Databases Meet Verification

What do we have in common, and how can we help each other?

Leonid Libkin

University of Edinburgh

PART I. A few random thoughts on connections between the two fields.
PART II. How verification techniques can help database people – a random (and biased) example.
PART III. How database techniques can help verification people (a similar type of example).

PART I: DATABASES AND VERIFICATION

The main goal of both is the same:

$\boxed{\text{Evaluate a logical formula on a finite structure}}$

PART I: DATABASES AND VERIFICATION

The main goal of both is the same:

$\boxed{\text{Evaluate a logical formula on a finite structure}}$

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

PART I: DATABASES AND VERIFICATION

The main goal of both is the same:

$\boxed{\text{Evaluate a logical formula on a finite structure}}$

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

$\boxed{\text{Evaluate a logical formula on a finite structure}}$

Main goal revised

$\boxed{\text{Evaluate a logical formula on a finite structure}}$

$$\Downarrow$$

$\boxed{\text{Efficiently evaluate a logical formula on a finite structure}}$

Main goal revised

$\boxed{\text{Evaluate a logical formula on a finite structure}}$

$\Downarrow$

$\boxed{\text{Efficiently evaluate a logical formula on a finite structure}}$

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

A side remark

Sometimes one looks at infinite structures with finite representations.

In databases

- Temporal, geographical data
- Represented by first-order constraints
- Query evaluation = mix of constraint solving and decision procedures for logical theories

In verification

- Parameterized systems; various infinite graphs with decidable MSO theories
- Often represented by automata/transducers
- Model-checking: typically by complex automata constructions

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 $= \mu$-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)

What's wrong with first-order logic?

- Complexity!
- It could be very high.
- But it could be lowered by changing the syntax to something that
  - is very natural to use for writing specifications (Pnueli's Turing award!) and
  - has good algorithmic properties

Different syntax means lower complexity: LTL

Syntax:

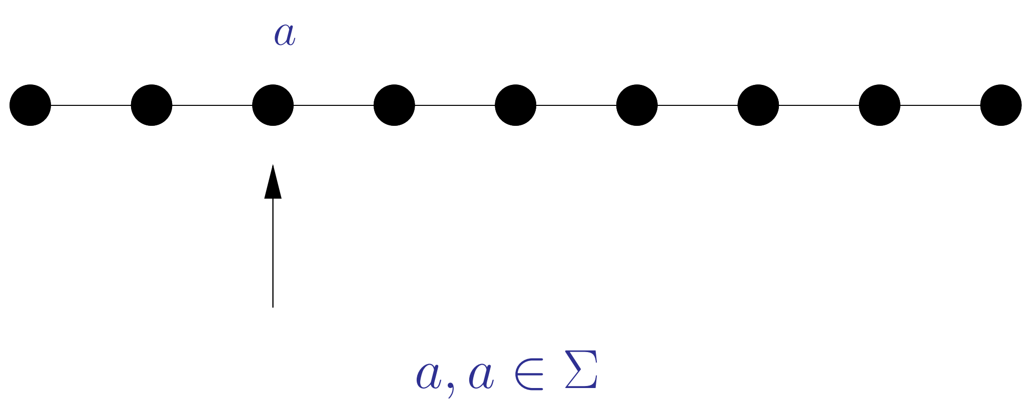
$$\varphi \ := \ a(\in \Sigma) \ | \ \varphi \vee \varphi' \ | \ \neg\varphi \ | \ \mathbf{X}\varphi \ | \ \varphi\mathbf{U}\varphi' \ | \ \mathbf{Y}\varphi \ | \ \varphi\mathbf{S}\varphi'$$

Different syntax means lower complexity: LTL

Syntax:

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

Semantics:

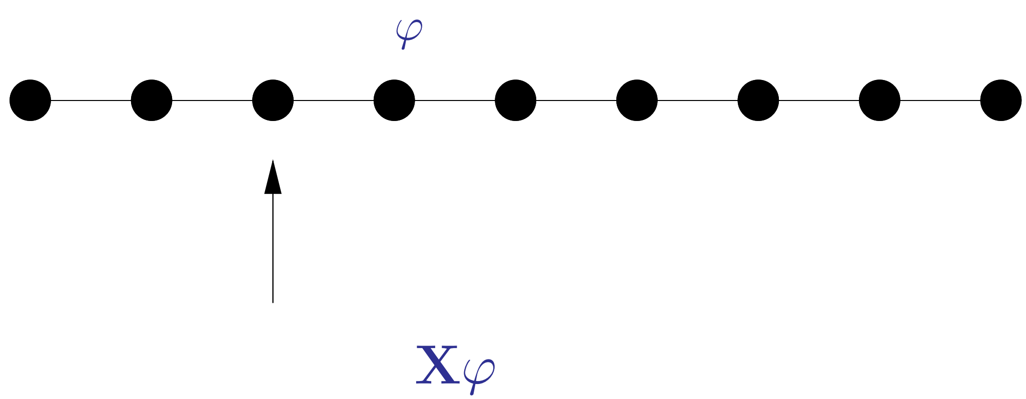


Different syntax means lower complexity: LTL

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:

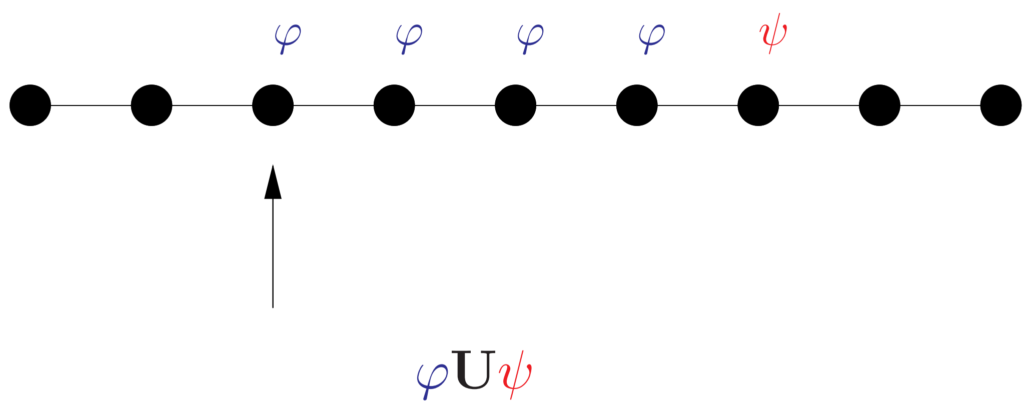


Different syntax means lower complexity: LTL

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:

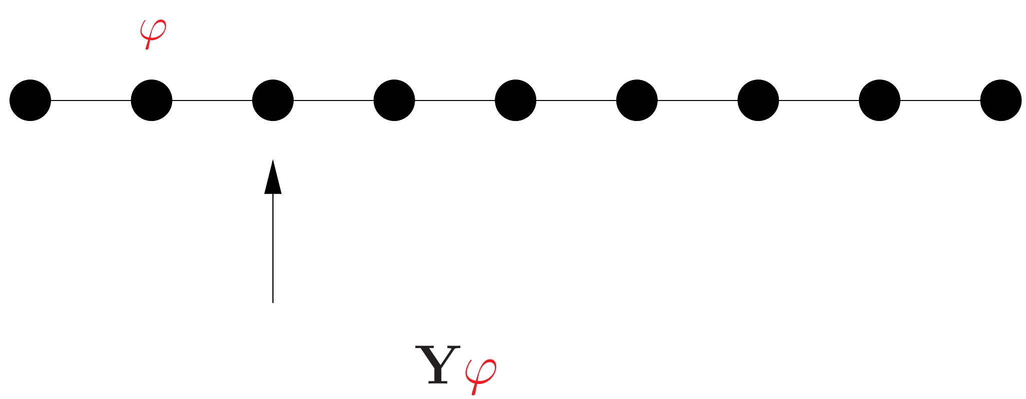


Different syntax means lower complexity: LTL

Syntax:

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

Semantics:

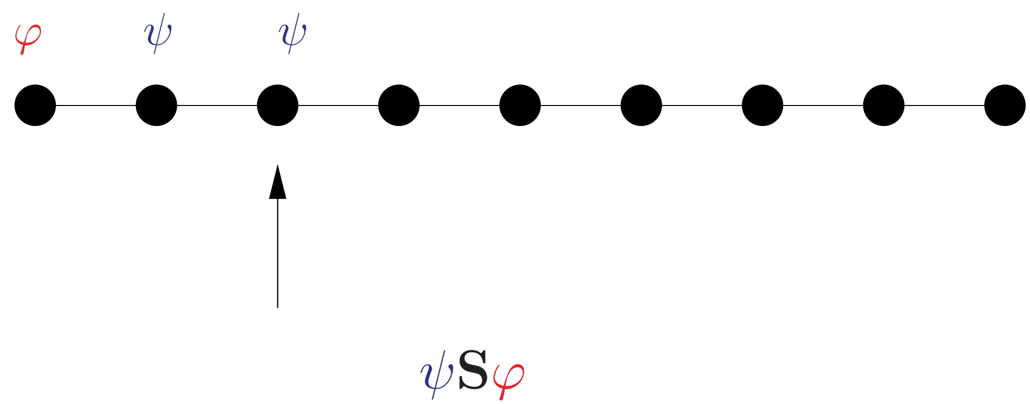


Different syntax means lower complexity: LTL

Syntax:

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

Semantics:



LTL cont'd

- LTL $=$ FO over strings (more precisely, FO formulae $\varphi(x)$)
- LTL over strings can be evaluated in time $O(\|\varphi\| \cdot |s|)$.
- Complexity:
  - in terms of data (s) – linear
  - in terms of query (φ) – linear
- What if we want to evaluate FO with linear data complexity?
- One needs non-elementary query complexity!
  (modulo some complexity-theoretic assumptions; Frick/Grohe 2002)

LTL cont'd

- Recall: in recursion theory, elementary $=$ a certain class of computable functions satisfying

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

- Dates back to the "optimistic" 1950s when those functions were viewed as relatively "simple". These "towers of 2s" grow very fast:

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

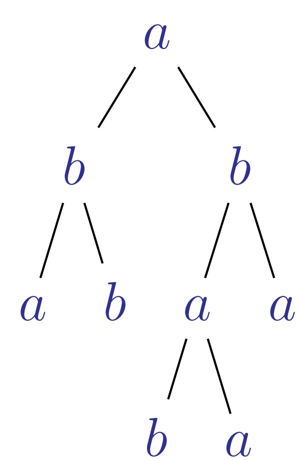
- $\textsc{Tower}(5)$ exceed the number of atoms in the visible universe.
- $\textsc{Tower}(6)$ exceed the number of atoms in the universe.
- Hence impractical...

