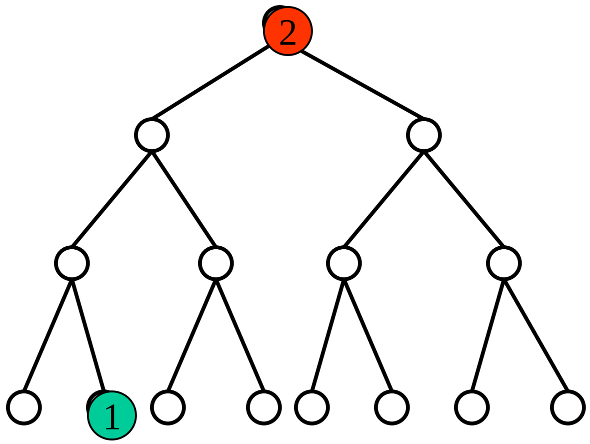


stack discipline for pebbles

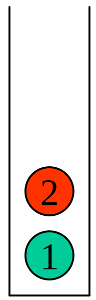


3 pebbles: 

k-pebble transducers

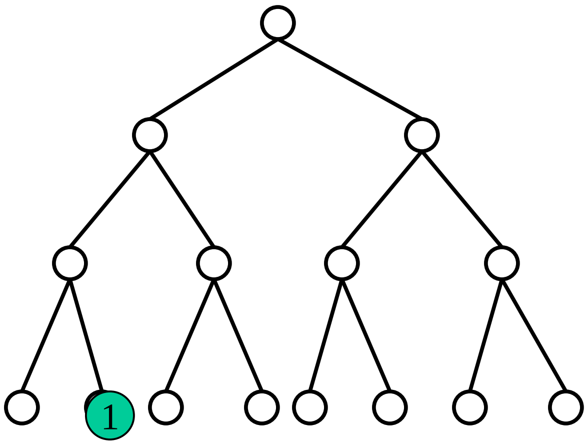


stack discipline for pebbles

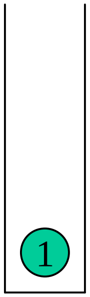


3 pebbles: 1 2 3

k-pebble transducers

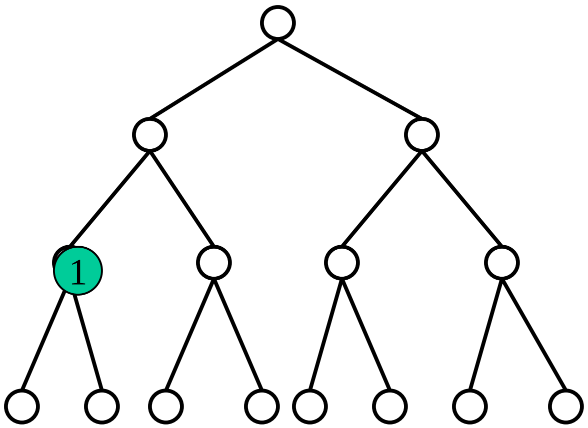


stack discipline for pebbles



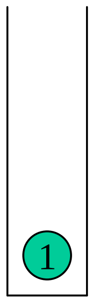
3 pebbles:

k-pebble transducers

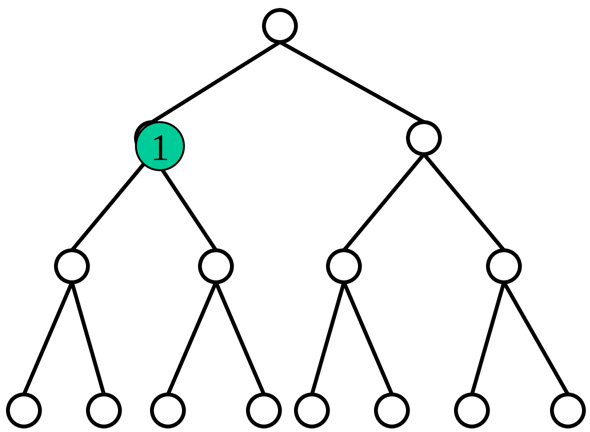


3 pebbles:

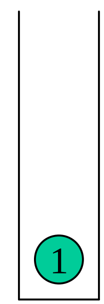
stack discipline for pebbles



k-pebble transducers

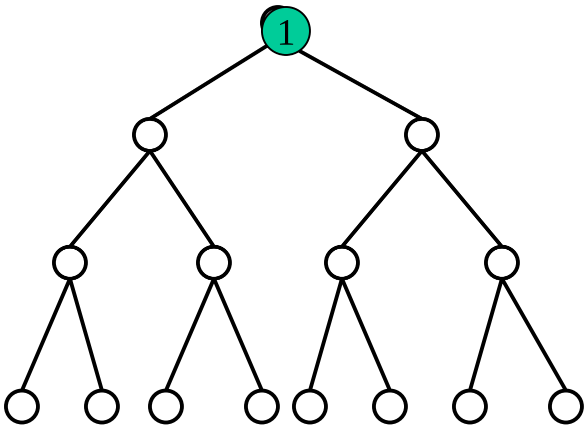


stack discipline for pebbles

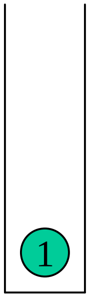


3 pebbles: 1 2 3

k-pebble transducers



stack discipline for pebbles

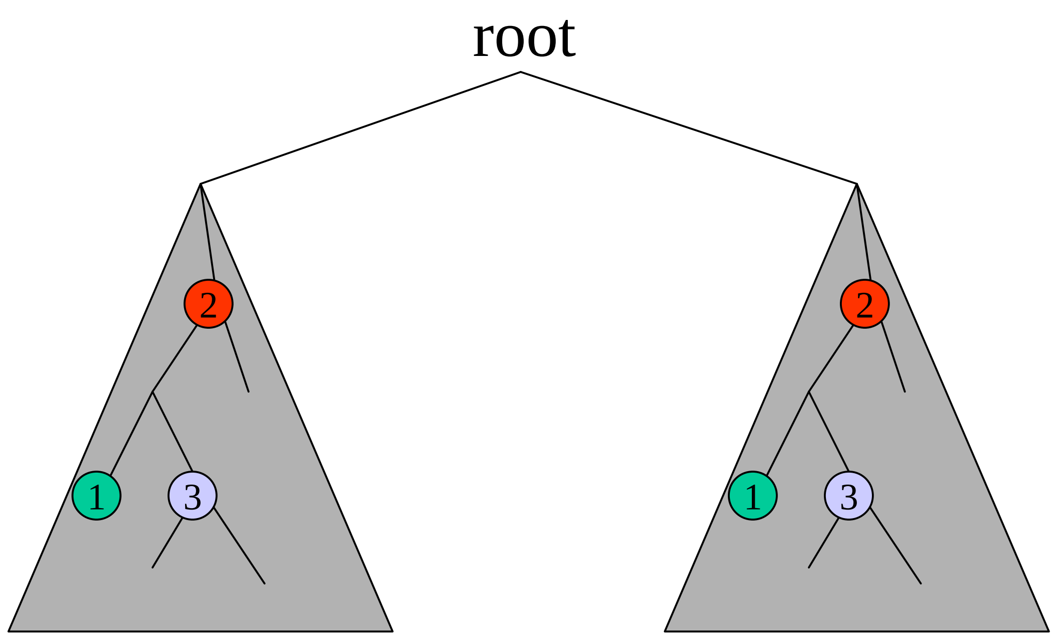


3 pebbles: 1 2 3

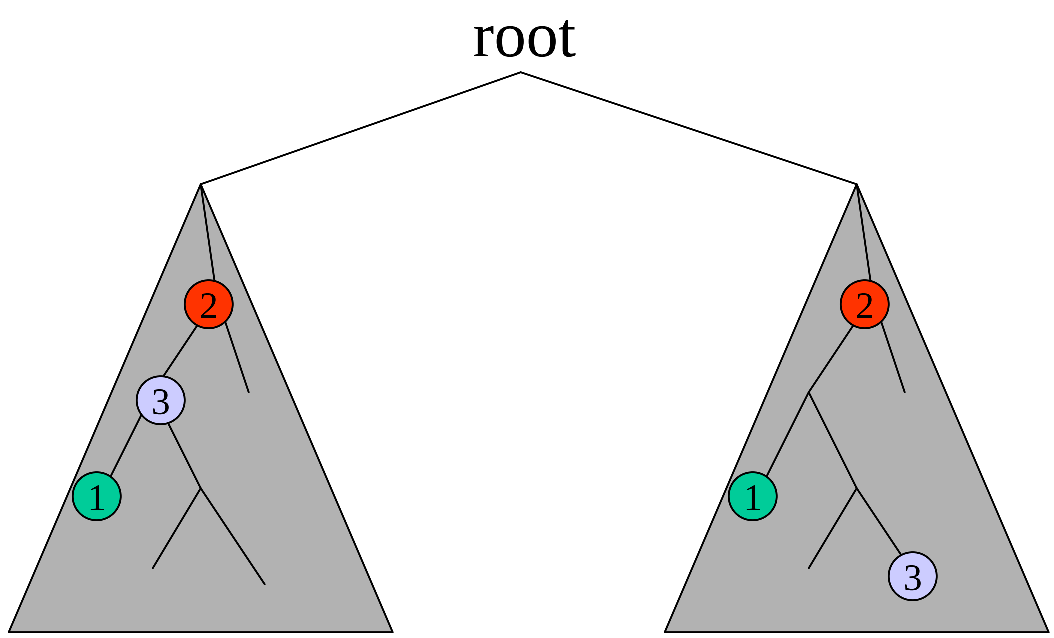
