FOUNDATIONS OF XML

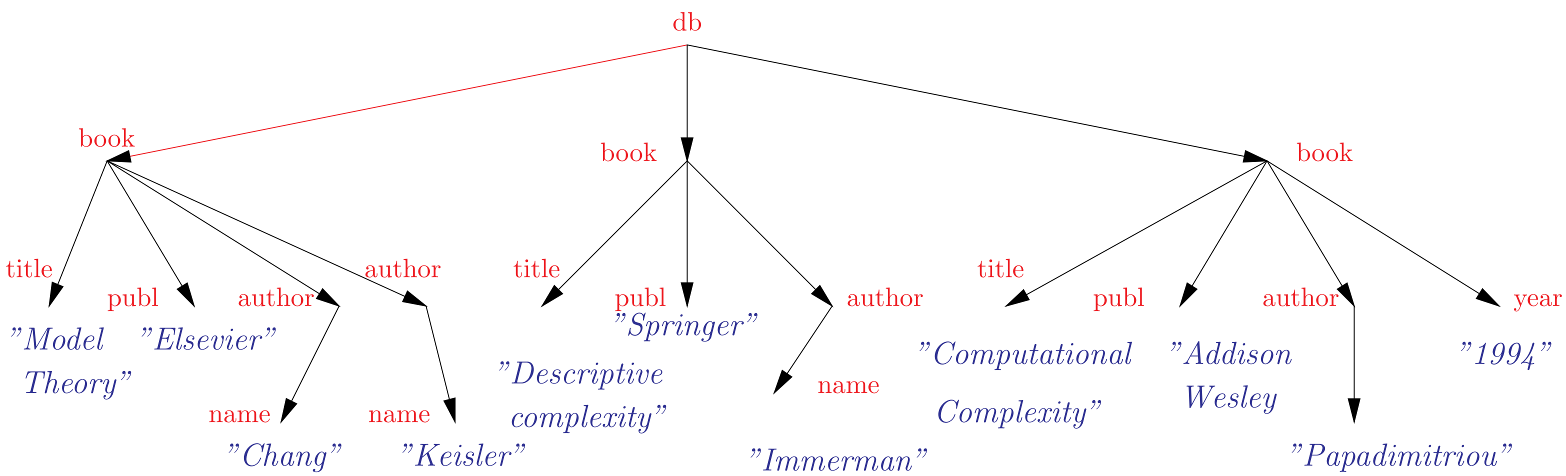
XML data

- XML = eXtensible Markup Language, the standard for exchanging data on the web.
- Has become one of the most common data formats.
- XML data is modeled as trees. In fact as *unranked* trees – we'll see soon what this means.
- W3C standards: XML Schema, XPath, XSLT, XQuery define types, navigation mechanism, transformations, and query languages for XML.
- Active work on XML in many communities, especially databases, information retrieval.
- Brings techniques (sometimes old and almost forgotten) from formal language theory and merges them nicely with logic.

XML documents look like this

```
<db>
  <book>
    <title attr_title="Model Theory"></title>
    <publisher publ_attr="Elsevier"></publisher>
    <author>
       <name name_attr="Change">
    </author>
    <author>
       <name name_attr="Keisler">
    </author>
  </book>
  <book>
    <title attr_title="Desciptive Complexity"></title>
    <publisher publ_attr="Springer"></publisher>
    <author>
       <name name_attr="Immerman">
    </author>
  </book>
  <book>
    <title attr_title="Computational Complexity"></title>
    <publisher publ_attr="Addison Wesley"></publisher>
    <year year_attr="1994"></year>
    <author>
       <name name_attr="Papadimitriou">
    </author>
  </book>
</db>
```

But we view them like this



Analogy: traditional databases

- Many ad hoc models before 1970:
  - hard to work with, hard to reason about
- 1970: Codd – relational model
  - data stored in relations
  - queried using a declarative language (e.g., relational calculus; syntactically SQL but the core is the same)
  - DBMS converts declarative queries into procedural queries that are optimized and executed
- Key advantages:
  - A clean mathematical model (based on logic)
  - Separation of declarative and procedural

XML development

- Clean model:
  - Structure – labeled unranked trees
  - Data – attributes
- Declarative languages (XPath, XQuery)
  - Flavor of traditional first-order logic or
  - Temporal logics – for describing navigation
- Procedural languages: automata-theoretic constructions
- Schemas: automata

Logic/automata connection

- Automata are a natural procedural counterpart of logic.
- All as occur before bs – a^*b^*:

$$\forall x \forall y\ Lab_a(x) \wedge Lab_b(y)\ \rightarrow\ x < y.$$

- The length is even – $\big((a|b)(a|b)\big)^*$

$$\exists X \left(\begin{array}{l} \quad \forall x(\mathit{first}(x) \rightarrow x \in X) \wedge (\mathit{last}(x) \rightarrow \neg X(x)) \\ \wedge\ \forall x\ (x \in X \rightarrow \mathit{successor}(x) \notin X) \\ \wedge\ \forall x\ (x \notin X \rightarrow \mathit{successor}(x) \in X) \end{array} \right)$$

- $\exists X$ – quantification over sets of positions.
- MSO — Monadic Second-Order logic – extension of first-order logic (relational calculus) with such quantifiers.

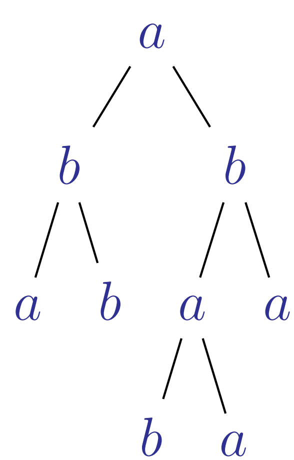
Logic/automata connection

- **Theorem** (Büchi, 1959) MSO-definable = Regular word languages.
- **Theorem** (Thatcher/Wright, late 60s): The same is true for finite binary trees.
- **Theorem** (Rabin 1970): The same is true for infinite binary trees.
- Initially these results were developed to prove decidability of logical theory.
- Sentences in a theory are converted into automata; then satisfiability is the nonemptiness problem for automata.
- These are easier to prove decidable.
- The ultimate result: decidability of S2S, monadic theory of the infinite binary tree. Almost everything else decidable can be encoded in this theory.

Ranked vs Unranked Trees

Typically in CS one works with ranked trees; e.g., binary, ternary, etc trees.

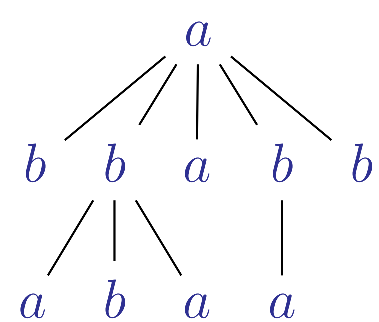
A binary tree:



Unranked trees

But for XML we need unranked trees.

In them, nodes can have arbitrarily many children, and different nodes may have different number of children.



We look at logic(s) for XML

Why?

- XML documents describe data.
- Standard relational database approach:
  - data model – relations
  - declarative languages for specifying queries
  - procedural languages for evaluating queries
- Standard declarative languages are all logic-based:
  - relational calculus = first-order logic (FO)
  - datalog = fragment of fixed-point logic
  - basic SQL = FO with counting

What does XML add?

- New logics.
- New procedural languages:
  - logic–automata connection.

What do logics do?

Most commonly they define:

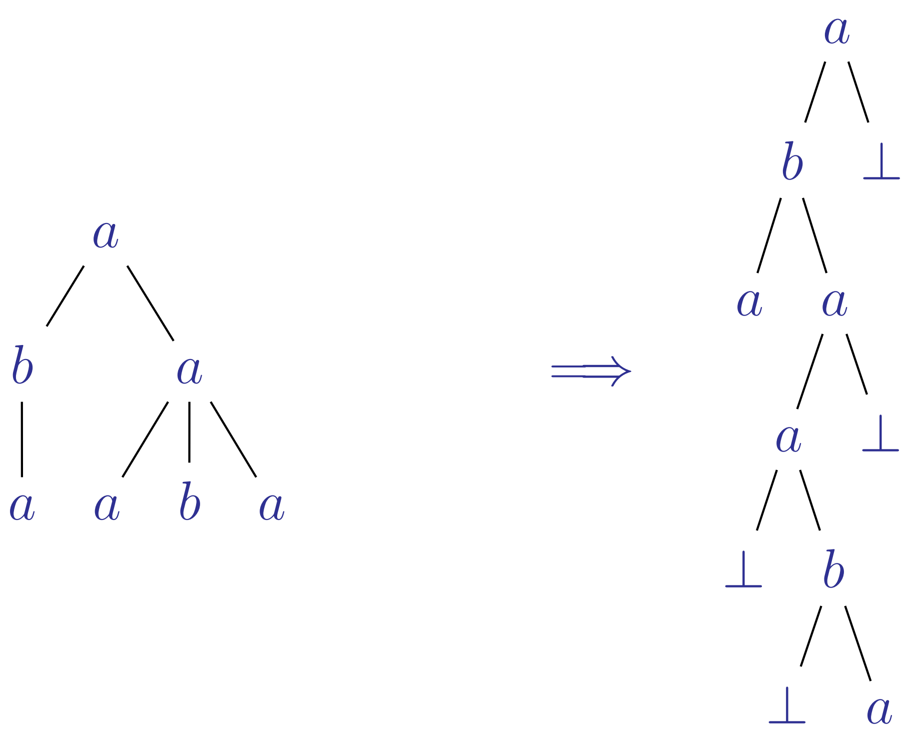
- Boolean (that is, yes/no) queries:
  - DTD conformance
  - Existence of certain paths
- Unary queries which select nodes in trees:
  - all nodes reachable by a certain path from the root;
  - all nodes with a certain label or certain data element.