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.

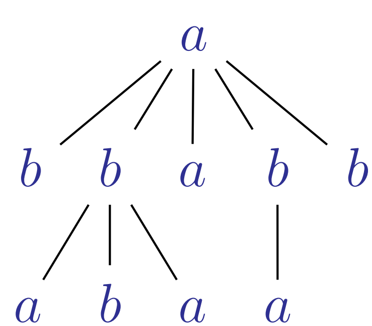
A binary tree:



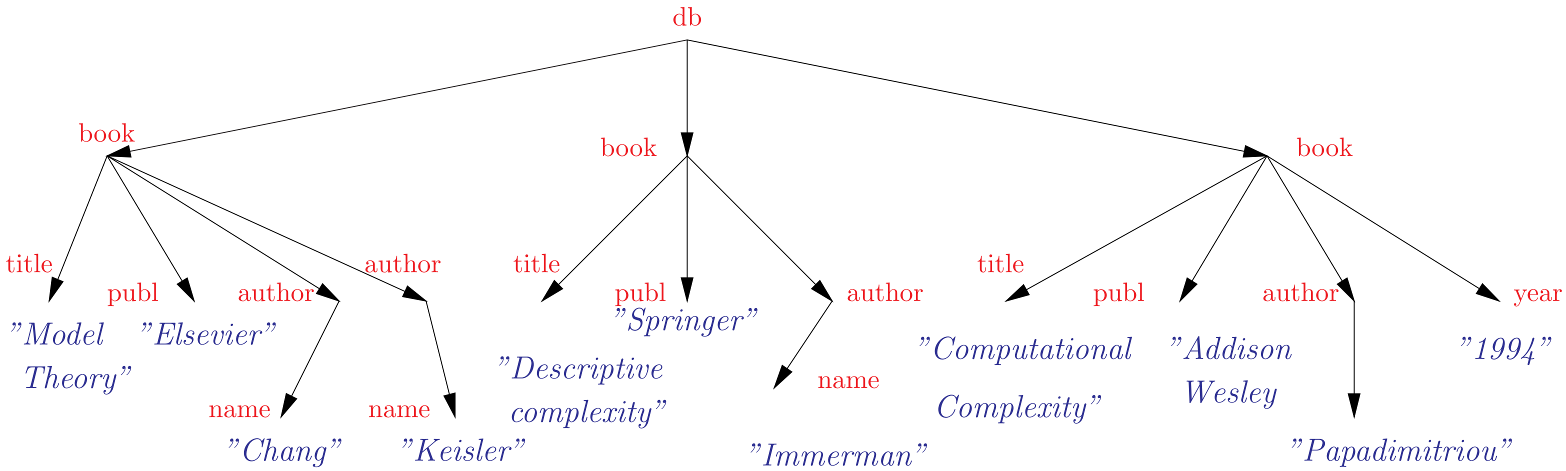
Unranked trees

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.



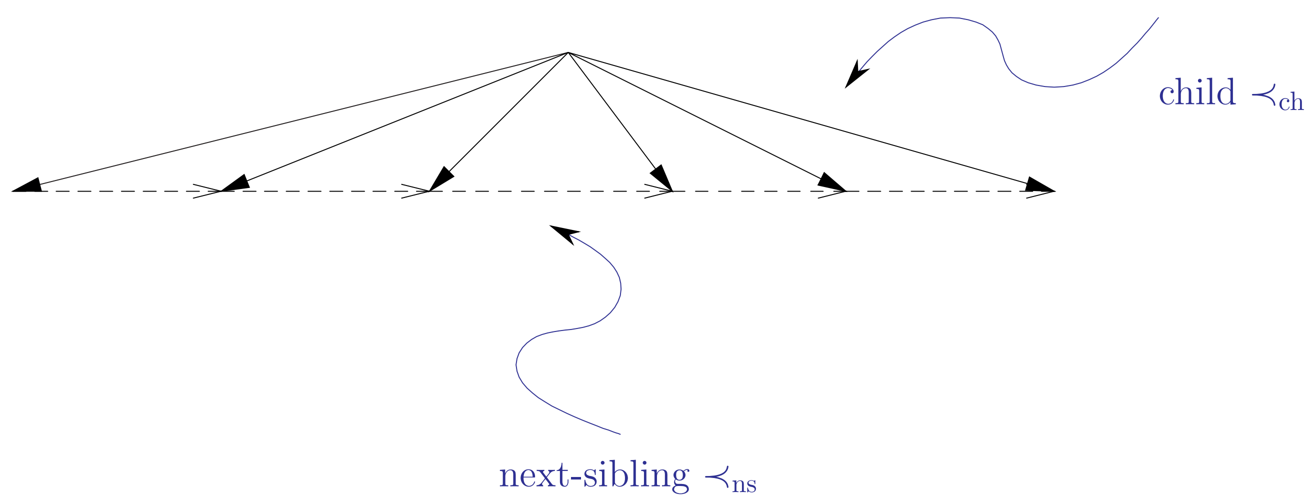
XML document as an unranked tree



Structures used in verification

- Kripke structures:
  - A set of states S
  - one or several binary relations $E_1, \ldots, E_k \subseteq S \times S$ – these define possible transitions
  - States are labeled.
- Temporal logic formulae talk about paths over the Kripke structure:
  - On every E-path, we eventually reach a node labeled a.
  - On every E-path, if we see a node labeled a, then later we will see a node labeled b.

