

TEMPORAL LOGICS AND XML QUERYING

Leonid Libkin

University of Edinburgh

Databases and verification

The **main goal** of both is the same:

Evaluate a **logical formula** on a **finite structure**

Databases and verification

The **main goal** of both is the same:

Evaluate a **logical formula** on a **finite structure**

In **Databases**:

- **Logical formulae**:
 - first-order (FO) = relational calculus
 - FO + counting \approx SQL
 - FO + fixed-point = datalog, etc
- **Finite structures**:
 - finite relational database
 - finite tree = XML document

Databases and verification

The **main goal** of both is the same:

Evaluate a **logical formula** on a **finite structure**

In **Verification**:

- **Logical formulae**:
 - linear-time temporal logics (LTL, etc)
 - branching time temporal logics (CTL, CTL*, etc)
 - fixed-point logics (μ -calculus)
- **Finite structures**:
 - Kripke structures
 - labeled transition systems

Main goal revised

Evaluate a logical formula on a finite structure

Main goal revised

Evaluate a logical formula on a finite structure



Efficiently evaluate a logical formula on a finite structure

Main goal revised

Evaluate a logical formula on a finite structure



Efficiently evaluate a logical formula on a finite structure

- In databases: query evaluation
- In verification: model-checking
- Both concentrate on efficient evaluation

Connections

Logics used in both fields are often closely connected:

- **Kamp's Theorem** Over finite and infinite words, $FO=LTL$
- **Hafer-Thomas's Theorem** Over finite binary trees, $FO = CTL^*$
- Over finite strings or finite binary trees, $Datalog = \mu\text{-calculus}$ (folklore)
- Temporal logics naturally define:
 - unary queries (i.e. one free variable in logics such as FO)
 - Boolean queries (sentences)
 - Often this is what is most important in the XML context (information extraction – Gottlob et al)

Different syntax means lower complexity: LTL

Syntax:

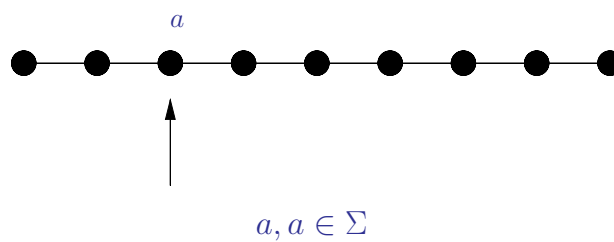
$$\varphi := a(\in \Sigma) \mid \varphi \vee \varphi' \mid \neg \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi' \mid \mathbf{Y}\varphi \mid \varphi \mathbf{S}\varphi'$$

Different syntax means lower complexity: LTL

Syntax:

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

Semantics:

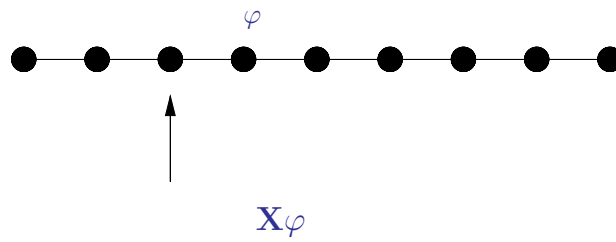


Different syntax means lower complexity: LTL

Syntax:

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

Semantics:

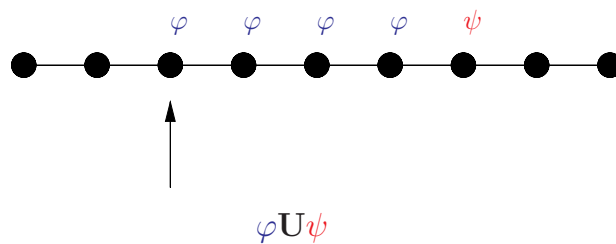


Different syntax means lower complexity: LTL

Syntax:

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

Semantics:

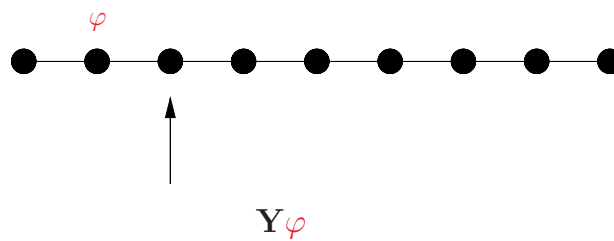


Different syntax means lower complexity: LTL

Syntax:

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

Semantics:

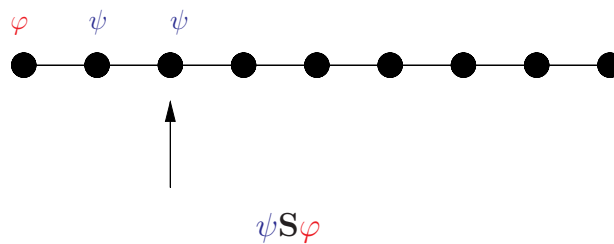


Different syntax means lower complexity: LTL

Syntax:

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

Semantics:



LTL cont'd

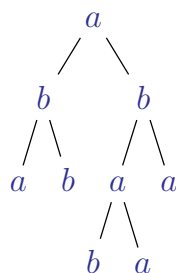
- LTL = FO over strings (more precisely, FO formulae $\varphi(x)$)
- LTL over strings can be evaluated in time $O(\|\varphi\| \cdot |s|)$.
- To evaluate FO with linear **data** complexity, one needs **non-elementary query** complexity (modulo some complexity-theoretic assumptions; Frick/Grohe 2002)
- Even for CQs fixed-parameter tractability or linearity is tricky to achieve (Yannakakis 81 $\rightarrow \dots \rightarrow$ Grohe/Schwentick/Segoufin; Gottlob/Leone/Scarcello)

Trees

Linear-time logics aren't often occurring in databases, but branching time logics do, especially for databases that represent trees.

Classical work: **ranked** trees; e.g., binary, ternary, etc trees.

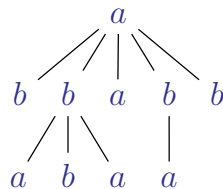
A **binary** tree:



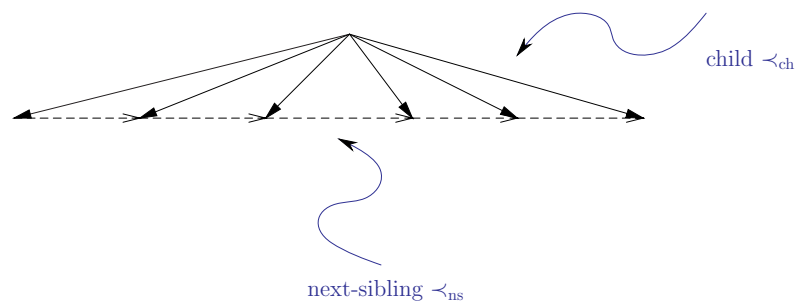
Unranked trees

But now, thanks to XML, we work with **unranked** trees.

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



Unranked trees are Kripke structures



Transitive closures:

- \prec_{ch}^* of \prec_{ch} (descendant)
- \prec_{ns}^* of \prec_{ns} (younger child)

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

Logic/automata connection

- Extremely important in verification
- **Regular** (tree) languages = given by (tree) **automata**.
- Typically characterized via **MSO** — **M**onadic **S**econd-**O**rder logic
 - MSO is an extension of first-order logic with quantification over sets.
- Classical results:
 - **Büchi**: For strings, Regular = MSO-definable.
 - **Thatcher-Wright**: For finite binary trees, Regular = MSO-definable.
 - **Thatcher**→**forgotten**→**folklore**: the same is true for unranked trees

MSO on unranked trees: tying together verification & databases

- One can check $T \models \varphi$ in time $f(\|\varphi\|) \cdot \|T\|$.
- f is necessarily nonelementary (Frick/Grohe): if not, then P=NP.
- But:
 - **Theorem** Over unranked trees:
$$\text{MSO} = (\text{monadic}) \text{ Datalog} = (\text{alternation-free}) \mu\text{-calculus}$$

(Gottlob/Koch '02, Barceló/L., '05)
- Monadic datalog and alternation-free μ -calculus can be evaluated in time $\|\varphi\| \cdot \|T\|$.
- μ -calculus on trees can be evaluated in time $\|\varphi\|^2 \cdot \|T\|$ (Mateescu, '02).

Back to FO: XPath

- 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.

XPath isn't really new!

- 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 $\beta := \text{LTL over state formulae}$
- Temporal operators must specify a relation of the Kripke structure (axes in the language of XPath) they apply to.

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

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

CTL* and FO over trees

Recall Hafer-Thomas '87: $CTL^* = FO$ over binary trees.