Output-producing transitions

root

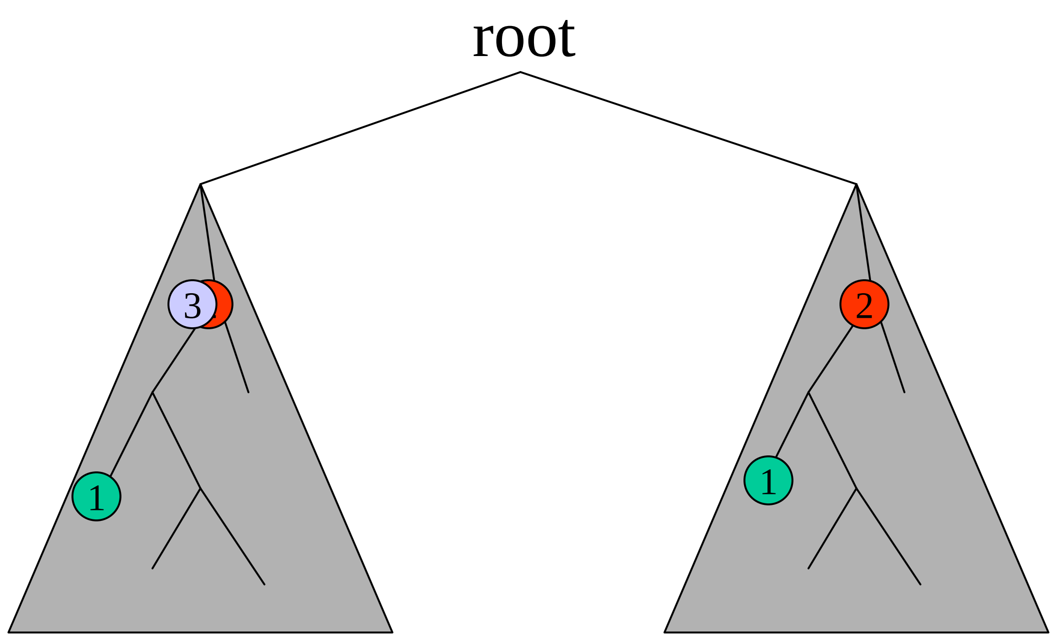
Output-producing transitions



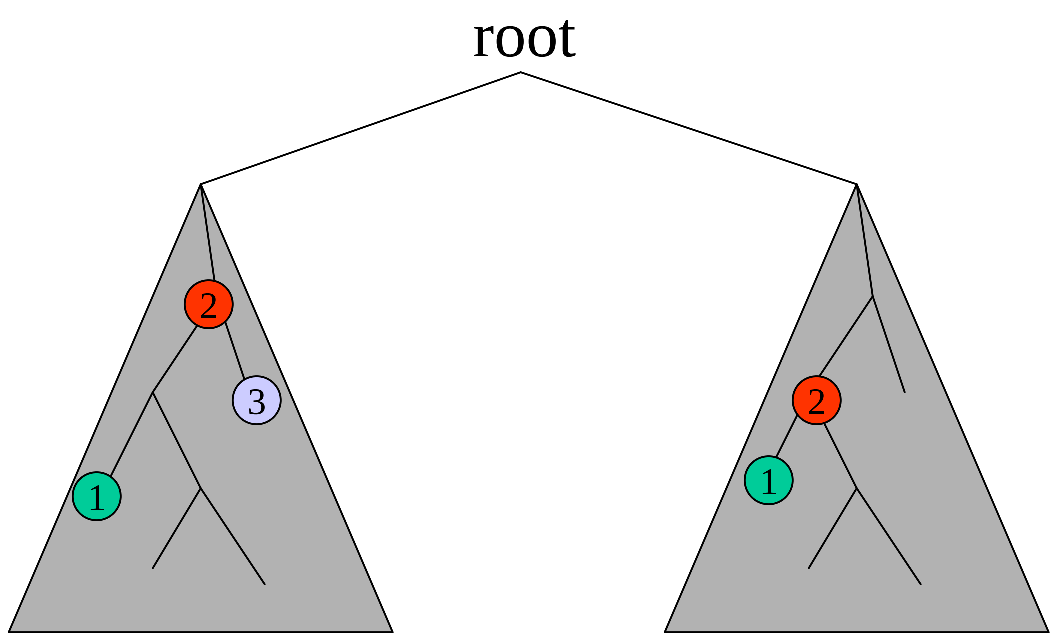
Output-producing transitions



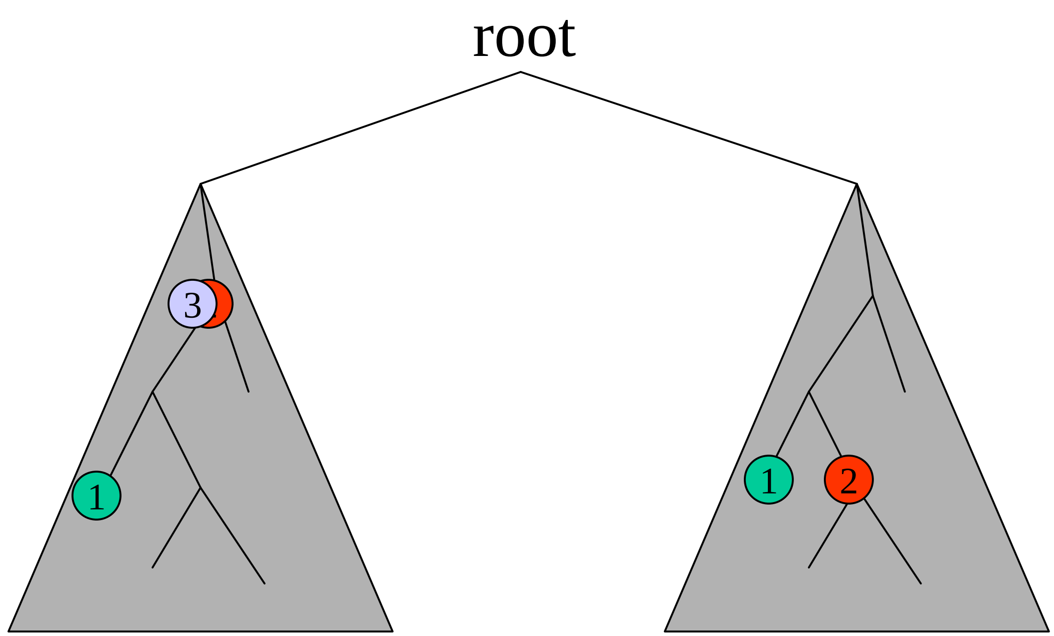
Output-producing transitions



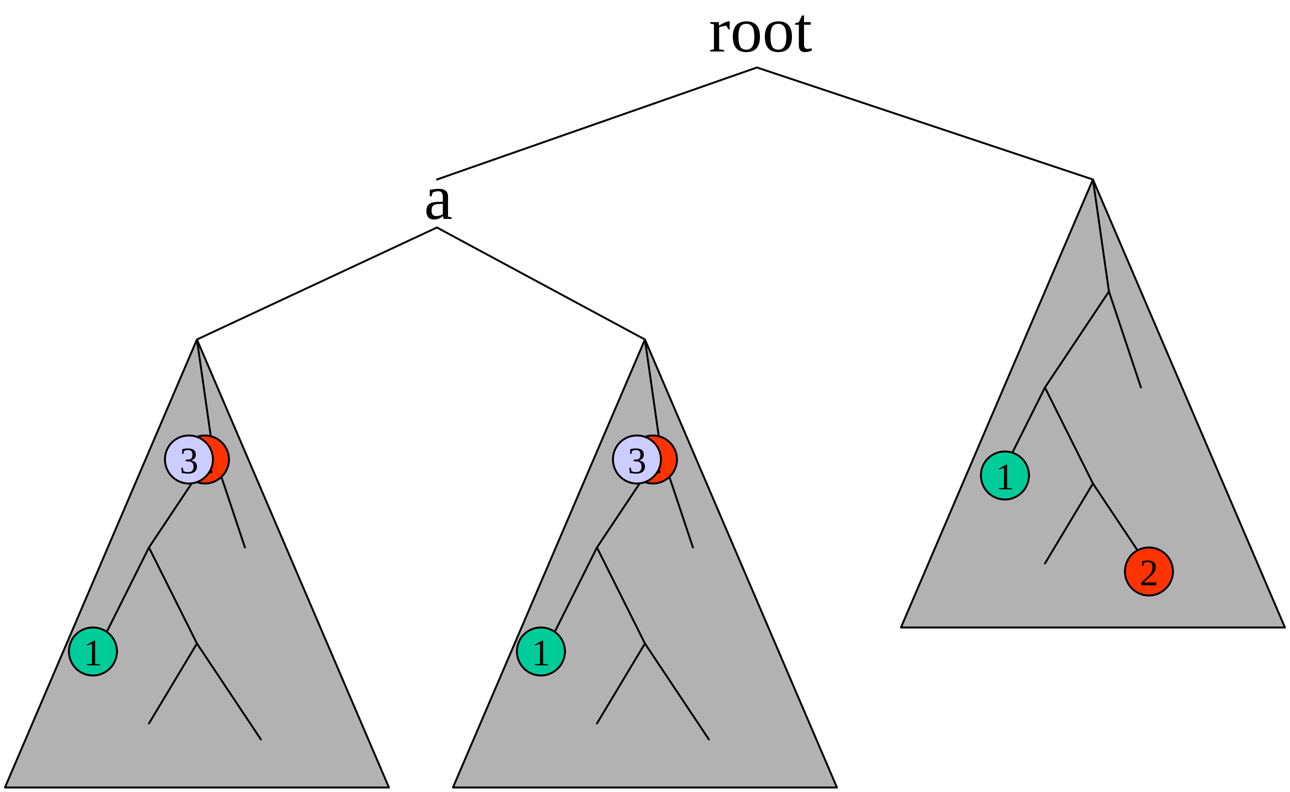
Output-producing transitions



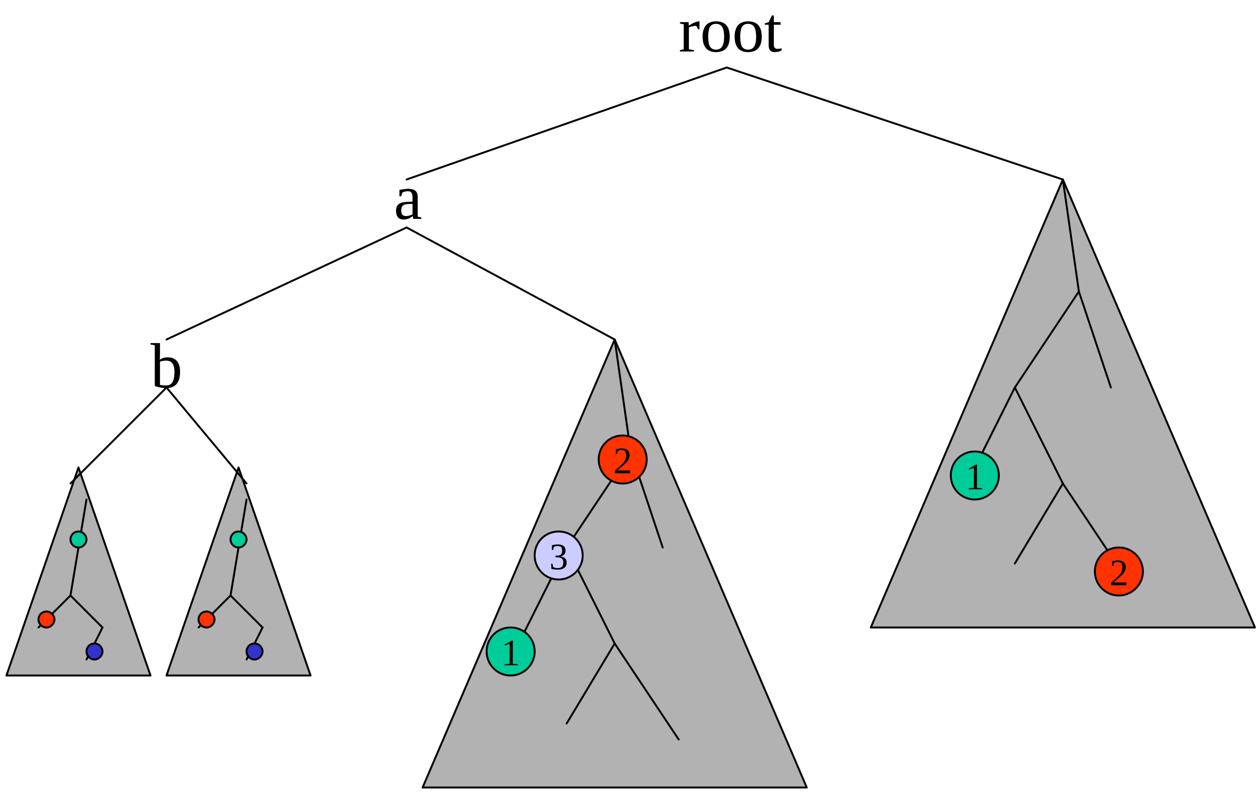
Output-producing transitions