Unranked trees are Kripke structures



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).

Logic/automata connection

- Very important in verification: 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 and unranked 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.

MSO on unranked trees: tying together verification & databases, via automata

- 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:

  $$\text{MSO} \;=\; \text{(monadic) Datalog} \;=\; \text{(alternation-free) } \mu\text{-calculus}$$

  (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.

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

XPath isn't really new!

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

$$\begin{array}{lrcl} \text{state formulae} & \alpha & := & a \mid \alpha \vee \alpha' \mid \neg\alpha \mid \mathbf{E}\beta \\ \text{path formulae} & \beta & := & \text{LTL over state formulae} \end{array}$$

- 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 $\Sigma = \{a, b\}$):

$$\neg\mathbf{E}\Big((a \vee b)\ \mathbf{U}_{\mathsf{desc}}\ b \Big)$$

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 $\mathrm{TL}^{\mathrm{tree}}$

Syntax:

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

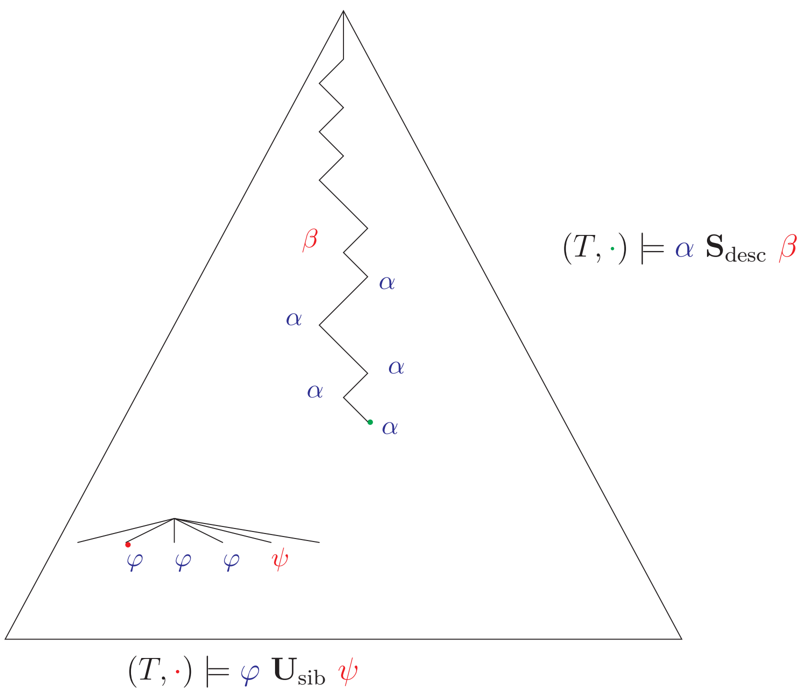
where * is either desc or sib.

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

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

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

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

PART II – Verification Helps Databases
Static XML reasoning

(based on L., Sirangelo, LPAR'08)

- 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.
  - LTL to nondeterministic or alternating Büchi automata
  - PDL, CTL, μ-calculus to (subclasses of) tree automata
- Two recent examples:
  - Calvanese/De Giacomo/Lenzerini/Vardi: regular XPath to alternating tree automata (similar in spirit to the LTL-to-alternating word automata translation);
  - L., Sirangelo: FO and equivalent formalisms + schemas into nondeterministic tree automata (similar to the classical Vardi-Wolper translation from LTL to Büchi)

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 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 (Neven/Schwentick) $\mathcal{QA}_\varphi$ that not only accepts/rejects but also selects states:

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

that is, it determines not only whether φ is true in the root, but gives a description of all nodes s where φ is true.

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

- $\mathsf{TL}^{\mathsf{tree}}$ is a convenient intermediate logic.
- FO-complete so XPath can be translated into it without increasing the overall complexity.

Proposition There is a translation of node formulae α of (core or conditional) XPath into formulae α' of $\mathsf{TL}^{\mathsf{tree}}$ such that the number of subformulae of α' is at most linear in the size of α. Moreover, if α does not use any disjunctions of path formulae, then the size of α' is at most linear in the size of α.

- 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 TL$^{\text{tree}}$ 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 $\|d\| \cdot 2^{O(\|e_1\|+\|e_2\|)}$

Application II cont'd

It is straightforward to extend this to complex containment statements:

- Setting:
  - A set $\{e_1, \ldots, e_n\}$ of XPath expressions;
  - a Boolean combination $\mathcal{C}$ of inclusions $e_i \subseteq e_j$.
  - Question: $d \models \mathcal{C}$?
- The same translation technique shows:
  - one can construct an unranked tree automaton of size $\|d\| \cdot 2^{O(\|\mathcal{C}\|)}$ whose language is empty iff $d \models \mathcal{C}$.

