Databases Meet Verification

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

Leonid Libkin

University of Edinburgh

PART I

DATABASES VS VERIFICATION

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

$\Downarrow$

Efficiently evaluate a logical formula on a finite structure

Main goal revised

Evaluate a logical formula on a finite structure

$\Downarrow$

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 = μ-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:

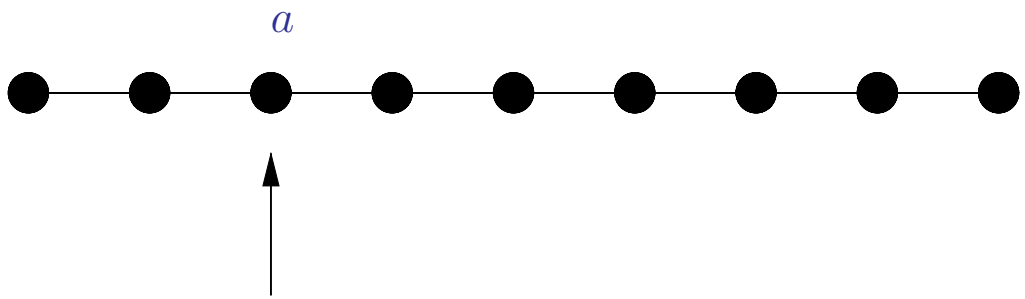
$$\varphi \;:=\; a(\in \Sigma) \;\mid\; \varphi \vee \varphi' \;\mid\; \neg\varphi \;\mid\; \mathbf{X}\varphi \;\mid\; \varphi \mathbf{U} \varphi' \;\mid\; \mathbf{Y}\varphi \;\mid\; \varphi \mathbf{S} \varphi'$$

Different syntax means lower complexity: LTL

Syntax:

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

Semantics:

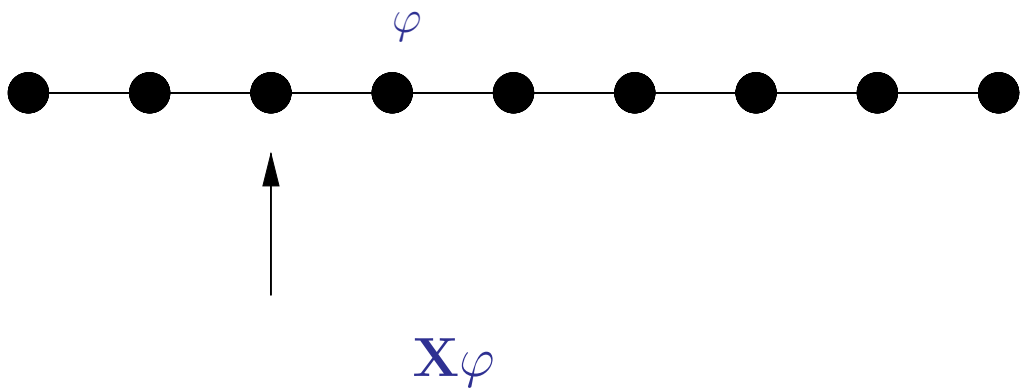


Different syntax means lower complexity: LTL

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:

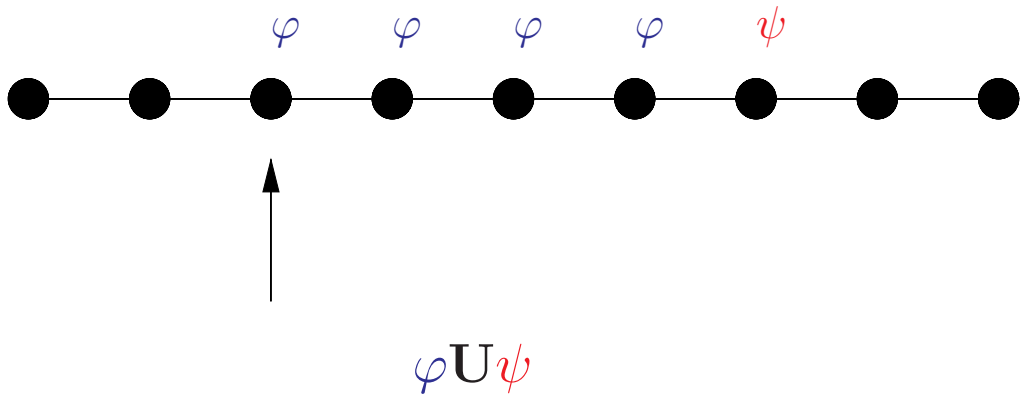


Different syntax means lower complexity: LTL

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:

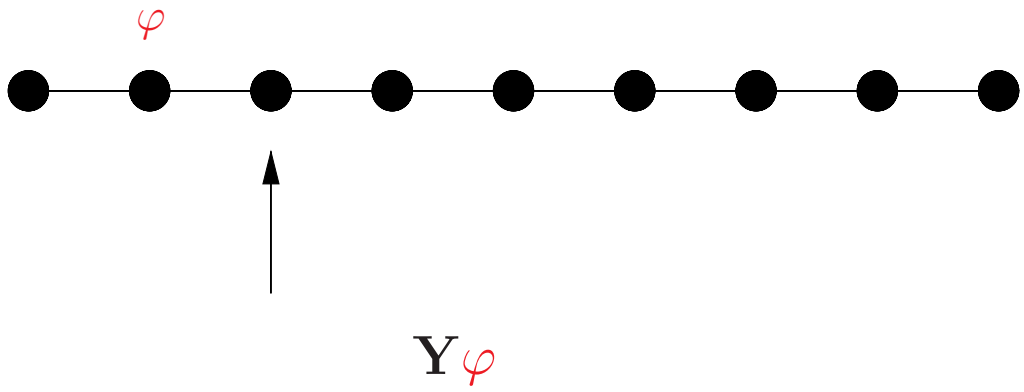


Different syntax means lower complexity: LTL

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:

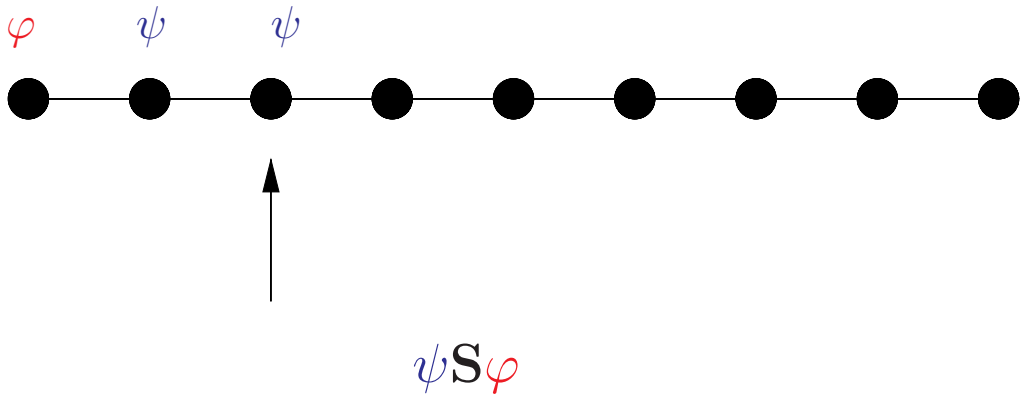


Different syntax means lower complexity: LTL

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

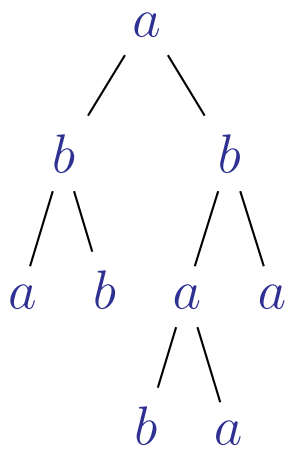
- LTL $=$ FO over strings (more precisely, FO formulae $\varphi(x)$)
- LTL over strings can be evaluated in time $O(\|\varphi\| \cdot |s|)$.
- To evaluate FO with linear data complexity, one needs non-elementary query complexity
  (modulo some complexity-theoretic assumptions; Frick/Grohe 2002)
- Even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

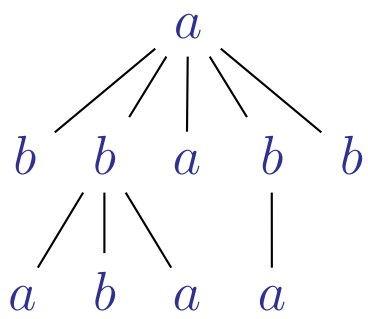
A binary tree:



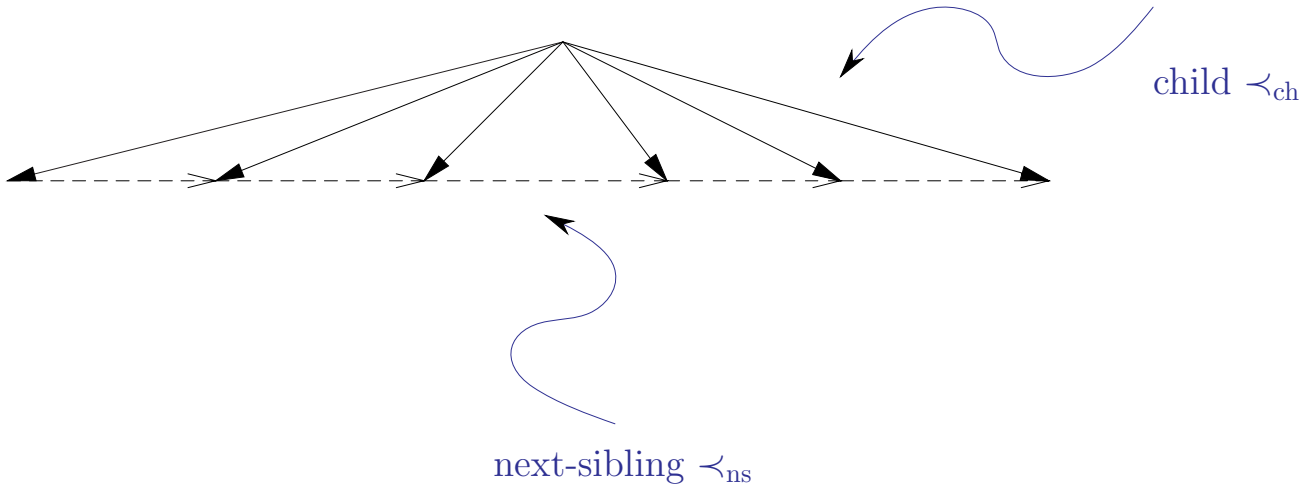
Unranked trees

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:

- $\prec_{\mathrm{ch}}^{*}$ 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

- Extremely important in verification
- Regular (tree) languages = given by (tree) automata.
- Typically characterized via MSO — Monadic Second-Order 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:

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

  (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	$\beta :=$ 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 $\Sigma = \{a, b\}$):

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

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

Syntax:

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

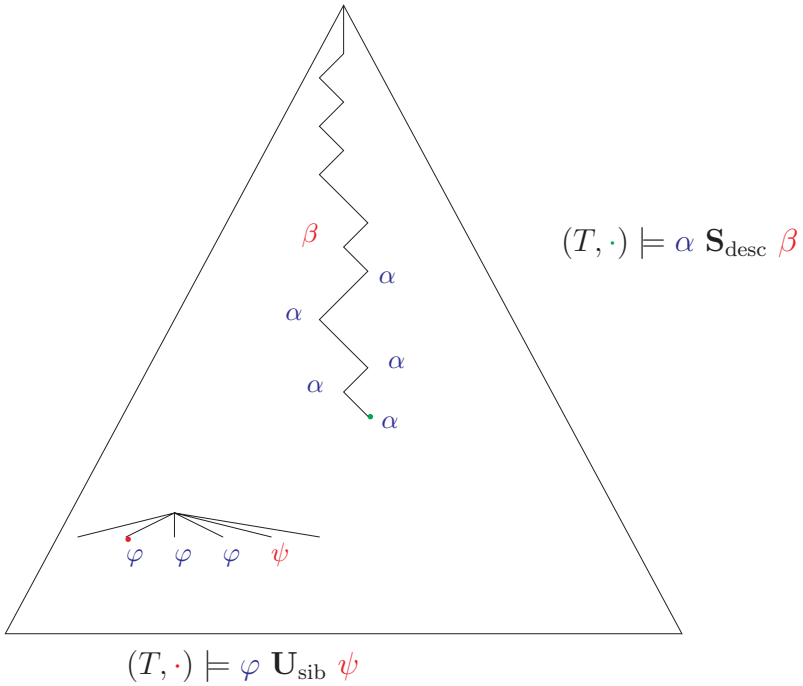
where * is either desc or sib.

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

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

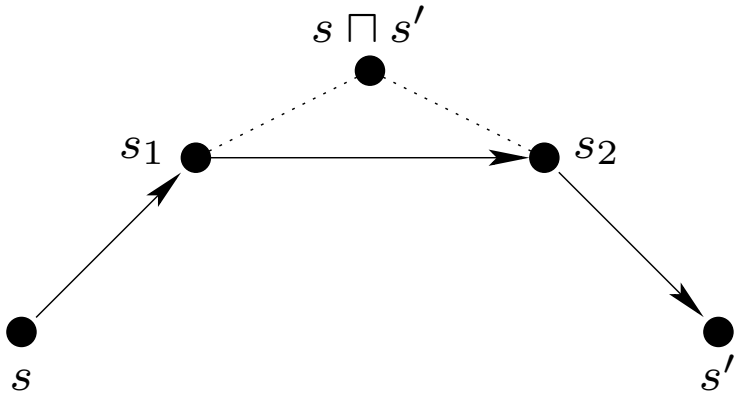
$$\mathsf{TL}^{\mathsf{tree}} = \mathsf{FO}$$

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

n-ary queries

- What if we need to return n-tuples of nodes?
- 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:
  - a set of terms, and
  - node tests

n-ary queries – terms



We have $s \sqcap s'$ (the longest common prefix) and:

- s_1 – successor of $s \sqcap s'$ in the direction of s
- s_2 – successor of $s \sqcap s'$ in the direction of s'
- The 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.

PART II

A Case Study: NESTED WORDS

Alur-Arenas-Barceló-Etessami-Immerman-Libkin, LICS 2007 (AABEIL)

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.
  - Corollary: $\mathbf{FO} = \mathbf{FO}^3$ over ω-words.
- 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

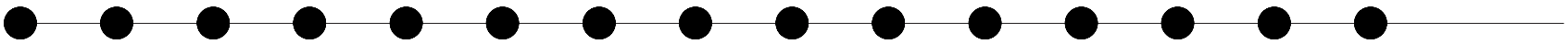
- An execution of a procedural program gives us more than just a linear sequence of program states.
- In addition, we have matching calls and returns.
- Hence we have an ω-word with a nesting structure — nested words.