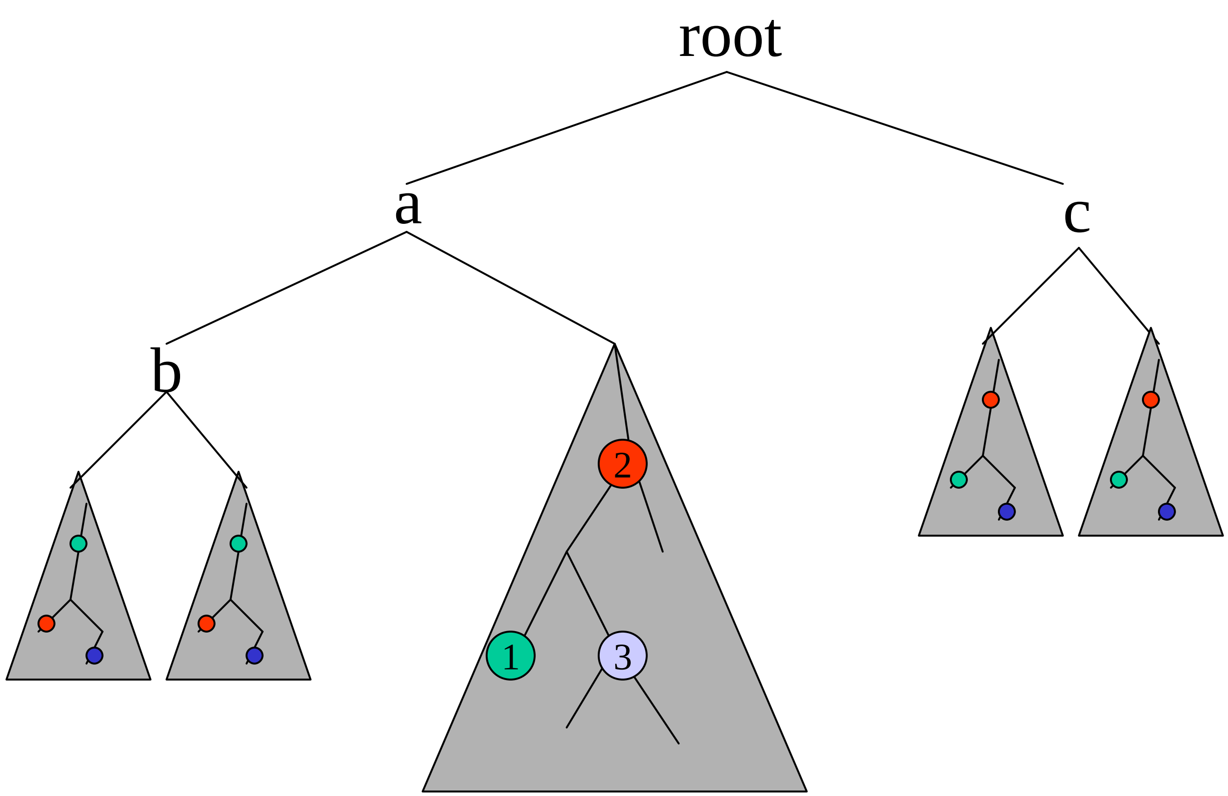
Output-producing transitions



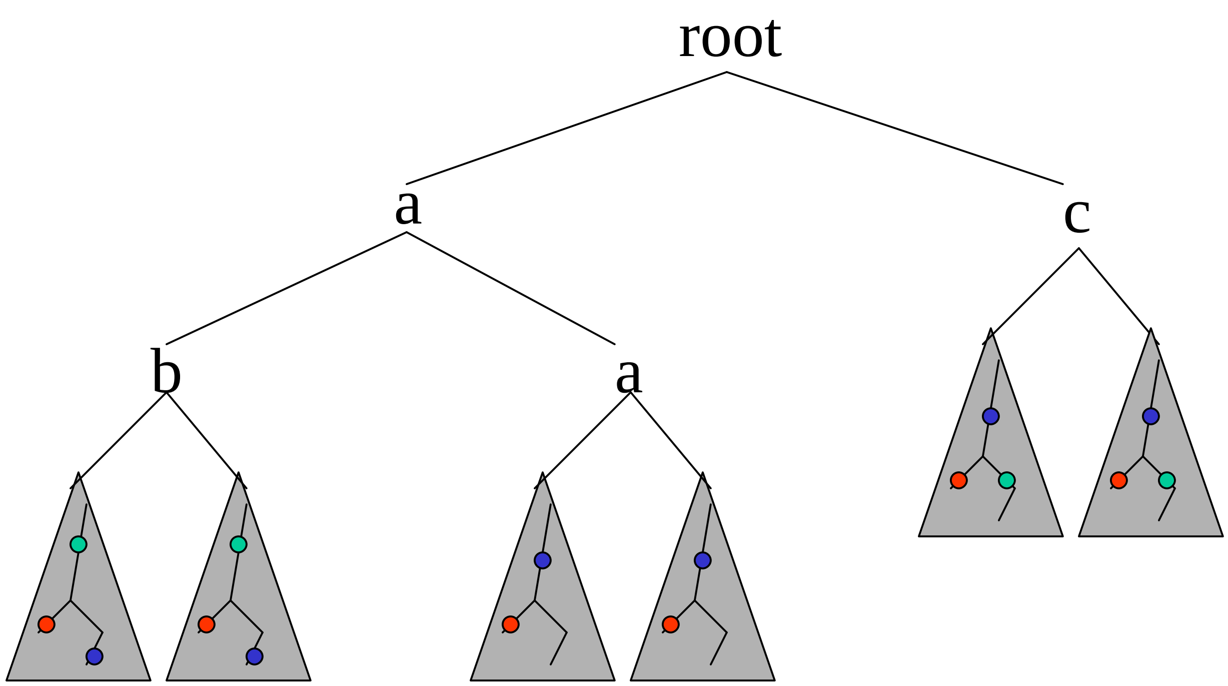
Output-producing transitions



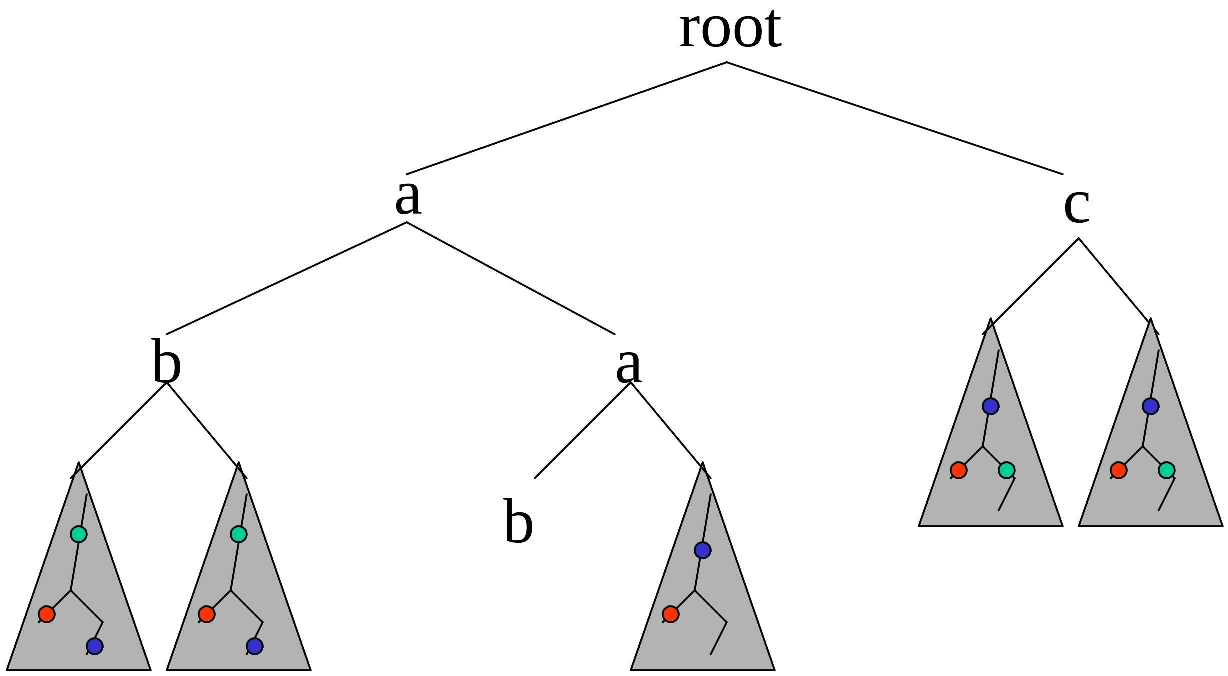
Output-producing transitions



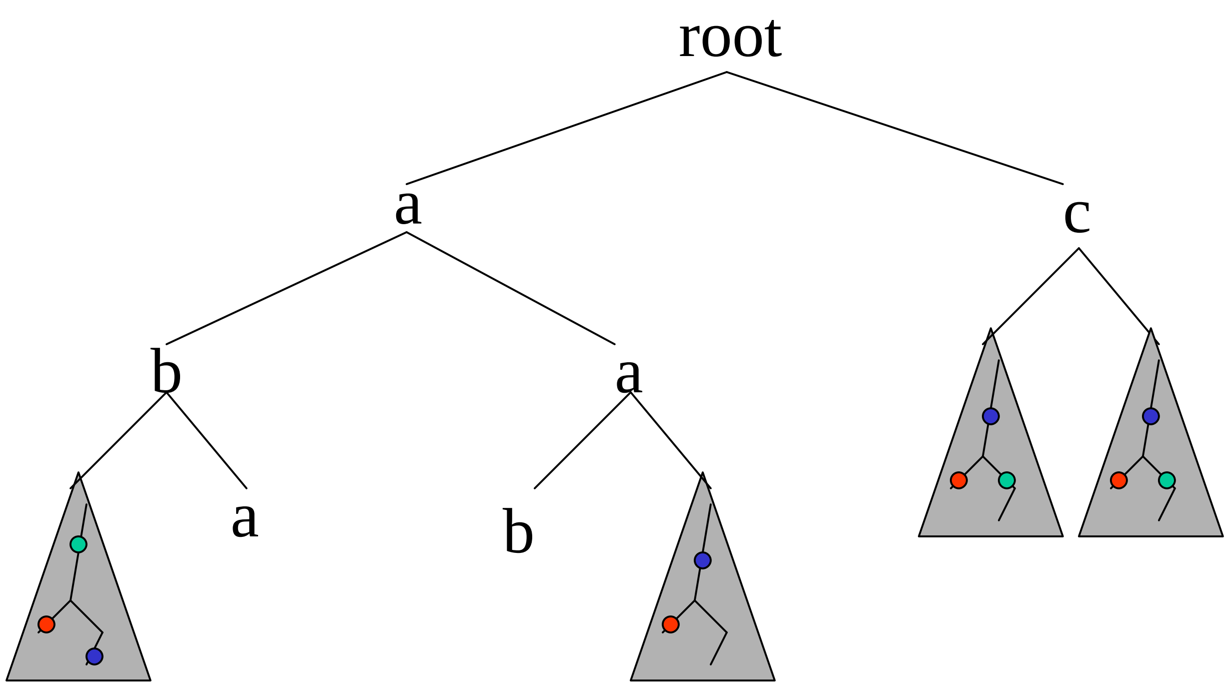
Output-producing transitions



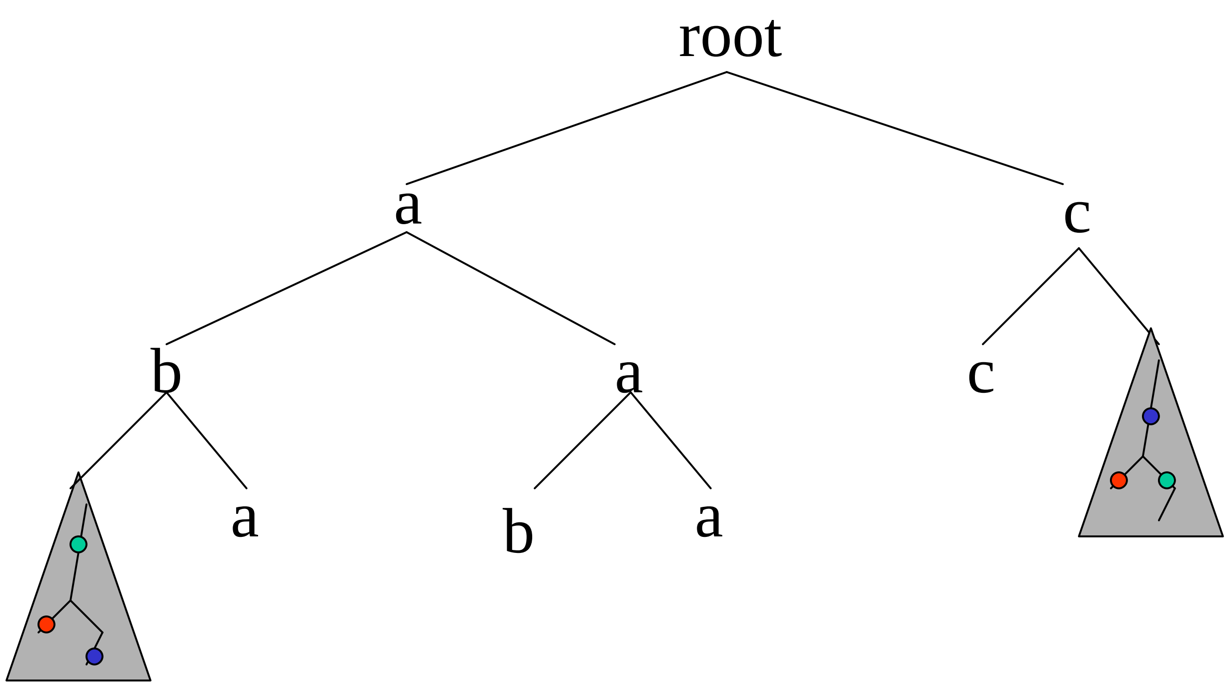
Output-producing transitions



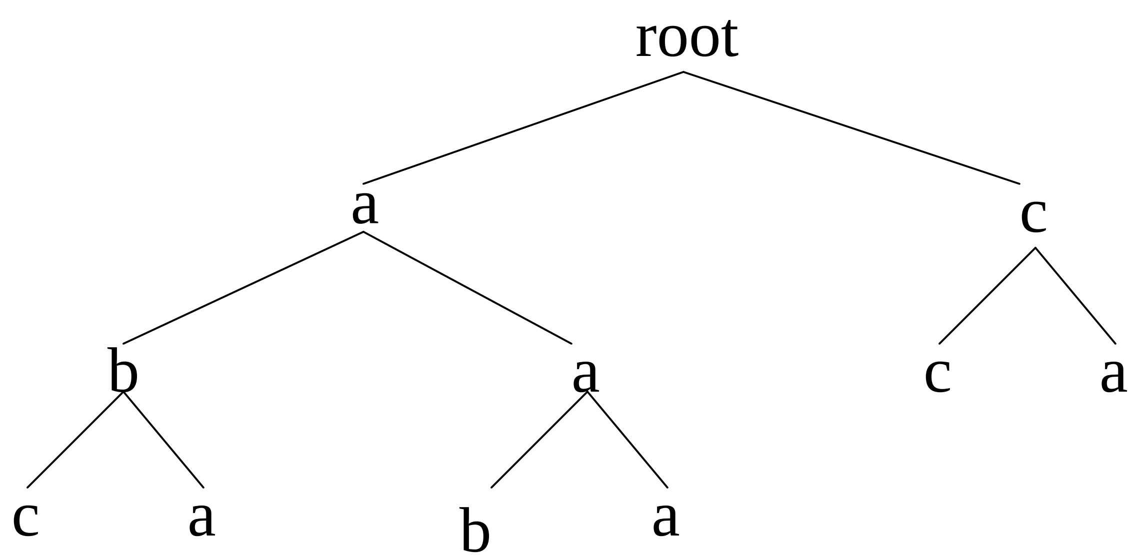