Commonalities between logics

- (Almost) all have associated automata models.
- Crucial aspect is navigation.
- Hence we often see close connection with temporal logics.
- Logics are specifically designed for unranked trees.

Ranked/Unranked Connection

(used by Rabin in 1970 to interpret SωS in S2S):



It preserves recognizability by automata, MSO-definability, FO-definability...

Why not just use it?

- Instead of defining logics for unranked trees, just translate them into binary trees and use known logical formalisms.
- Problem: hard to navigate!
- A path in a translation becomes a union of arbitrarily many child paths and sibling paths.
- Hard (at least not natural) to express many properties such as DTD conformance.

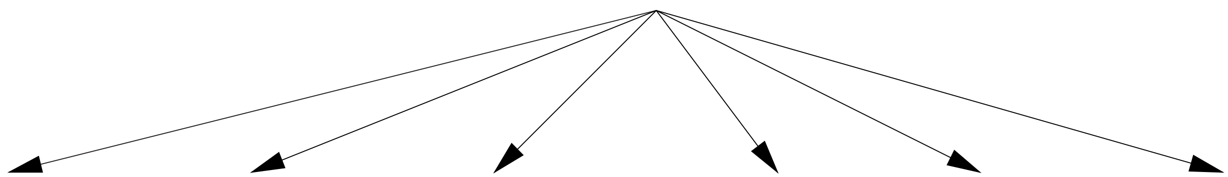
Classification: Yardstick logic

Most logics are based either on FO or MSO.

- FO:
  - Boolean connectives $\vee, \wedge, \neg$,
  - quantifiers $\exists x,\ \forall x$ ranging over nodes of trees.
- MSO: in addition
  - quantifiers $\exists X,\ \forall X$ ranging over sets of nodes;
  - plus new formulae $x \in X$.

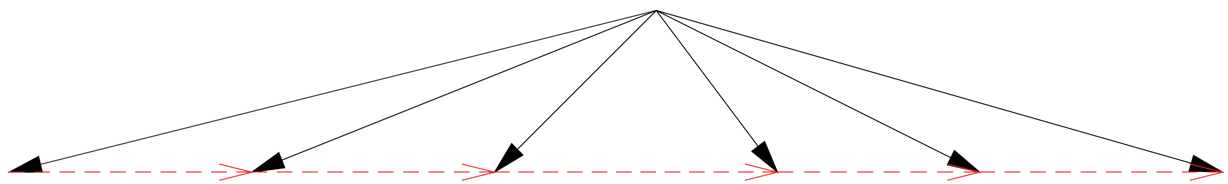
Classification: Ordered vs Unordered Trees

In unordered trees, there is no order among siblings (children of the same node).



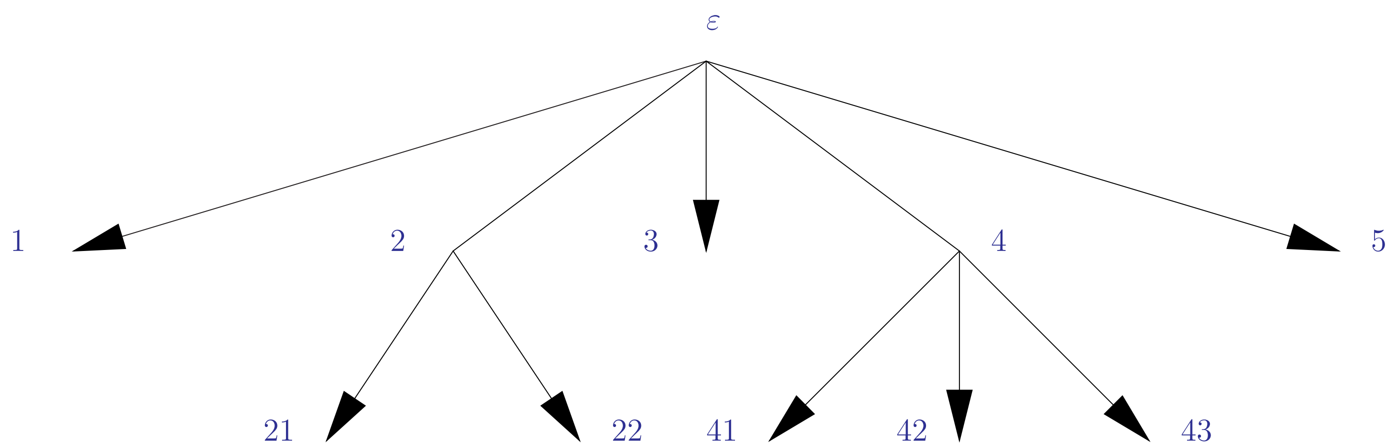
Classification: Ordered vs Unordered Trees

In unordered trees, there is no order among siblings (children of the same node).



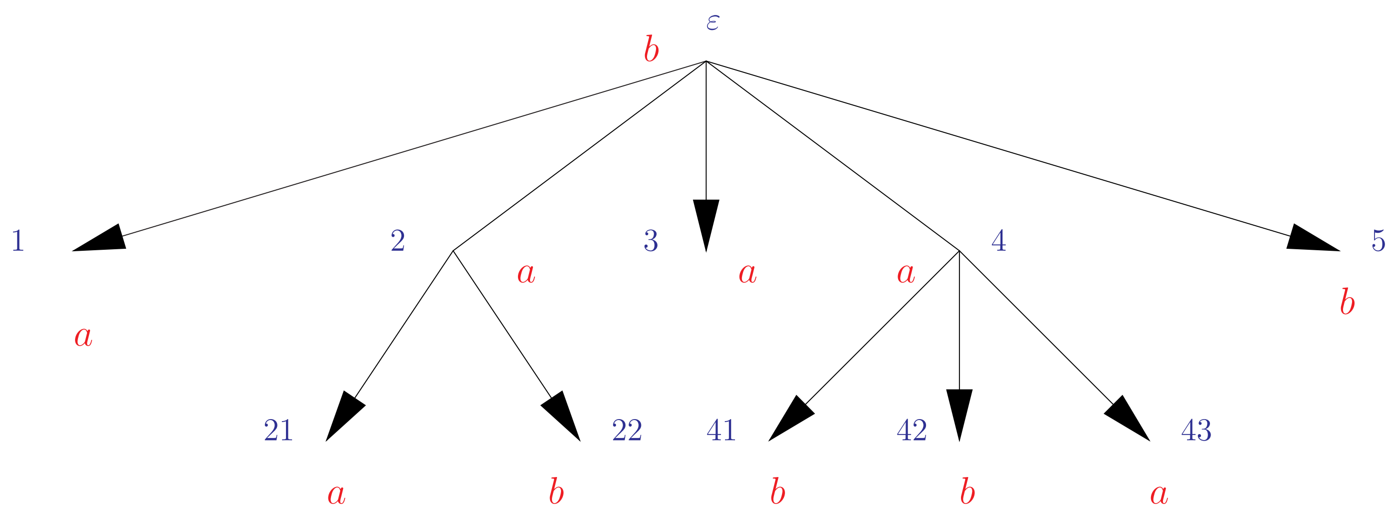
In ordered trees, siblings are ordered (from the oldest to the youngest).

Formal definition of unranked trees



Tree domain: prefix-closed subset D of $\mathbb{N}^*$ such that $s \cdot i \in D$ implies $s \cdot j \in D$ for $j < i$.