PART III

Databases help verification: NESTED WORDS

Alur-Arenas-Barceló-Etessami-Immerman-Libkin, LICS 2007
full version in LMCS (LICS'07 issue)

Verifying linear-time properties

- Specifications are given by LTL formulae φ.
- Programs are modeled as finite structures $\mathcal{M}$:
  - Kripke structures;
  - labeled transition systems.
- Execution of a program – an infinite path through $\mathcal{M}$.
- Model-checking problem: does every path in $\mathcal{M}$ satisfy φ, i.e.

$$\mathcal{M} \models \varphi.$$

Basic properties of LTL

- Kamp's Theorem: over ω-words, LTL $=$ First-Order Logic (FO).
  I.e, LTL is expressively complete for FO.

- Vardi-Wolper: Each LTL formula φ can be translated into an equivalent Büchi automaton $\mathcal{A}_\varphi$ of size $2^{O(|\varphi|)}$.

- Model-checking:

$$\mathcal{M} \models \varphi \quad \Leftrightarrow \quad L(\mathcal{M} \times \mathcal{A}_{\neg\varphi}) \neq \emptyset.$$

- Complexity: both model-checking and satisfiability are solvable in exponential-time, and are PSPACE-complete.

Adding nesting structure: calls and returns

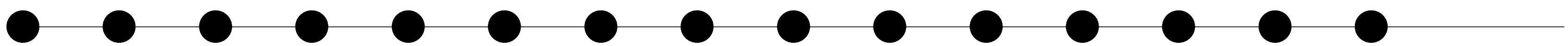
- An execution of a procedural program gives us more than just a linear sequence of program states.
- In addition, we have matching calls and returns.
- Hence we have an ω-word with a nesting structure — nested words.

Adding nesting structure: calls and returns

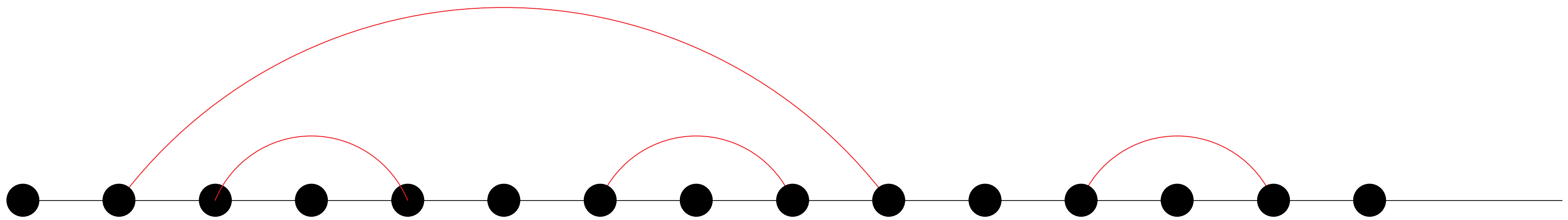
- An execution of a procedural program gives us more than just a linear sequence of program states.
- In addition, we have matching calls and returns.
- Hence we have an ω-word with a nesting structure — nested words.

- Several abstract models of procedural programs that generate nested words:
  - pushdown systems;
  - recursive state machines;
  - Boolean programs.
- In the finite case, nested words are essentially XML trees under the SAX representation.

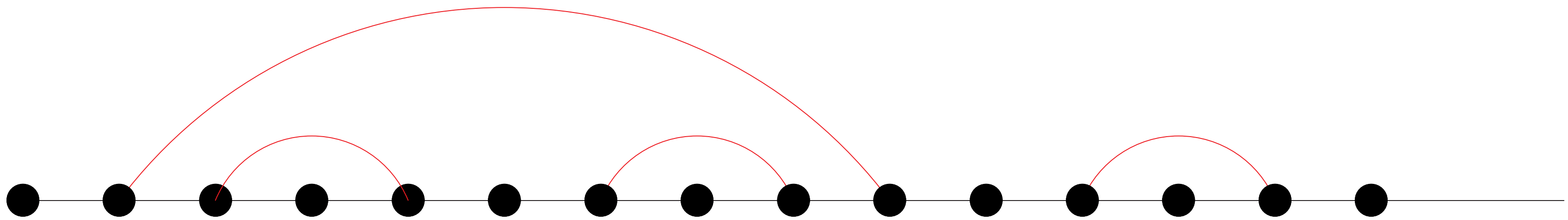
Adding nesting structure



Adding nesting structure



Adding nesting structure



Nested word: a usual (finite or ω) word plus a matching relation.

Matching: a binary 1-to-1 relation μ on so that

- if $\mu(i, j)$, then $i < j$;
- if $\mu(i, j)$ and $\mu(i', j')$ and $i < i'$ then either $j < i'$ or $j' < j$.