Adding nesting structure: calls and returns

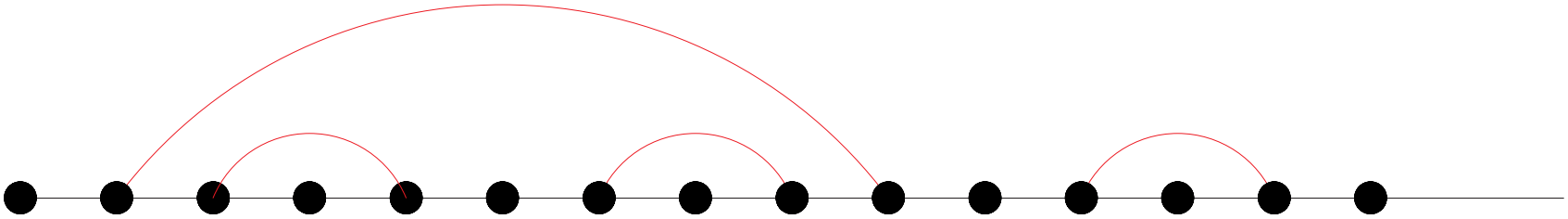
- An execution of a procedural program gives us more than just a linear sequence of program states.
- In addition, we have matching calls and returns.
- Hence we have an ω-word with a nesting structure — nested words.

- 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.

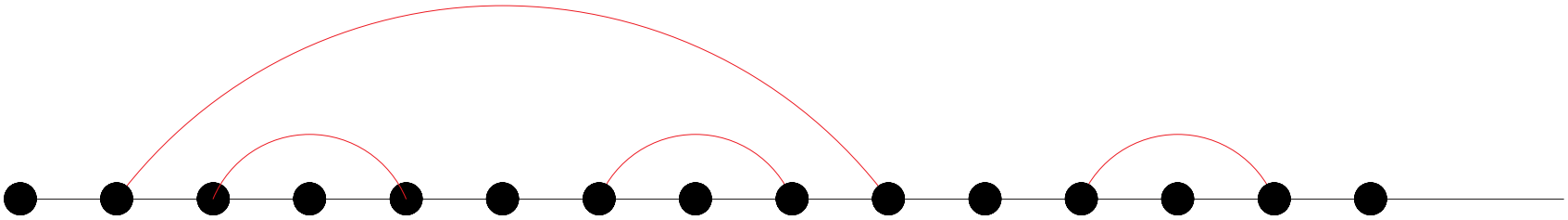
Adding nesting structure



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 $j' < j$.

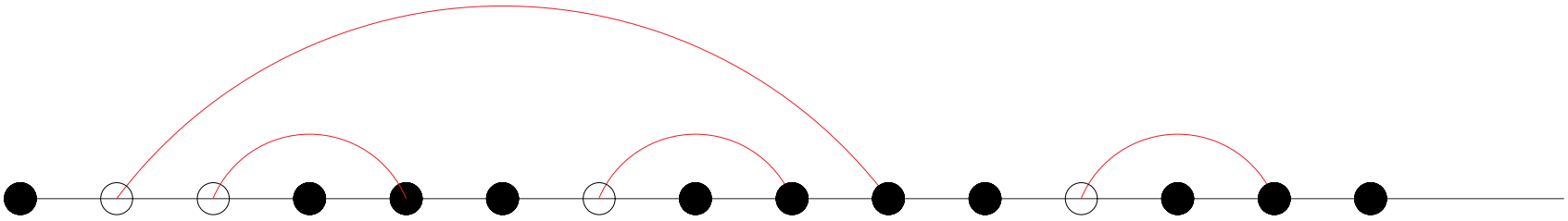
Calls and returns

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.

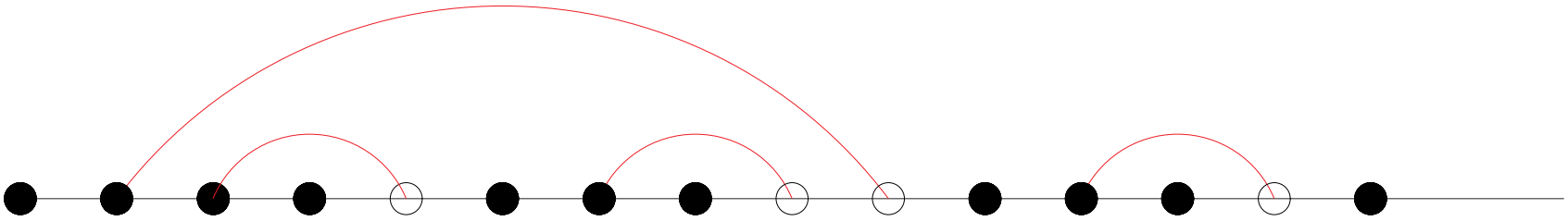
Calls:



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

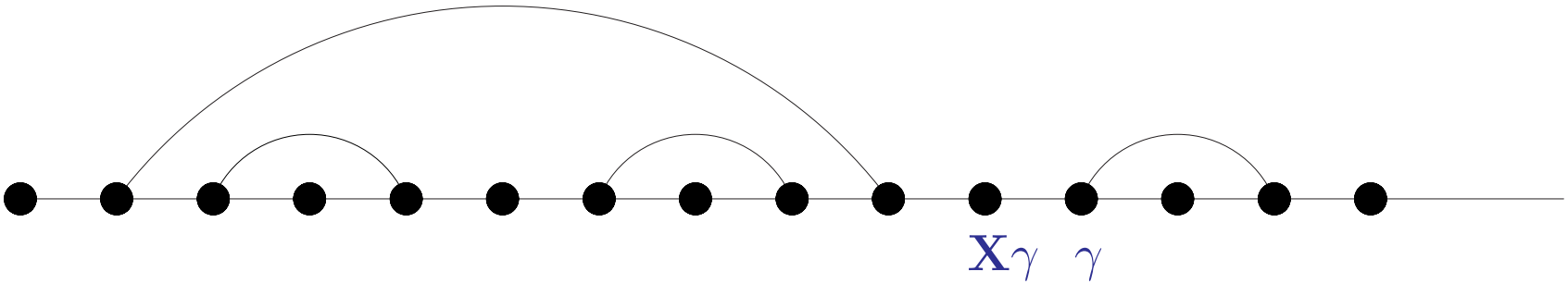
Next operators:

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

Temporal logics for nested words

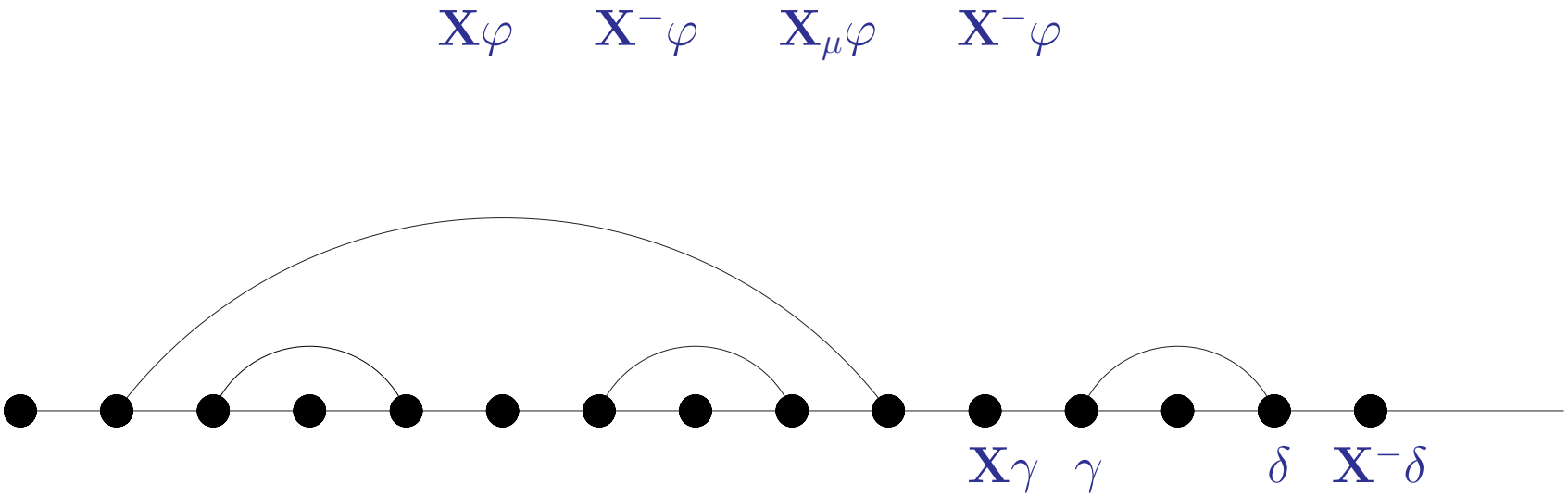
Next/previous operators:

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



Temporal logics for nested words

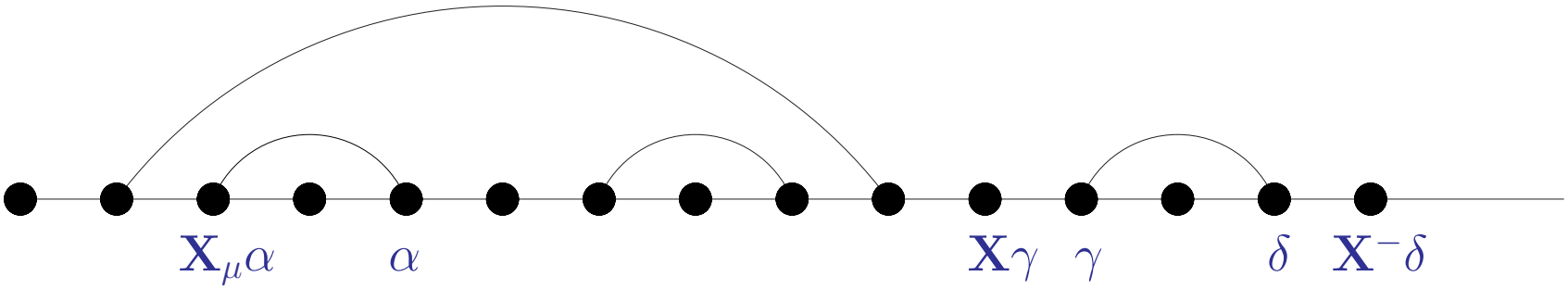
Next operators:



Temporal logics for nested words

Next operators:

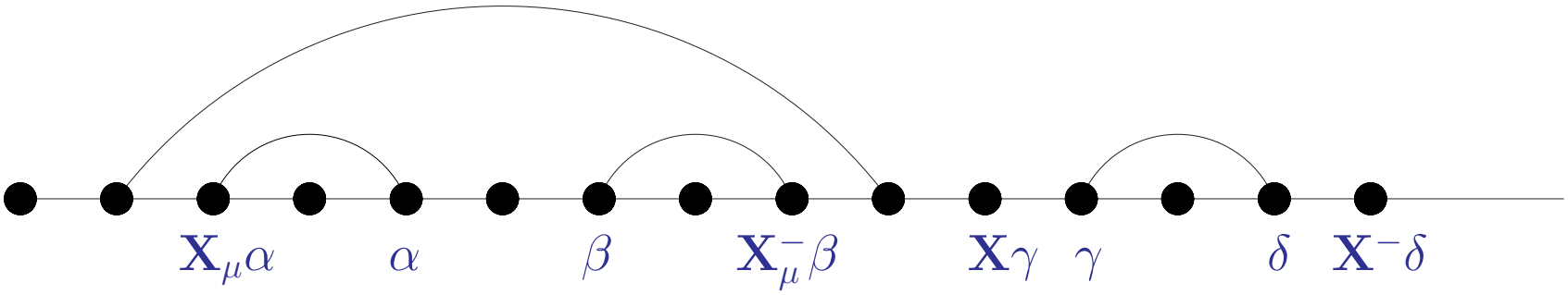
$$\mathbf{X}\varphi \quad \mathbf{X}^{-}\varphi \quad \mathbf{X}_{\mu}\varphi \quad \mathbf{X}^{-}\varphi$$



Temporal logics for nested words

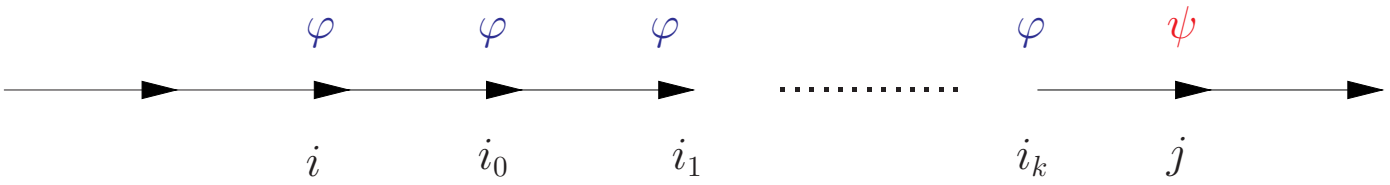
Next operators:

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

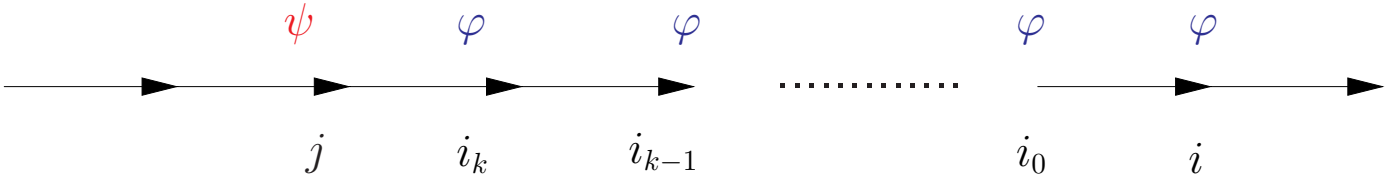


Temporal logics for nested words

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.

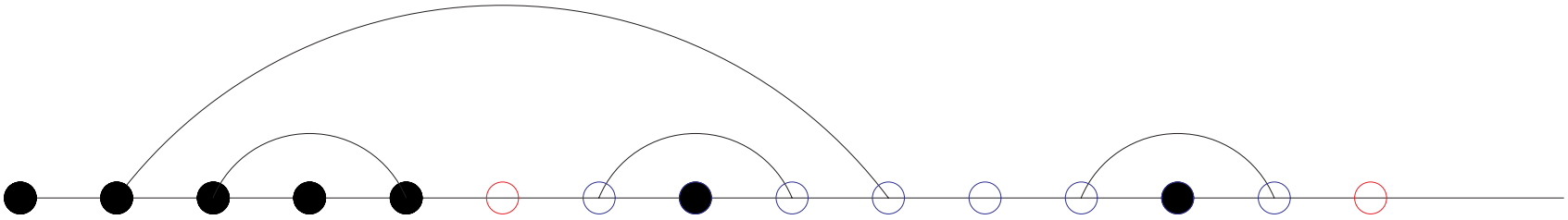
But in addition the nesting structure gives us many new possibilities.