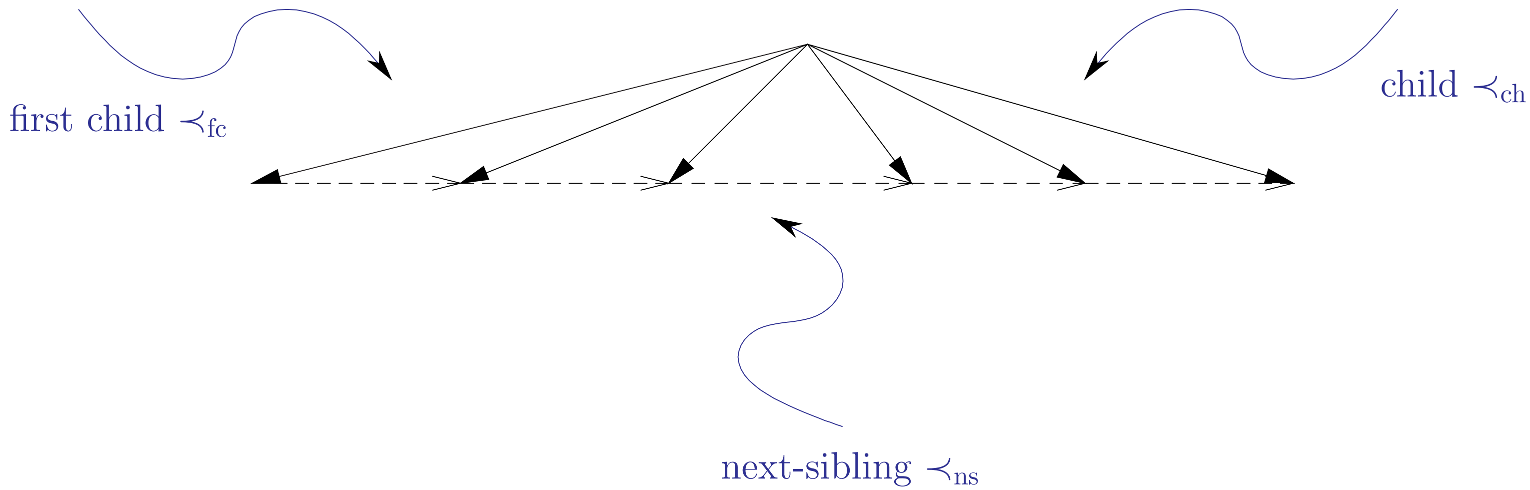
Formal definition of unranked trees



Tree domain: prefix-closed subset D of $\mathbb{N}^*$ such that $s \cdot i \in D$ implies $s \cdot j \in D$ for $j < i$.

Tree over finite alphabet Σ: tree domain plus a mapping from it to Σ.

Basic predicates



Transitive closures:
- $\prec_{\mathrm{ch}}^*$ of $\prec_{\mathrm{ch}}$ (descendant)
- $\prec_{\mathrm{ns}}^*$ of $\prec_{\mathrm{ns}}$ (younger child)

We normally use transitive closures (since they are not definable in FO).

For MSO, we can use either $\prec_{\mathrm{ch}}$, $\prec_{\mathrm{ns}}$ or $\prec_{\mathrm{ch}}^*$, $\prec_{\mathrm{ns}}^*$ as they are inter-definable.

LOGICS FOR ORDERED TREES

Logic/automata connection

A set $\mathcal{T}$ of trees is definable in a logic $\mathcal{L}$ iff there is a sentence φ of $\mathcal{L}$ such that

$$T \in \mathcal{T} \quad \Leftrightarrow \quad T \models \varphi$$

A set $\mathcal{T}$ of trees is regular if it is recognizable by a tree automaton.

Theorem

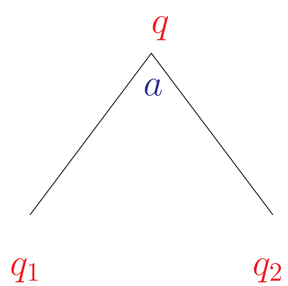
- A set of binary trees is regular iff it is MSO-definable (Thatcher-Wright, 1966).
- A set of unranked trees is regular iff it is MSO-definable (Thatcher 1967 $\longrightarrow$ forgotten $\longrightarrow$ folklore, republished many times)

Tree automata: the ranked case

Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.

Tree automata: the ranked case

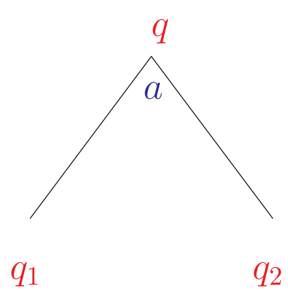
Transitions are δ : $\mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.

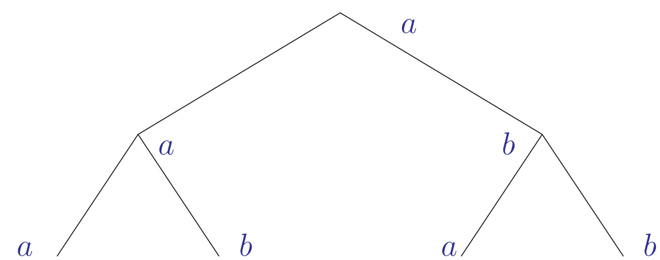


if $q \in \delta(q_1, q_2, a)$

Tree automata: the ranked case

Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.

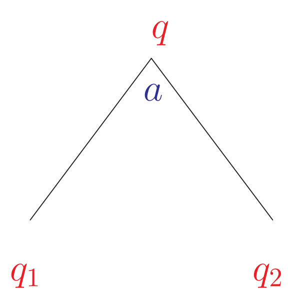


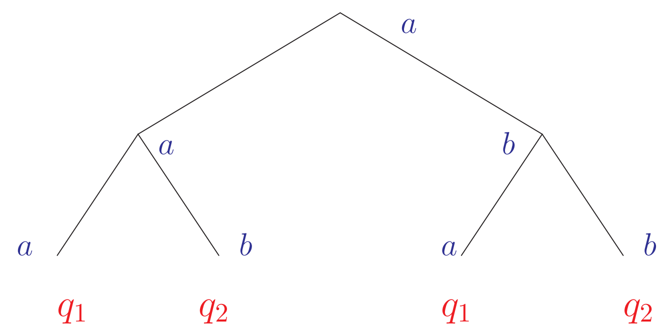


if $q \in \delta(q_1, q_2, a)$

Tree automata: the ranked case

Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.

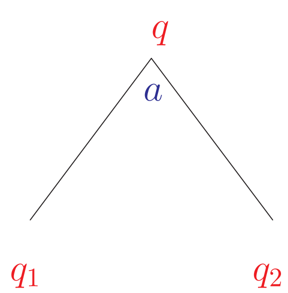




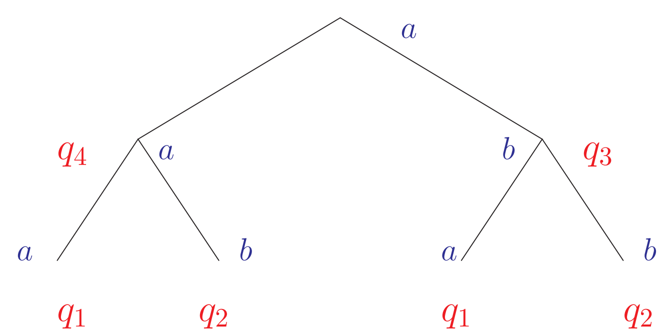
if $q \in \delta(q_1, q_2, a)$

Tree automata: the ranked case

Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.

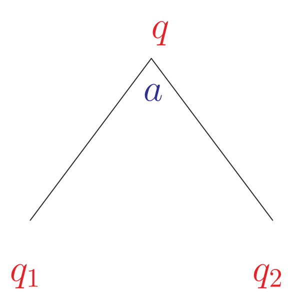


if $q \in \delta(q_1, q_2, a)$

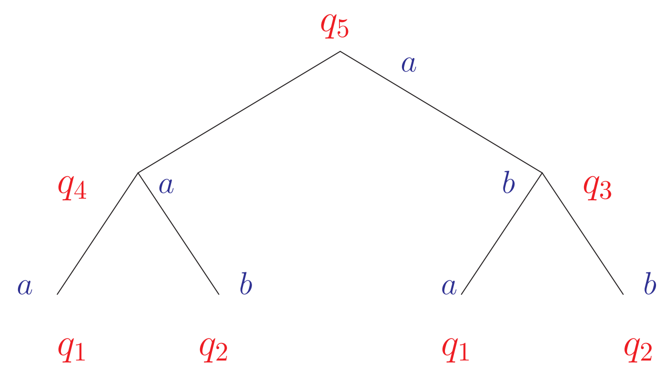


Tree automata: the ranked case

Transitions are δ : States $\times$ States $\times\ \Sigma \longrightarrow 2^{\text{States}}$.

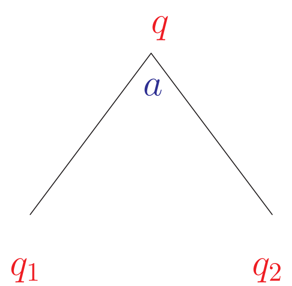


if $q \in \delta(q_1, q_2, a)$

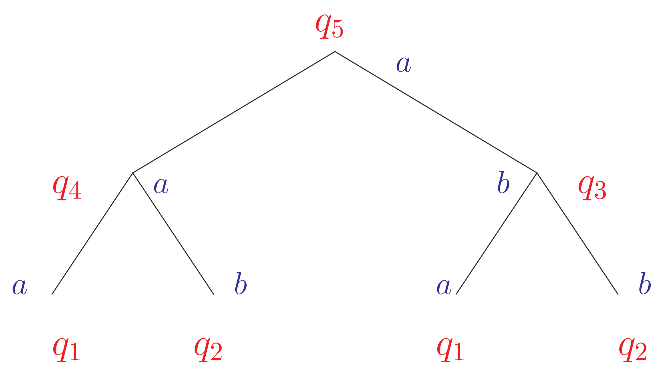


Tree automata: the ranked case

Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.



if $q \in \delta(q_1, q_2, a)$



accepted if q_5 is a final state

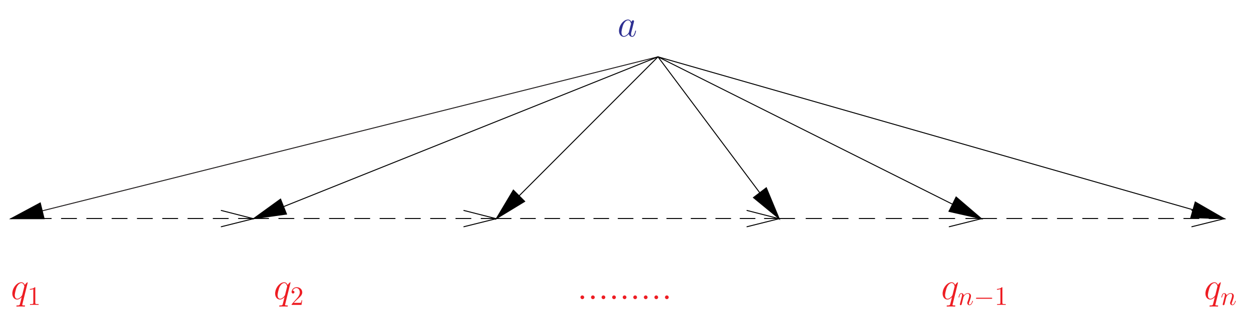
Tree automata: unranked case

Transitions are $\delta : \mathsf{States} \times \Sigma \to 2^{\mathsf{States}^*}$ so that each $\delta(q, a) \subseteq \mathsf{States}^*$ is a regular language.