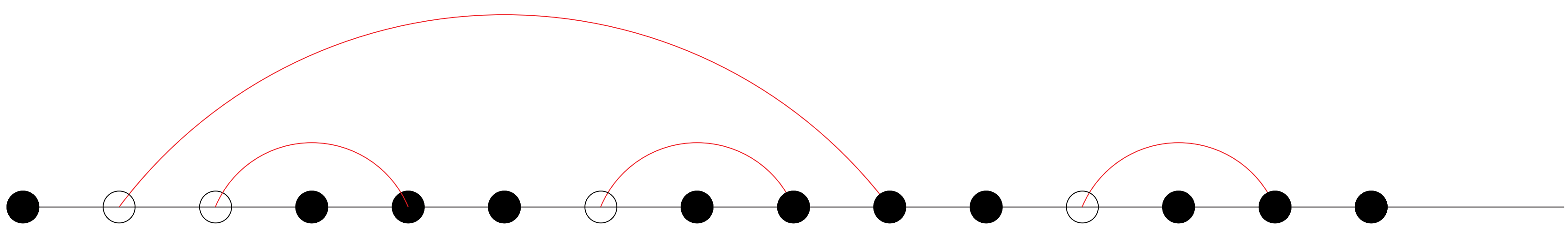
Calls and returns

If $\mu(i,j)$, then i is a call position, and j is a return position.

Calls and returns

If $\mu(i, j)$, then i is a call position, and j is a return position.

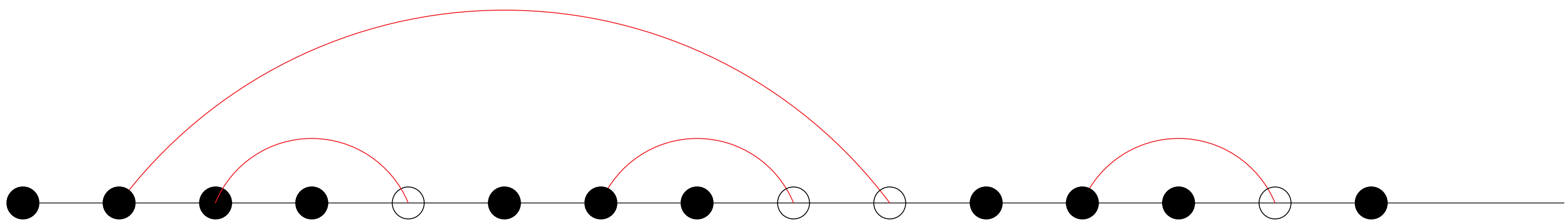
Calls:



Calls and returns

If $\mu(i, j)$, then i is a call position, and j is a return position.

Returns:



Positions that are neither calls nor returns are internal positions.

Temporal logics for nested words

The basics:

- Atomic propositions
- call, ret, int for call/return/internal positions.
- Boolean connectives $\wedge, \vee, \neg$.

Temporal logics for nested words

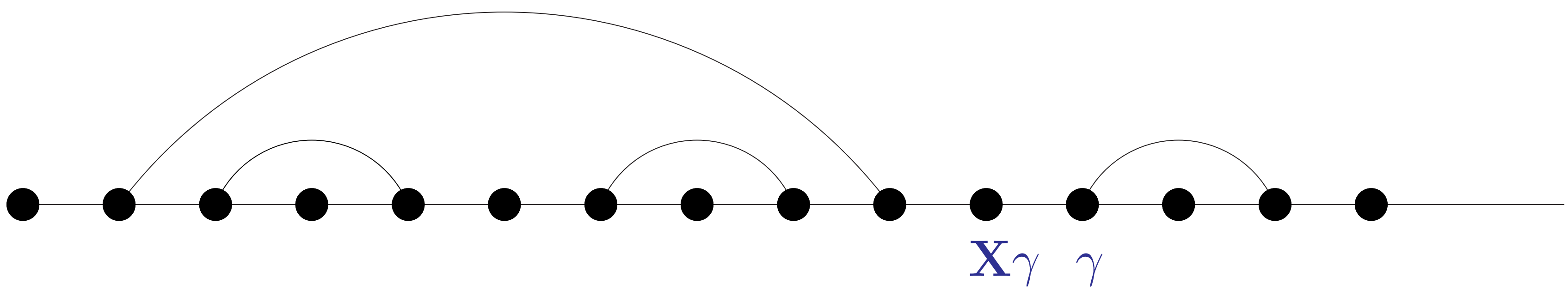
Next operators:

$$\mathbf{X}\varphi \qquad \mathbf{X}^{-}\varphi \qquad \mathbf{X}_{\mu}\varphi \qquad \mathbf{X}_{\mu}^{-}\varphi$$

Temporal logics for nested words

Next/previous operators:

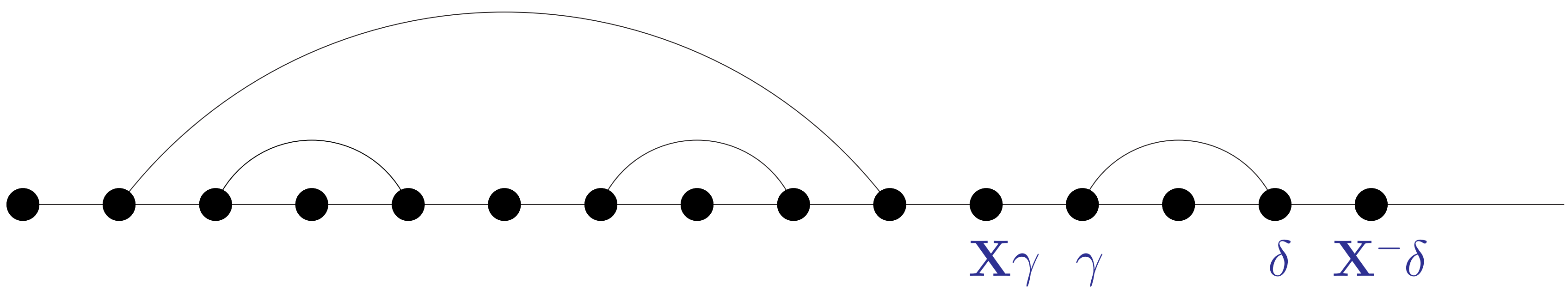
$$\mathbf{X}\varphi \quad \mathbf{X}^{-}\varphi \quad \mathbf{X}_{\mu}\varphi \quad \mathbf{X}^{-}\varphi$$