In general, for each type of path:

- between every i and $j > i$ there is at most one path of that type.
- Sometimes there is none.

Paths in nested words

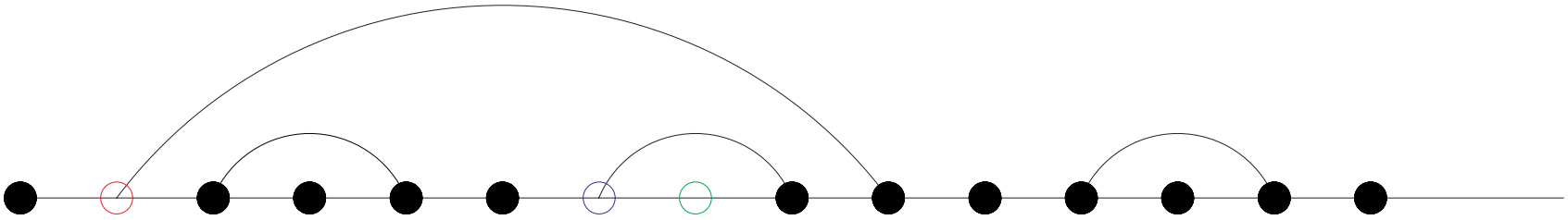
Abstract paths: skip calls



From red to red via blue.

Paths in nested words

Call path: connect positions to their innermost calls.



Path: red–blue–green.

Logic CaRet

Call-Return logic of Alur-Etessami-Madhusudan 2004:

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

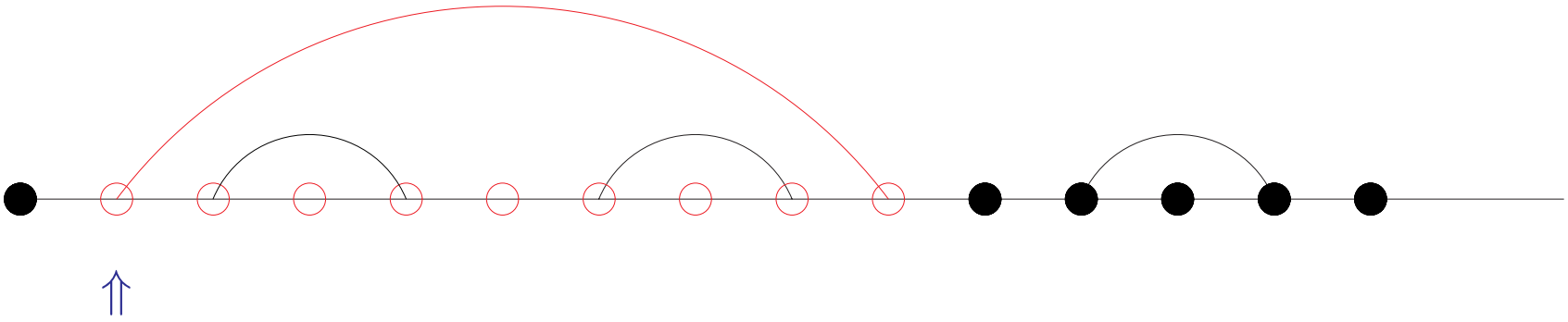
Open problem: Is CaRet expressively complete for FO?

Almost certainly no, but the problem seems to be hard.

Question (2004): what do we add to CaRet to achieve expressive completeness?

Within operator

Add a new within operator to CaRet: $\mathcal{W}\varphi$



$\mathcal{W}\varphi$ is true here φ is true within the call.

Expressive completeness, take 1

Theorem (AABEIL)

CaRet + $\mathcal{W}$ is expressively complete for FO.

Expressive completeness, take 1

Theorem (AABEIL)

CaRet + $\mathcal{W}$ is expressively complete for FO.

Are we happy?

Expressive completeness, take 1

Theorem (AABEIL)

CaRet + $\mathcal{W}$ is expressively complete for FO.

Are we happy? NO!

Because:

Proposition (AABEIL) Model-checking for CaRet + $\mathcal{W}$ is doubly exponential.

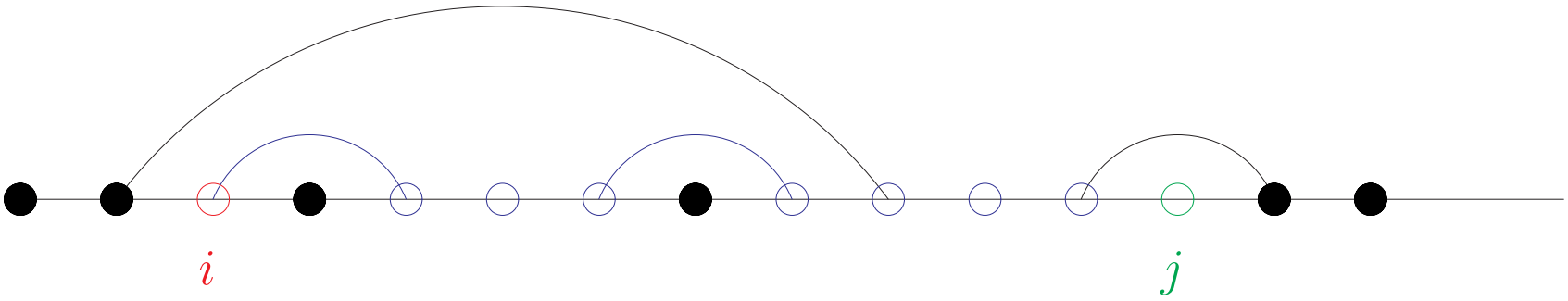
In fact, 2EXPTIME-completeness can be shown.