Tree automata: unranked case

Transitions are δ : States $\times \Sigma \longrightarrow 2^{\text{States}^*}$ so that each $\delta(q,a) \subseteq$ States* is a regular language.

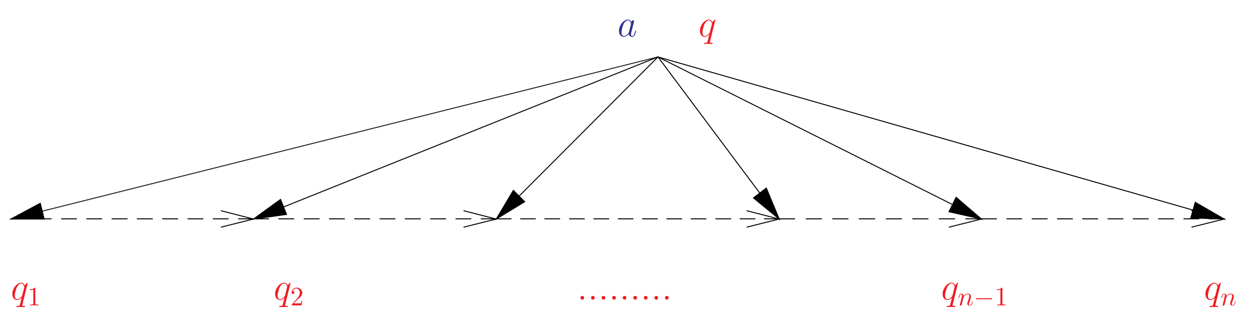
The run is the same as before:



Tree automata: unranked case

Transitions are $\delta : \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}^*}$ so that each $\delta(q, a) \subseteq \mathsf{States}^*$ is a regular language.

The run is the same as before:

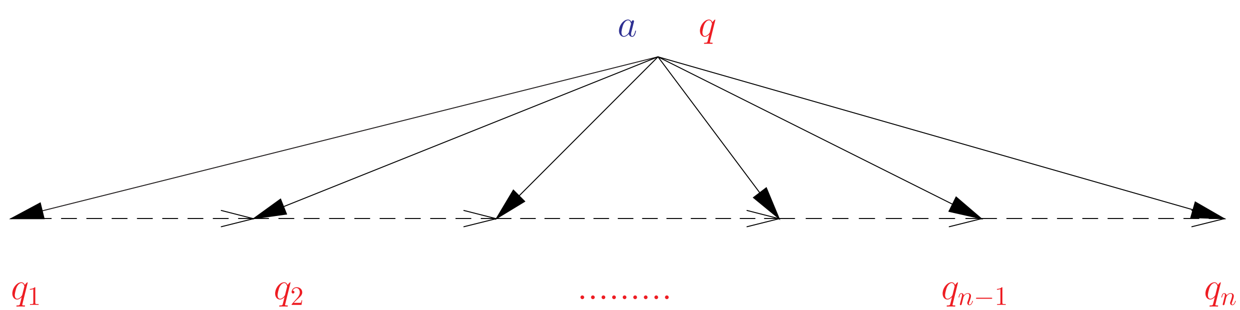


if $q_1 \cdots q_n \in \delta(q, a)$

Tree automata: unranked case

Transitions are δ : States $\times \Sigma \longrightarrow 2^{\text{States}^*}$ so that each $\delta(q,a) \subseteq$ States* is a regular language.

The run is the same as before:



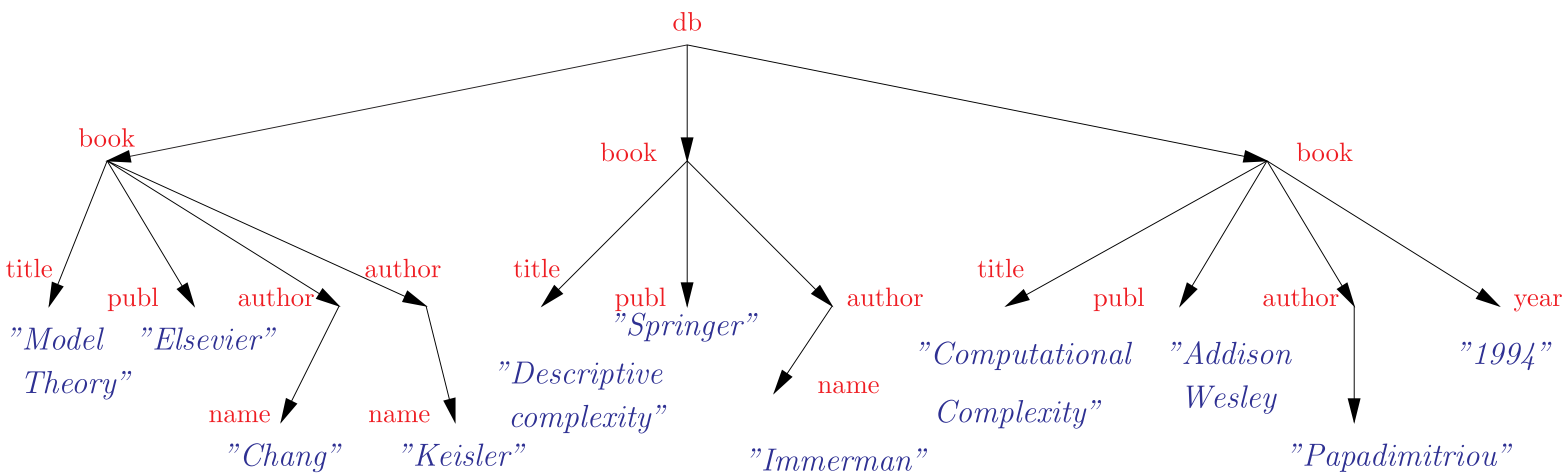
if $q_1 \cdots q_n \in \delta(q,a)$

A tree is accepted if there is a run such that the root is assigned an accepting state.

Properties of tree automata

- Languages defined by automata are called regular.
- Regular languages are closed under:
  - union ($O(n)$ algorithm)
  - intersection ($O(n^2)$ algorithm – product construction)
  - complementation ($2^{O(n)}$ algorithm – powerset construction)
- Binary case: equivalent to deterministic automata
  - for unranked it's a bit tricky to say what "deterministic" means
- Nonemptiness problem: Given an automaton $\mathcal{A}$, does it accept a tree?
  - solvable in polynomial time (naively in $O(n^3)$), same as for context-free grammars

Automata and XML schemas



Document description (DTD = Document Type Definition)

$$
\begin{aligned}
\text{db} &\longrightarrow \text{book}^* \\
\text{book} &\longrightarrow \text{title, publ, author}^+\text{, year?} \\
\text{author} &\longrightarrow \text{name}
\end{aligned}
$$

MSO and DTDs

- DTDs have rules such as

$$\text{book} \quad \longrightarrow \quad \text{title}, \text{publ}, \text{author}^{+}, \text{year}?$$

- Not a new invention: known for ages as *extended context-free grammars*, i.e., CFGs in which right hand sides of productions can be arbitrary regular expressions.
- Since regular string languages are precisely those MSO-definable, it follows that all DTDs are MSO-definable.
- Are DTDs and MSO equal?
- The answer is negative.

MSO and DTDs cont'd

- EDTDs = Extended DTDs: these are DTDs over a larger alphabet $\Sigma' \supseteq \Sigma$ together with a projection $\pi : \Sigma' \to \Sigma$.
- Trees over Σ that conform to an EDTD: projections of conforming trees over Σ'

Theorem (Thatcher 1967; rediscovered several times recently)

$$\text{EDTDs} \quad = \quad \text{MSO}$$

- Led to a key addition in XML Schema: specialization
- Essentially a non-context-free feature

Unary queries

- A unary query selects a set of nodes in a tree.
- A surprisingly simple automaton model captures them.
- Query automaton:

$$\text{QA} = \text{unranked tree automaton} + \text{selecting set } S \subseteq \text{States}.$$

- QA selects a node s from a tree T if there is an accepting run that assigns a state q to s such that

$$q \in S.$$

Theorem For unary queries over unranked trees,

$$\text{Query Automata} \quad = \quad \text{MSO}.$$

MSO over trees: Complexity

- Evaluating queries:

INPUT:	tree T, sentence φ
QUESTION:	Is $T \models \varphi$?