Temporal logics for nested words

Next operators:

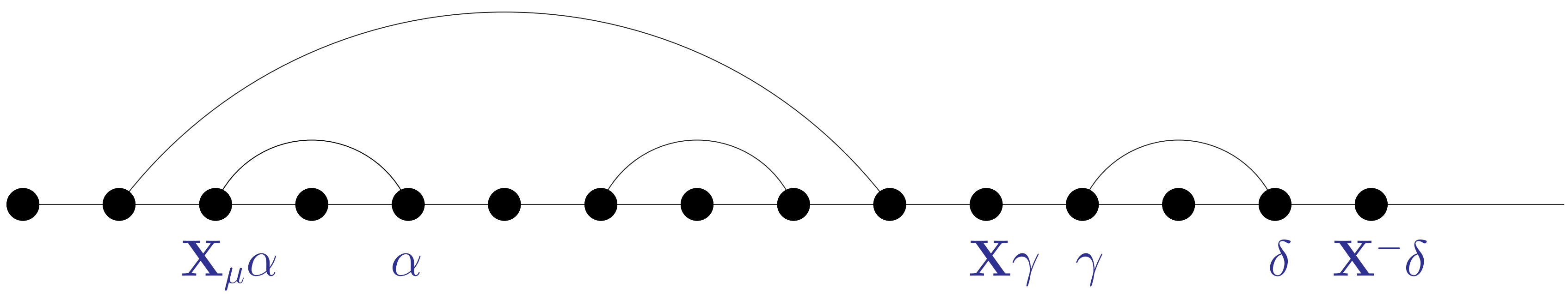
$$\mathbf{X}\varphi \qquad \mathbf{X}^{-}\varphi \qquad \mathbf{X}_{\mu}\varphi \qquad \mathbf{X}^{-}\varphi$$



Temporal logics for nested words

Next operators:

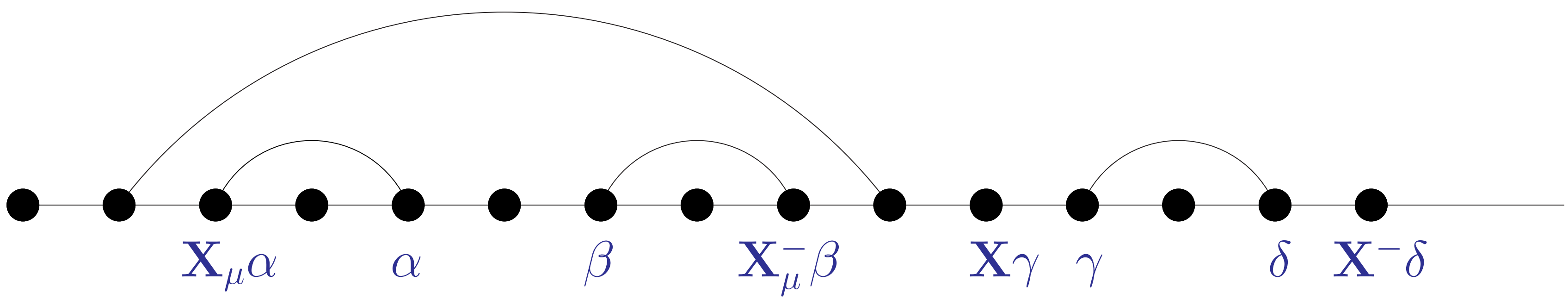
$$\mathbf{X}\varphi \quad \mathbf{X}^{-}\varphi \quad \mathbf{X}_{\mu}\varphi \quad \mathbf{X}^{-}\varphi$$



Temporal logics for nested words

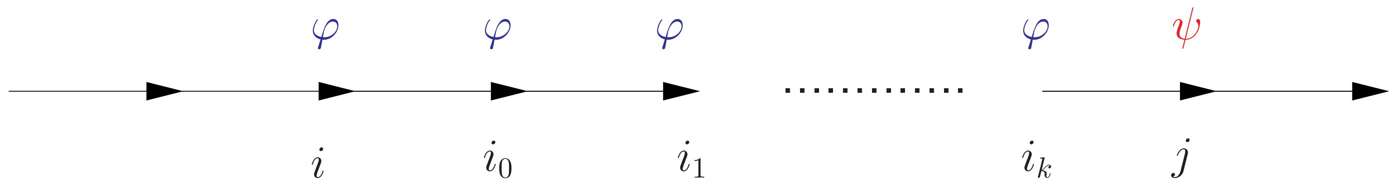
Next operators:

$$\mathbf{X}\varphi \qquad \mathbf{X}^{-}\varphi \qquad \mathbf{X}_{\mu}\varphi \qquad \mathbf{X}^{-}\varphi$$