Theorem Over unranked trees,

- $CTL^* = FO$
(Barcelo, L., 2005);
- Conditional XPath (XPath + Until) = FO
(Marx 2004)

Linear-time on trees

- CTL^* isn't the best temporal logic from the complexity-of-evaluation point of view
 - more expressive than LTL
 - translations into μ -calculus exhibit exponential blowup
- But, despite trees being branching and not linear, a logic similar to LTL can be defined for them

Linear-time tree logic TL^{tree}

Syntax:

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

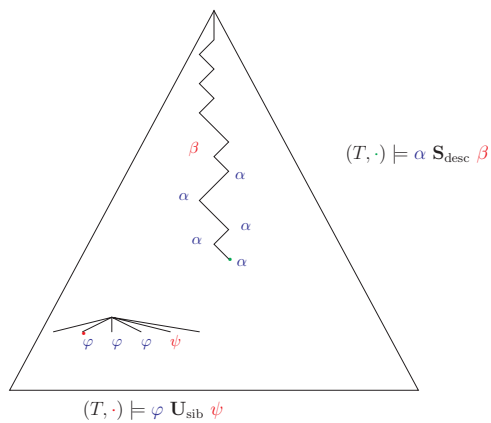
where $*$ is either `desc` or `sib`.

Linear-time tree logic TL^{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 TL^{tree}

Theorem

$$TL^{\text{tree}} = FO$$

- over unordered tree, if sibling-edge temporal connectives are not used (Schlingloff '92)
- over ordered trees (Marx '04)
- Complexity of query evaluation: $O(\|\varphi\| \cdot \|T\|)$

n -ary queries

- What if we need to return n -tuples of nodes?
- Two solutions
 - **solution 1**: start with a language for **pairs** and add one binary term.
 - **solution 2**: start with a language for **unary queries** and add several binary terms.

***n*-ary queries: solution 1**

- Binary operation: the largest common prefix of two strings $s \sqcap s'$
- Terms in n variables:

$$t, t' := x_i, i \leq n \mid \varepsilon \mid t \sqcap t'$$

Theorem (implicit in Schwentick '00)

Let \mathcal{L} be a language that captures binary queries $q(x, y)$ in FO or MSO.

Then Boolean combinations of

$$q(t, t')$$

capture n -ary queries over FO or MSO, respectively.

***n*-ary queries: solution 1**

- Binary operation: the largest common prefix of two strings $s \sqcap s'$
- Terms in n variables:

$$t, t' := x_i, i \leq n \mid \varepsilon \mid t \sqcap t'$$

Theorem (implicit in Schwentick '00)

Let \mathcal{L} be a language that captures binary queries $q(x, y)$ in FO or MSO.

Then Boolean combinations of

$$q(t, t')$$

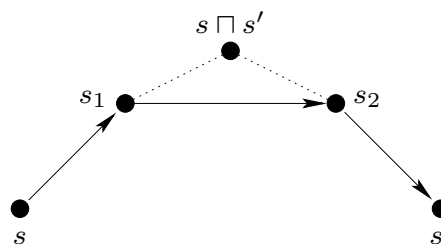
capture n -ary queries over FO or MSO, respectively.

- Languages for binary queries exist (Marx and colleagues) but they aren't the most natural ones.

n -ary queries: solution 2

- Idea (for FO): we start with
 - a logic \mathcal{L}_1 that captures **Boolean FO** over words (e.g., LTL)
 - a logic \mathcal{L}_2 that captures **unary FO** over trees
- and **combine** them using:
 - an extended set of terms, and
 - node tests

n -ary queries: solution 2 – terms



In addition to $s \sqcap s'$ we have:

- s_1 – successor of $s \sqcap s'$ in the direction of s
- s_2 – successor of $s \sqcap s'$ in the direction of s'
- Extended set of terms:
 - longest common prefix \sqcap
 - successor of s in the direction of s'

n -ary queries: solution 2 – combination mechanism

To construct a **basic** formula:

- compute two terms, t and t'
- view an “interval” between them as a word labeled by \mathcal{L}_2 formulae, i.e.:
 - in every position of that word evaluate some formulae of \mathcal{L}_2
- evaluate an \mathcal{L}_1 formula over the word.

Theorem (Arenas, Barceló, L., '07)

Boolean combinations of **basic** formulae capture n -ary FO.

One can replace FO by MSO everywhere.

n -ary queries: evaluation

Assume that:

- For \mathcal{L}_1 formulae α and words w , checking $w \models \alpha$ is done in $f_1(\|\alpha\|) \cdot \|w\|^k$.
- For \mathcal{L}_2 formulae β and trees T , computing $\{s \mid T \models \beta(s)\}$ takes $f_2(\|\alpha\|) \cdot \|T\|^\ell$.

Then:

- For a combined formula φ , a tree T and an n -tuple of nodes \bar{s} , checking whether $T \models \varphi(\bar{s})$ can be done in
$$O(\max\{f_1(\|\varphi\|), f_2(\|\varphi\|)\} \cdot \|T\|^{\max\{k,\ell\}})$$
- In particular, we can find logics \mathcal{L}_1 and \mathcal{L}_2 so that query evaluation is **linear**: $O(\|\varphi\| \cdot \|T\|)$.