- Or could be: $T \models \psi(a)$ for a unary formula $\psi(x)$.
- Two ways:
  - $\|T\|$ is the only input: data complexity
  - both $\|T\|$ and $\|\varphi\|$ are inputs: query complexity

MSO over trees: Complexity cont'd

- By translation to automata:
  - Every MSO sentence φ can be transformed into an automaton $\mathcal{A}_\varphi$
  - To run $\mathcal{A}_\varphi$ on T – linear time in the size of T.
- Hence the data complexity of MSO is linear.

MSO over trees: Complexity cont'd

- But what what about query complexity?
- How big can $\mathcal{A}_\varphi$ be in terms of T?
- Answer: non-elementary.
- Recall: in recursion theory, elementary means a function

$$f(n) \quad < \quad 2^{2^{\cdot^{\cdot^{\cdot^{n}}}}} \Big\}\ell \text{ times} \quad \text{for a fixed } \ell.$$

- Dates back to the "optimistic" 1950s when those functions were viewed as relatively "simple".
- For the size of $\mathcal{A}_\varphi$, the number ℓ may depend on φ.

MSO over trees: Complexity cont'd

- These "towers of 2s" grow very fast:

$$\textsc{Tower}(0) = 1 \qquad \textsc{Tower}(n+1) = 2^{\textsc{Tower}(n)}$$

- $\textsc{Tower}(5)$ exceed the number of atoms in the visible universe.
- $\textsc{Tower}(6)$ exceed the number of atoms in the universe.
- Hence impractical...
- An even bigger problem: if we keep data complexity linear, the query complexity is necessarily non-elementary (even if we manage to avoid automata – Frick/Grohe, 2002)
- Can we do better?
- Yes, by finding different logics that have the power of MSO, and yet better model-checking properties.

Changing syntax to lower complexity: LTL

Syntax:

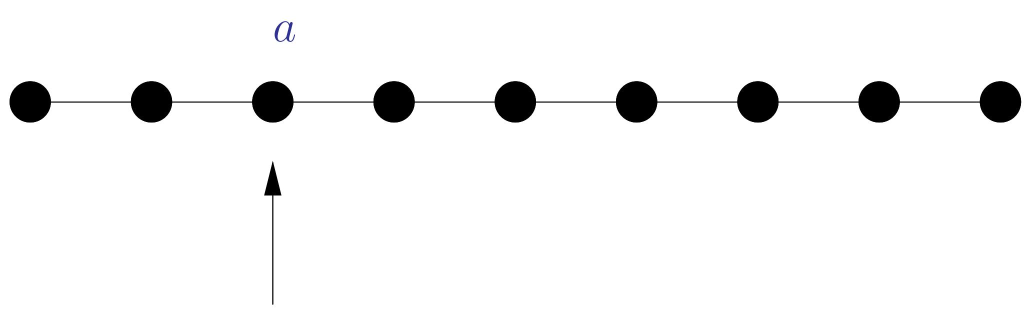
$$\varphi \ := \ a(\in \Sigma) \ \mid \ \varphi \vee \varphi' \ \mid \ \neg\varphi \ \mid \ \mathbf{X}\varphi \ \mid \ \varphi\mathbf{U}\varphi'$$

Changing syntax to lower complexity: LTL

Syntax:

$$\varphi \ := \ a(\in \Sigma) \ \mid \ \varphi \vee \varphi' \ \mid \ \neg\varphi \ \mid \ \mathbf{X}\varphi \ \mid \ \varphi \mathbf{U} \varphi'$$

Semantics:



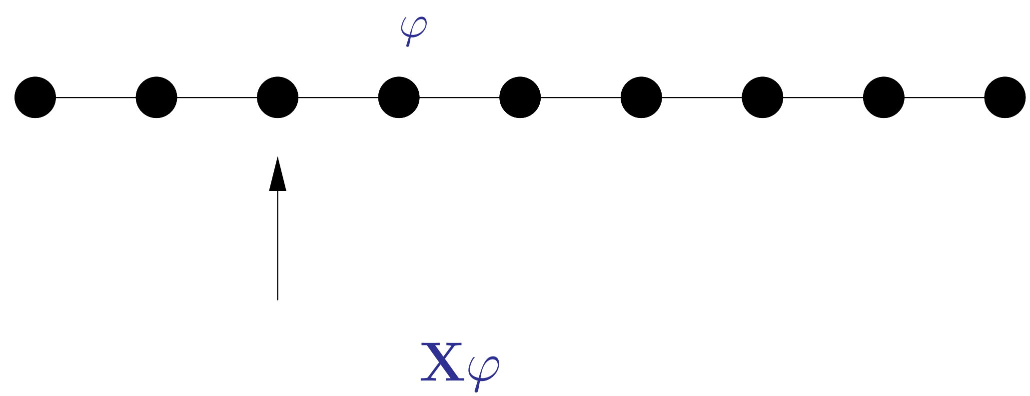
$$a, a \in \Sigma$$

Changing syntax to lower complexity: LTL

Syntax:

$$\varphi \;:=\; a(\in \Sigma) \;\mid\; \varphi \vee \varphi' \;\mid\; \neg\varphi \;\mid\; \mathbf{X}\varphi \;\mid\; \varphi\mathbf{U}\varphi'$$

Semantics:

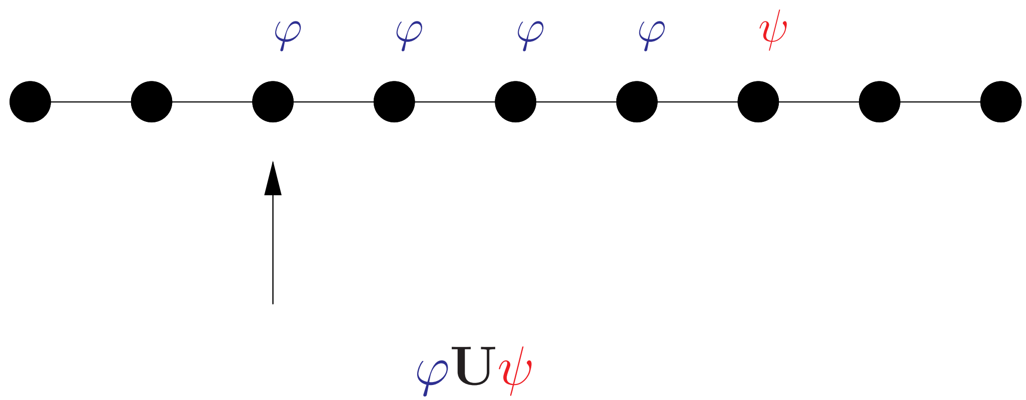


Changing syntax to lower complexity: LTL

Syntax:

$$\varphi \ := \ a(\in \Sigma) \ \mid \ \varphi \vee \varphi' \ \mid \ \neg\varphi \ \mid \ \mathbf{X}\varphi \ \mid \ \varphi \mathbf{U} \varphi'$$

Semantics:



LTL cont'd

- LTL $=$ FO over strings (Kamp's theorem).
- To evaluate FO with linear data complexity, one needs non-elementary query complexity
(modulo some complexity-theoretic assumptions; Frick/Grohe 2002)
- But LTL over strings can be evaluated in time

$$O(\|\varphi\| \cdot |s|)$$

- Of course this implies that translation from FO to LTL is non-elementary.

A good and practical logic for XML – Monadic Datalog

- Datalog $=$ database query language; essentially extension of positive FO with least fixed point. (Transitive closure example – board.)
- Can also be viewed as prolog without function symbols.
- Datalog program is monadic if all introduced predicates (intensional predicates) are monadic – have one free variable.
- Example: select (in predicate D) all nodes s such that all their descendants (including s) are labeled a:

$$\begin{array}{rcllll} D(x) & :- & P_a(x), & \mathsf{Leaf}(x) & & \\ D(x) & :- & P_a(x), & x \prec_{\mathrm{fc}} y, & S(y) & \\ S(y) & :- & P_a(x), & \mathsf{LastChild}(y), & D(y) & \\ S(y) & :- & P_a(x), & x \prec_{\mathrm{ns}} y, & S(y), & D(y) \end{array}$$

Monadic Datalog cont'd

Assume that Leaf and LastChild are available as basic predicates.

Then:

- Monadic Datalog $=$ MSO
- A Monadic Datalog program $\mathcal{P}$ can be evaluated on a tree T in time

$$O(\|\mathcal{P}\| \cdot \|T\|)$$

This is heavily used in Web data extraction: real-life languages are based on monadic datalog, which combines expressiveness and very good evaluation properties.