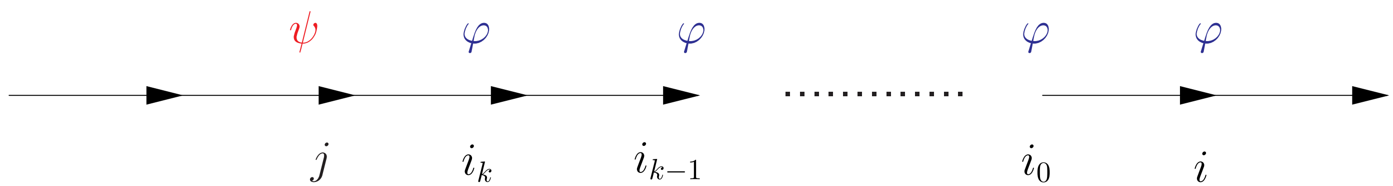


Temporal logics for nested words

Until/Since operators are standard but need a notion of a path:



$$(w, i) \models \varphi \mathbf{U} \psi$$



$$(w, i) \models \varphi \mathbf{S} \psi$$

Paths in nested words

Of course we have the same linear path

$$i,\ i+1,\ i+2,\ i+3, \ldots$$

as in the usual words and ω-words.

But in addition the nesting structure gives us many new possibilities.

Alur/Etessami/Madhusudan: several paths inspired by specifications, but the resulting logic CaReT is (probably) not FO-complete.

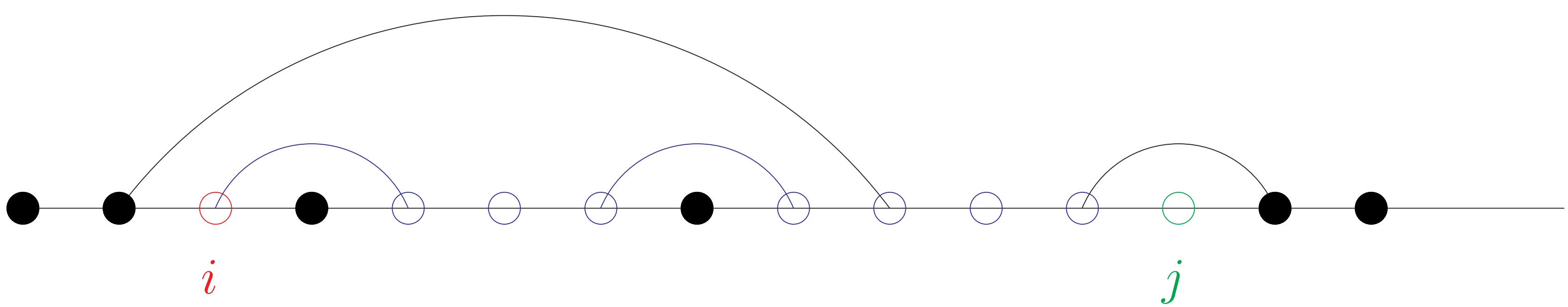
Moreover, natural FO-complete extensions have 2-exponential complexity – very bad!

A new notion of path

Summary path — the shortest path between i and j, where $j > i$.

Idea: if on the way from i to j we encounter a call position k, then:

- if j is inside the call, continue to $k+1$;
- if not, skip the call and jump to its return.



Nested Word Temporal Logic NWTL

NWTL includes:

- propositions, Boolean connectives;
- next/previous operators;
- Until and Since for summary paths.

Theorem NWTL is expressively complete for FO.

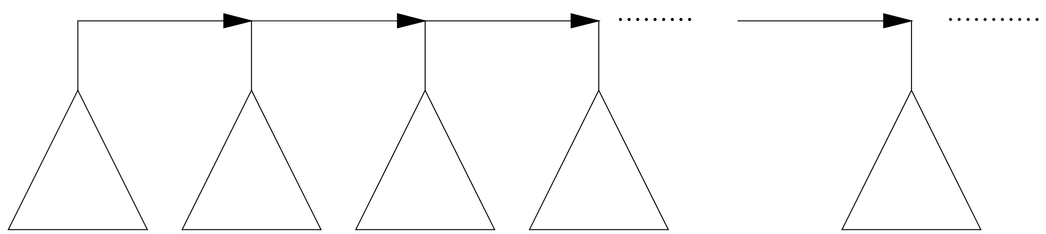
It also can be translated into automata so that:

- The translation is single-exponential;
- Emptinness checking is polynomial.

BUT: where are databases???

In the proof: Marxist composition

Translation into trees: each nested ω-word can be translated into a tree like this:



where all subtrees are finite.

Then an FO formula is, by a composition argument and Kamp's theorem, an LTL formula, where atomic propositions are types of finite subtrees.

Finite subtrees then are viewed as unranked trees, and we use Marx's expressive completeness of Conditional XPath, and translate it back into NWTL.

Final thought(s)

- Historically, there was very little interaction between databases and verification — despite the fields being close to each other, at least in terms of techniques.
- Each field can easily benefit from techniques developed in the other, sometimes in an unexpected way (verification techniques for programs with recursive procedure calls based on XML navigation languages).
- There will be more interaction as we start studying behavior and verification of webservices that depend on answers to databases queries.