Output-producing transitions



Output-producing transitions



Output-producing transitions

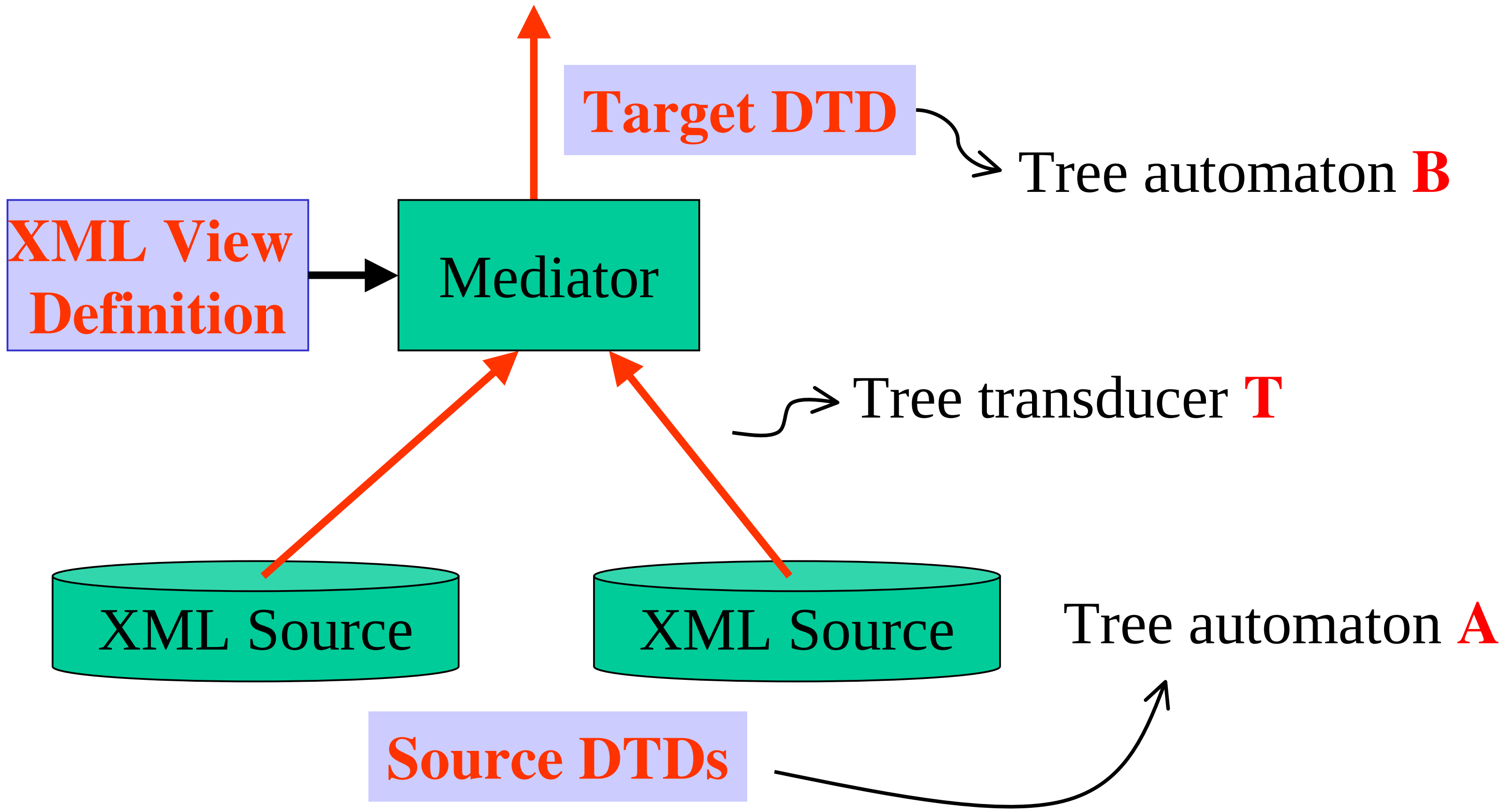


XML Query languages vs. k-pebble transducers

k-pebble transducers subsume the tree manipulation core of the common XML query languages:

XML-QL, XSLT, XQuery.

Application: Conformance Checking



Need to check: $T(A) \subseteq B$

Equivalently: $A \subseteq T^{-1}(B)$

Theorem: $\mathbf{T^{-1}(B)}$ is a regular tree language

[Milo+Suciu+V.-]

Conformance checking:

1. Compute from **T** and **B** the tree automaton for $\mathbf{T^{-1}(B)}$
2. Check that $\mathbf{A} \subseteq \mathbf{T^{-1}(B)}$

Caveat: no data joins

Connection to tree automata theory

- Recurring theme in recent research on XML
- Query automata [Neven+Schwentick]
  - define unary queries
- Attribute grammars [Neven+Van den Bussche]
  - FO inductions of linear depth
  - parallel RAMs with polynomially many processors

Connection to tree automata theory

- Feedback into automata theory

XML $\longrightarrow$ unranked ordered trees

data values $\longrightarrow$ infinite alphabets

Back in fashion
(courtesy XML-Schema)

- Extended context-free grammars
- Top-down deterministic tree automata
- 1-unambiguous regular expressions

Conclusion

- Logic has provided the foundations of databases, from relational databases all the way to XML
- FO lies at the core of relational database systems
- Logic underlies much of database theory
- XML and its query languages are closely related to tree automata, tree transducers, and logics on trees