μ-calculus over unranked trees

- μ-calculus (Kozen 82): extension of a temporal logic with the least fixed point operator.
- Subsumes many logics used in verification: LTL, CTL, CTL*.
- Syntax:

$$\varphi \ := \ S \mid a \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid \neg\varphi \mid \mathbf{X}_E\varphi \mid \mu S\ \varphi(S)$$

  - S must occur positively;
  - E ranges over relations $\prec_{\rm ch}$ and $\prec_{\rm ns}$
- Full μ-calculus: one can talk about the past.
  - That is, E also ranges over inverses of $\prec_{\rm ch}$ and $\prec_{\rm ns}$

μ-calculus over unranked trees cont'd

Over unranked trees:

$$\text{full } \mu\text{-calculus} \quad = \quad \text{MSO}$$

For Boolean queries:

- MSO $=$ alternation-free μ-calculus (all negations pushed to atomic formulae)
- Complexity of model-checking
  - $O(\|\varphi\|^2 \cdot \|T\|)$ for μ-calculus;
  - $O(\|\varphi\| \cdot \|T\|)$ for alternation-free μ-calculus.

First-Order based formalisms

- These are often studied in connection with XPath
- XPath – a W3C standard, essentially the navigation language for XML.

XPath – an informal introduction

- XPath has two kinds of formulae: node tests and path formulae.
- Node tests are closed under Boolean connectives and can check if a path satisfying a path formula can start in a given node.
- Path formulae can:
  - test if a node test is true in the first node of a path;
  - test if a path starts by going to a child, first child, next child, previous child, parent, descendant, ancestor, etc;
  - take union or composition of two paths.

Example: `//book[/author[name="Keisler"]]/title`
gives titles of books coauthored by Keisler.

CTL* vs XPath

- There is a well-known logic, CTL*, that similarly combines node (called *state*) and path formulae.
- Syntax:

state formulae	$\alpha \ := \ a \mid \alpha \vee \alpha' \mid \neg\alpha \mid \mathbf{E}\beta$
path formulae	$\beta \ := \ $ LTL over state formulae

Example: all descendants of a given node (including self) are labeled a (with $\Sigma = \{a, b\}$):

$$\neg\mathbf{E}\left((a \vee b) \ \mathbf{U} \ b\right)$$

CTL^* and FO over trees

With respect to Boolean queries, over both binary and unranked trees, $\mathrm{CTL}^* = \mathrm{FO}$

For unary queries, one adds reasoning about the past (temporal operators $\mathbf{Y}$ – yesterday, and $\mathbf{S}$ – since).

Linear-time logic on trees

- Defined by Schlingloff in 1992
- Rediscovered, modified and used for proving results about XPath by Marx 2004
- Various names: $\mathcal{X}_{until}$ by Marx, $\{\mathcal{S},\mathcal{U}\} \cup \{\mathcal{X}_k \mid k < \omega\}$ by Schlingloff
- We call it $\mathsf{TL}^{\mathsf{tree}}$

Linear-time tree logic $\mathbf{TL}^{\mathbf{tree}}$

Syntax:

$$\varphi \ := \ a(\in \Sigma) \ | \ \varphi \vee \varphi' \ | \ \neg\varphi \ | \ \mathbf{X}_*\varphi \ | \ \varphi\mathbf{U}_*\varphi' \ | \ \mathbf{Y}_*\varphi \ | \ \varphi\mathbf{S}_*\varphi'$$

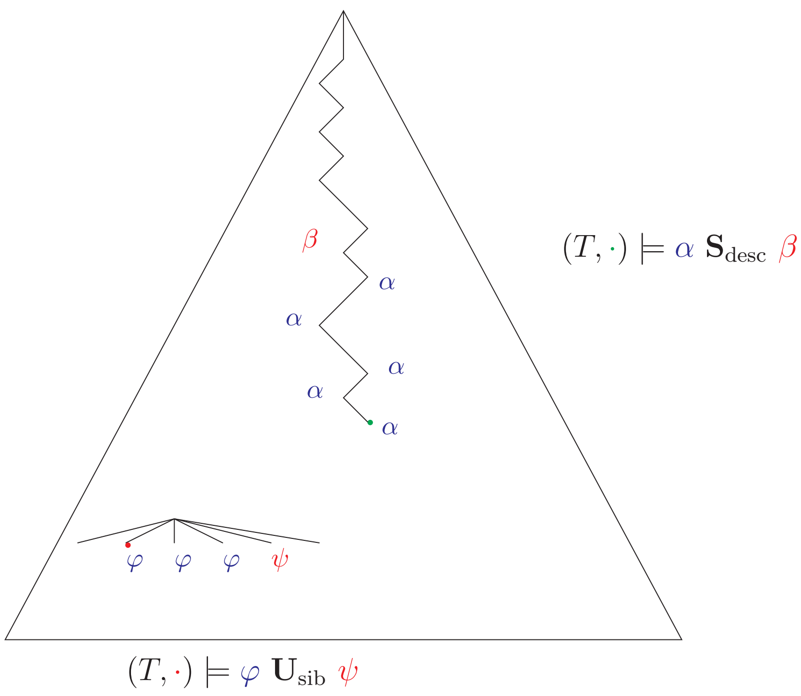
where * is either desc or sib.

Linear-time tree logic $\mathsf{TL}^{\mathsf{tree}}$

Syntax:

$$\varphi \;:=\; a(\in \Sigma) \;\mid\; \varphi \vee \varphi' \;\mid\; \neg\varphi \;\mid\; \mathbf{X}_*\varphi \;\mid\; \varphi\mathbf{U}_*\varphi' \;\mid\; \mathbf{Y}_*\varphi \;\mid\; \varphi\mathbf{S}_*\varphi'$$

where * is either desc or sib.



Linear-time tree logic $\mathsf{TL}^{\mathsf{tree}}$

Theorem (Marx '04; Schlingloff '92)
Over ordered unranked trees,

$$\mathsf{TL}^{\mathsf{tree}} \; = \; \mathsf{FO}$$

Application: Static XML reasoning

- Documents and constraints
- Constraints and DTDs
- XPath satisfiability (with DTDs)
- XPath containment (query optimization, more generally)
- Properties of updates
- Properties of schema mappings
- Security guarantees provided by views
- . . . and many others.

XML reasoning: logics+automata

- We need to combine logics that have a temporal flavor and automata.
- This is at the core of many software and hardware verification techniques (aka model checking) that have been implemented in industrial strength products and are widely used (NASA, Intel, Cadence, NEC, Synopsis etc)

XML reasoning: logics+automata

- Logic-automata translations:
  - LTL to nondeterministic or alternating Büchi automata
  - $\left\{ \begin{array}{l} \text{PDL} \\ \text{CTL} \\ \mu\text{-calculus} \end{array} \right\}$ to (subclasses of) tree automata
  - call-return logics to visibly pushdown automata, etc
- So to reason about XML, one can combine:
  - a logical specification (e.g. navigation)
  - an automaton specification (e.g. a schema) and
  - a logic-automata translation

Reasoning task: example I – XPath satisfiability

- Important in program optimization
- Can we
  - disregard an XPath expression? (satisfiability)
  - replace it by an easier one? (equivalence/containment)
- Satisfiability:
  - Given an XPath expression e and a DTD d
  - Question: Is there a tree T that satisfies d so that e selects at least one node in it?
- In other words, are e and d compatible?
- Known complexity bounds: ranges from polynomial-time to exponential-time. For many fragments of XPath it is NP-complete or even EXPTIME-complete.

Reasoning task: example II – XPath containment

- Containment:
  - Given a XPath expressions e, e' and a DTD d
  - Question: is it true that $d \models e \subseteq e'$?
  - I.e., whether for every tree T that satisfies d, each node selected by e is also selected by e'.
- Optimization = Equivalence: $d \models e = e'$ which is just
  - $d \models e \subseteq e'$ and
  - $d \models e' \subseteq e$.

Key ingredient: TL$^{\text{tree}}$ to query automata

Theorem Every TL$^{\text{tree}}$ formula φ can be translated, in exponential time, into an equivalent automaton $\mathcal{A}_\varphi$ of size $2^{O(\|\varphi\|)}$, i.e. an automaton that accepts T whenever φ is true in the root of T.

(Even more: get a query automaton $\mathcal{QA}_\varphi$ such that

$$\mathcal{QA}_\varphi(T) = \{s \mid (T, s) \models \varphi\}$$

for every tree T.)

Second ingredient: $\mathbf{TL}^{\mathbf{tree}}$ vs (Conditional) XPath

- Both are FO-complete so there is a translation
- The number of subformulae is what gives us the size of the automaton.
- Hence we have a simple single-exponential translation from (conditional) XPath to automata.

Application I: Reasoning about navigation and schemas

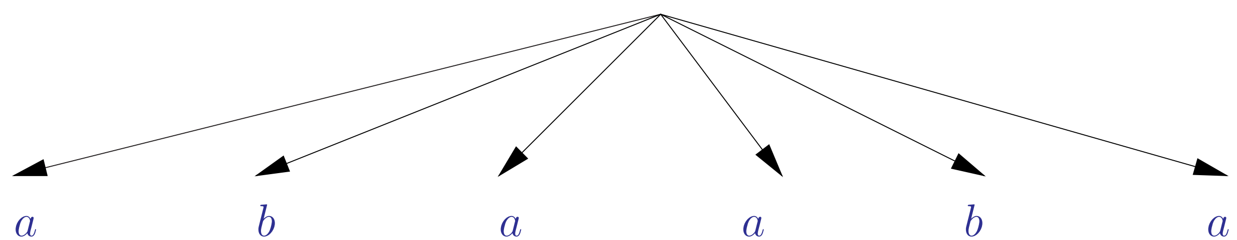
- XPath satisfiability with DTDs:
  - Translate e into a query automaton $\mathcal{QA}_e$ (time complexity: $2^{O(\|e\|)}$)
  - Take the product with the linear-size automaton $\mathcal{A}_d$ for d
  - Test $\mathcal{QA}_e \times \mathcal{A}_d$ for nonemptiness
- Time complexity:
  - Exponential in e
  - Polynomial in d

Application II: containment

- XPath containment with DTDs (i.e. $d \models e_1 \subseteq e_2$):
  - Translate e_1 and e_2 into $\mathsf{TL}^{\mathsf{tree}}$ formulae ψ_{e_1} and ψ_{e_2}
  - Construct query automaton for $\psi_{e_1} \wedge \neg\psi_{e_2}$
  - Take the product with $\mathcal{A}_d$
- The result is a query automaton that describes the set of counterexamples to containment
- Its size is $\|d\| \cdot 2^{O(\|e_1\|+\|e_2\|)}$

Unordered trees: easy punchline

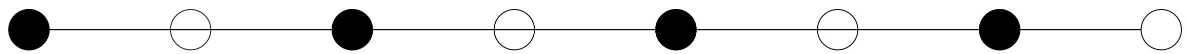
- Order buys us counting.
- Without order, counting has to be introduced explicitly.



- There is no way to say in a temporal logic that there are at least 2 children labeled a.

MSO, order, and counting

- With MSO, ordering gives us even more powerful modulo counting.
- Example: parity in MSO

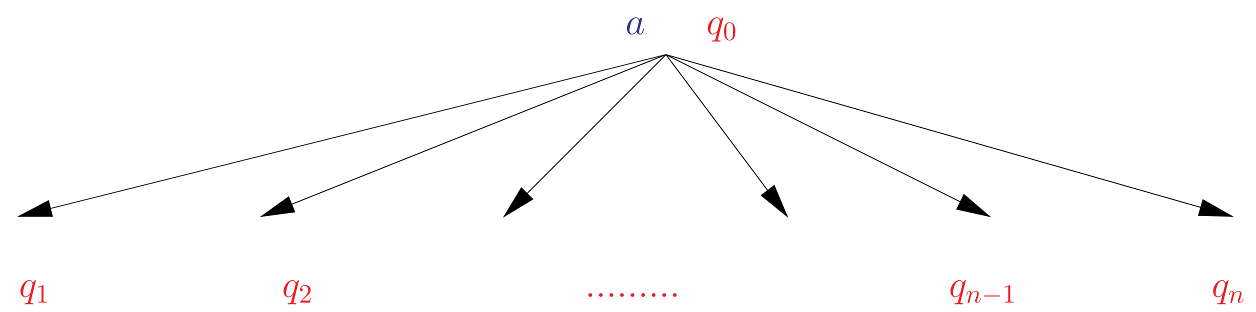


- The black set:
  - contains the first element;
  - contains every other element;
  - does not contain the last element.
- But if we only have:

we cannot say it.

Automata with counting

- New transition: δ : States $\times \Sigma \longrightarrow$ Boolean function over(V)
- $V = \{v_q^k \mid k \in \mathbb{N},\ q \in \text{States}\}$.
- A new notion of run:



- For each q, set v_q^k to true if the number of children in state q is at least k.
- If $\delta(q_0, a)$ evaluates to true, then state q_0 can be assigned.

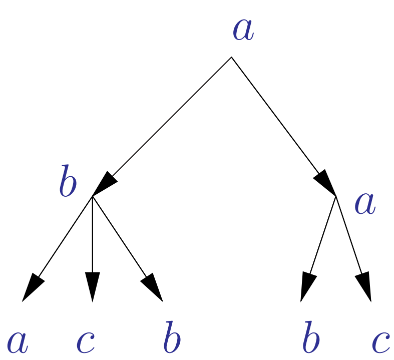
Counting in logics

Extend μ-calculus and CTL* to counting versions by changing $\mathbf{X}$ to $\mathbf{X}^k$, meaning the existence of at least k elements satisfying a formula.

Then:

- MSO $=$ counting μ-calculus
- FO $=$ counting CTL*
- For unary queries, one adds both counting and past

Other directions: streaming




```
<a>
 <b>
  <a></a>
  <c></c>
  <b></b>
 </b>
 <a>
  <b></b>
  <c></c>
 </a>
</a>
```

Streamed representation: $aba\bar{a}c\bar{c}b\bar{b}\bar{b}ab\bar{b}c\bar{c}\bar{a}\bar{a}$

Question: what properties of trees can we check by a finite automaton over the streamed representation?

Since the language of balanced parentheses is not regular, we may assume the input is already a valid stream.

Not all DTDs can be verified in a streamed fashion

- $$\begin{array}{rcl} a & \rightarrow & ab \mid ca \mid \varepsilon \\ b & \rightarrow & \varepsilon \\ c & \rightarrow & \varepsilon \end{array}$$

- For every MSO sentence φ one can find two strings of the form

  $$ab\bar{b}ab\bar{b}\ldots ab\bar{b}a\ldots a\bar{a}c\bar{c}\ldots\bar{a}c\bar{c}\bar{a}\ldots\bar{a}$$

  that agree on φ; one of them corresponds to the above DTD, and the other one to:

  $$\begin{array}{rcl} a & \rightarrow & a \mid ab \mid ca \mid \varepsilon \\ b & \rightarrow & \varepsilon \\ c & \rightarrow & \varepsilon \end{array}$$

- The problem is still open in general: when can we verify DTDs fast?

Other directions: data values

- So far we considered only labels on trees (e.g., book, author) but no data values (e.g., "WH Press").
- Adding data values quickly leads to undecidability or at least intractability.

When it becomes problematic: static analysis

- DTDs + key/foreign key constraints
- Occur all the time: relational data $\Longrightarrow$ XML
- Satisfiability problem: is a specification consistent?
- Why it might be inconsistent?
- Because one puts data in a "wrong" place in the hierarchy.

DTDs + key/foreign key constraints

$$\text{DTD:}\quad \begin{array}{lcl} \text{teachers} & \longrightarrow & (\text{teacher}^{+}) \\ \text{teacher} & \longrightarrow & (\text{teach, research}) \\ \text{teach} & \longrightarrow & (\text{subject, subject}) \end{array}$$

teacher has an attribute `name`
subject has an attribute `taught_by`.

$$\text{Constraints:}\quad \begin{array}{lcl} \text{teacher.name} & \longrightarrow & \text{teacher,} \\ \text{subject.taught\_by} & \longrightarrow & \text{subject,} \\ \text{subject.taught\_by} & \subseteq & \text{teacher.name.} \end{array}$$

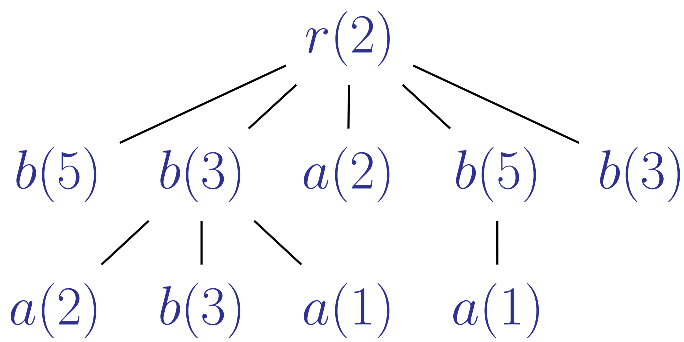
`name` is a key of `teacher`
`taught_by` is a key of `subject`
plus a foreign key referencing `name` of `teacher`

This is inconsistent!

Consistency: known facts

- Surprisingly hard problem:
- It is NP-complete for unary constraints (like those in the example, based on single-attributes)
- It is undecidable even for binary constraints (e.g., title and author determine publisher).

Model: data trees



Each node carries:

- a label, e.g., r, a, b — from a finite alphabet
- a piece of a date, e.g., a natural number — from an infinite alphabet
- easy to model multiple ones as well
- Tree automata lose their nice properties on infinite alphabets.

Data trees and logic

Addition: predicate $x \sim y$ saying that x and y carry the same data value

Example: data values of as form a key:

$$\forall x \forall y \ (Lab_a(x) \wedge Lab_a(y) \wedge x \sim y) \ \rightarrow \ x = y$$

Example: foreign key from a to b

$$\forall x \ Lab_a(x) \ \rightarrow \ \exists y \ (Lab_b(x) \wedge x \sim y)$$

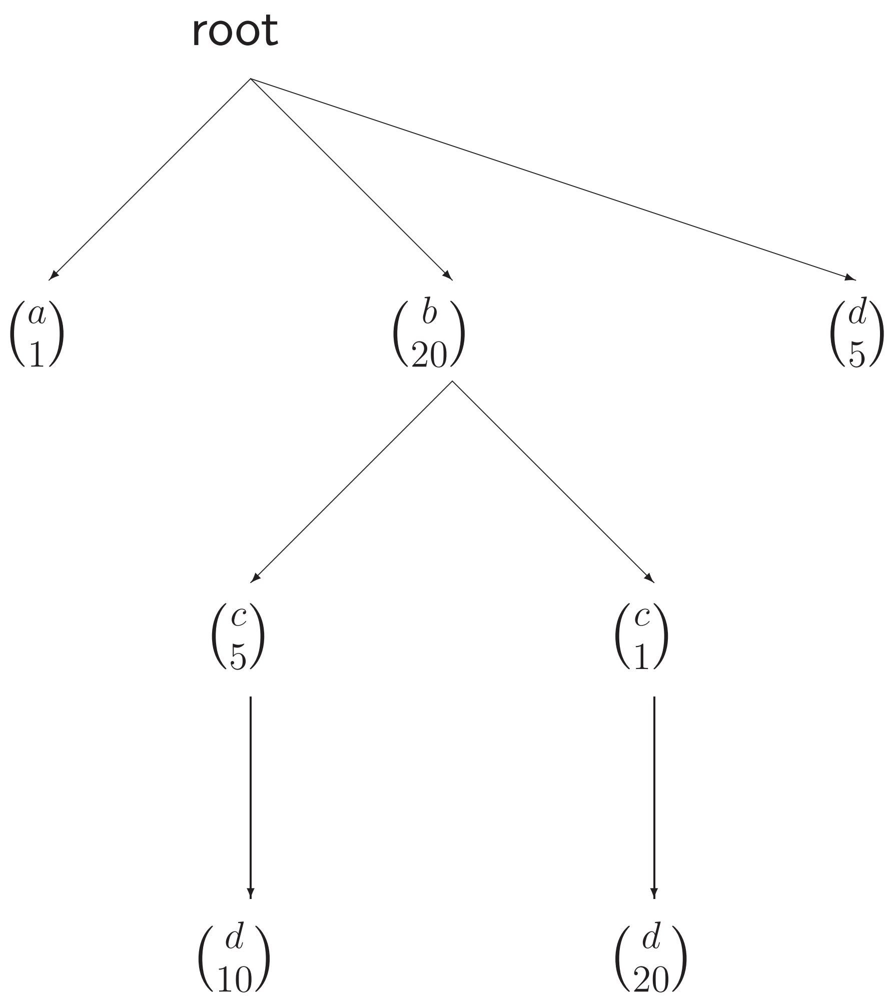
What is common here? We only use 2 variables, x and y.

Dichotomy: With two variables, first-order logic is decidable on data trees; with three or more, it is not.

Problem: complexity is astronomical (tower of 4 exponents).

A better explanation. Parameters for a data tree T

- $V_t(a)$ is the set of data values found in a-labeled nodes in T.
- $\#_t(a)$ is the number of a-nodes in t.



$$\begin{aligned} V_t(a) &= \{1\} \\ V_t(b) &= \{20\} \\ V_t(c) &= \{1, 5\} \\ V_t(d) &= \{5, 10, 20\} \end{aligned}$$

$$\begin{aligned} \#_t(a) &= 1 \\ \#_t(b) &= 1 \\ \#_t(c) &= 2 \\ \#_t(d) &= 3 \end{aligned}$$

$$\begin{array}{lcll} V(a) \cap V(b) & = & \emptyset & \surd \\ V(a) \cap V(c) & \subseteq & V(d) & \text{X} \\ V(a) \cup V(c) & \subseteq & \overline{V(b)} & \surd \end{array}$$

Unary keys/foreign keys

Logical formula	Semantics		
Key: $\forall x\, \forall y\; Lab_a(x) \wedge Lab_a(y) \wedge x \sim y \rightarrow x = y$	$\#_t(a) = \|V_t(a)\|$	$\Rightarrow$	Linear constraints
Foreign key: $\forall x\, \exists y \quad Lab_a(x) \rightarrow Lab_b(y) \wedge x \sim y$	$V_t(a) \subseteq V_t(b)$	$\Rightarrow$	Set constraints

The structure of the tree must satisfy the given schema:

DTD
Extended DTD tree automata
XML Schema

Consistency of constraints

Input:

- A tree automaton $\mathcal{A}$
- A collection $\mathcal{C}$ of set and linear constraints

Question: Is there a data tree T accepted by $\mathcal{A}$ that satisfies all constraints in $\mathcal{C}$?

This problem can be solved in NP (i.e., single exponential time)

- Automata can be encoded with linear constraints
- And so can be set constraints
- So just solve an instance of integer linear programming

Notes on papers

1. Frank Neven: Automata, Logic, and XML. CSL 2002: 2-26

   A survey of automata theoretic techniques in XML, particularly XML standards

2. Leonid Libkin: Logics for Unranked Trees: An Overview. Logical Methods in Computer Science 2(3) (2006)

   A survey of logical techniques for languages used in XML (schema, navigation, querying)

3. Georg Gottlob, Christoph Koch, Reinhard Pichler: Efficient algorithms for processing XPath queries. ACM Trans. Database Syst. 30(2): 444-491 (2005)

   How to evaluate XPath most efficiently

4. Georg Gottlob, Christoph Koch, Reinhard Pichler, Luc Segoufin: The complexity of XPath query evaluation and XML typing. Journal of the ACM 52(2): 284-335 (2005)

   Pinpointing exact complexity of many problems related to XPath evaluation and XML schemas

5. Frank Neven, Thomas Schwentick: Query automata over finite trees. Theor. Comput. Sci. 275(1-2): 633-674 (2002)

   Extending automata to formalisms that select nodes in trees

6. Georg Gottlob, Christoph Koch: Monadic datalog and the expressive power of languages for Web information extraction. Journal of the ACM 51(1): 74-113 (2004)

   Capturing MSO with a very efficient language, and applications in Web data extraction

7. Pablo Barcelo, Leonid Libkin: Temporal logics over unranked trees. LICS 2005: 31-40

   How XML languages are related to logics used in software/hardware verification

8. Leonid Libkin, Cristina Sirangelo: Reasoning about XML with temporal logics and automata. J. Applied Logic 8(2): 210-232 (2010)

   ... and how to exploit the connection to verify properties of XML

9. Thomas Schwentick: XPath query containment. SIGMOD Record 33(1): 101-109 (2004)

   A survey of techniques for testing containment and equivalence of XPath queries

10. Wenfei Fan, Leonid Libkin: On XML integrity constraints in the presence of DTDs. Journal of the ACM 49(3): 368-406 (2002)

    Explaining why unary keys and foreign keys are in NP for XML, and why beyond unary they are undecidable

11. Marcelo Arenas, Wenfei Fan, Leonid Libkin: On the Complexity of Verifying Consistency of XML Specifications. SIAM J. Comput. 38(3): 841-880 (2008)

    Pushing this further to more expressive constraints used in XML, such as those in XML Schema

12. Wim Martens, Frank Neven, Thomas Schwentick: Simple off the shelf abstractions for XML schema. SIGMOD Record 36(3): 15-22 (2007)

    Theoretical reconstructions of XML Schema

13. Wim Martens, Frank Neven, Thomas Schwentick, Geert Jan Bex: Expressiveness and complexity of XML Schema. ACM Trans. Database Syst. 31(3): 770-813 (2006)

    An automaton model for XML schema, and its use in efficient typing of documents

14. Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin: Two-variable logic on data trees and XML reasoning. Journal of the ACM 56(3) (2009)

    Decidability/undecidability boundary for 2 vs 3 variables over data trees

15. Tony Tan: Extending two-variable logic on data trees with order on data values and its automata. ACM Trans. Comput. Log. 15(1): 8 (2014)

    Pushing this to more expressive formalisms, with better (more readable) algorithms

16. Claire David, Leonid Libkin, Tony Tan: Efficient reasoning about data trees via integer linear programming. ACM Trans. Database Syst. 37(3): 19 (2012)

    The NP bound for sets and linear constraints

17. Henrik Bjrklund, Wim Martens, Thomas Schwentick: Conjunctive query containment over trees. J. Comput. Syst. Sci. 77(3): 450-472 (2011)

    Extending CQ containment from relations to databases

18. Wojciech Czerwinski, Wim Martens, Pawel Parys, Marcin Przybylko: The (Almost) Complete Guide to Tree Pattern Containment. PODS 2015: 117-130

    CQs over trees are essentially patterns: a detailed study of their containment

19. Maarten Marx: Conditional XPath. ACM Trans. Database Syst. 30(4): 929-959 (2005)

    An XPath extension that captures all first-order queries over XML documents

20. Balder ten Cate, Maarten Marx: Navigational XPath: calculus and algebra. SIGMOD Record 36(2): 19-26 (2007)

    Providing algebraic counterpart for XPath fragments

21. Loredana Afanasiev, Maarten Marx: An analysis of XQuery benchmarks. Inf. Syst. 33(2): 155-181 (2008)

The title says it all

22. Luc Segoufin, Cristina Sirangelo: Constant-Memory Validation of Streaming XML Documents Against DTDs. ICDT 2007: 299-313

Analyzing which DTDs can be checked over streamed documents