Back to the drawing board: 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 (AABEIL) NWTL is expressively complete for FO.

Nested Word Temporal Logic NWTL

NWTL includes:

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

Theorem (AABEIL) NWTL is expressively complete for FO.

Are we happy? Yes!

But a few observations first.

NWTL – the 3-variable property

By translating NWTL into FO, we show:

Corollary FO=FO^3 for formulae with at most one free variable.

That is, FO formulae with at most one free variable are equivalent to formulae that use at most 3 variables in total.

Furthermore, we can show:

Theorem (AABEIL)

FO=FO^3 for formulae with at most two free variables.

Proof: pebble games.

NWTL and the past

We know that every FO sentence over words and ω-words is equivalent to an LTL formula without the past (i.e., previous and since), evaluated in the first position.

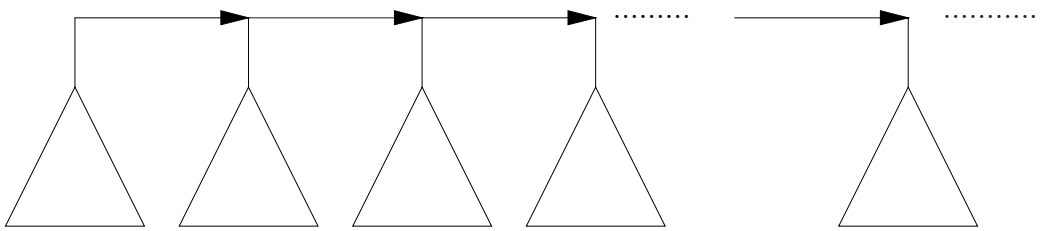
For NWTL, this is only partly true:

Corollary Over nested words, every FO sentence is equivalent to an NWTL formula without the since operator that is evaluated in the first position of the word.

However, previous operators are necessary.

The proof: Marxist composition

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



where all subtrees are finite.

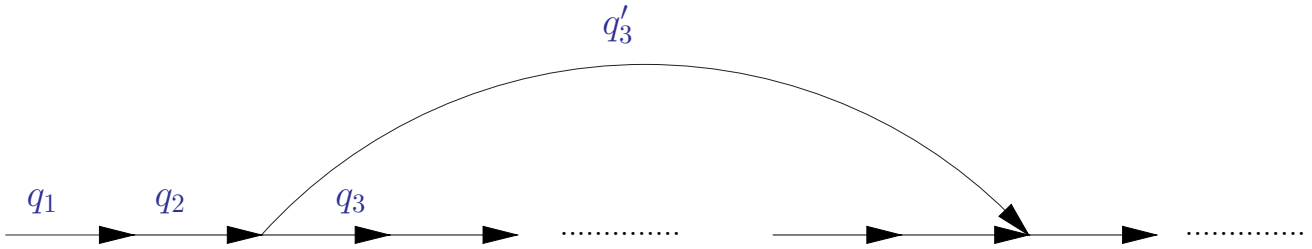
Then an FO formula is, by a composition argument and Kamp's theorem, an LTL formula, where atomic propositions are types of finite subtrees.

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

Towards NWTL model-checking: Nested Word Automata

Usual automata over words propagate the state along successor edges.

Nested word automata, in a call position, propagate a state along the successor edge, and a state along the call-return edge:



A run ensures consistency; we use Büchi acceptance condition.

Nonemptiness is in PTIME.

NWTL model-checking

We throw in all until/since pairs (it's free!) This is the logic NWTL$^+$.

Theorem (AABEIL)
For every NWTL$^+$ formula φ, one can construct an equivalent nested-word automaton $\mathcal{A}_\varphi$ of size

$$2^{O(|\varphi|)}.$$

NWTL model-checking

We throw in all until/since pairs (it's free!) This is the logic NWTL$^+$.

Theorem (AABEIL)
For every NWTL$^+$ formula φ, one can construct an equivalent nested-word automaton $\mathcal{A}_\varphi$ of size

$$2^{O(|\varphi|)}.$$

Corollary The satisfiability problem for NWTL$^+$ is EXPTIME-complete.

NWTL model-checking

We throw in all until/since pairs (it's free!) This is the logic NWTL$^+$.

Theorem (AABEIL)
For every NWTL$^+$ formula φ, one can construct an equivalent nested-word automaton $\mathcal{A}_\varphi$ of size

$$2^{O(|\varphi|)}.$$

Corollary The satisfiability problem for NWTL$^+$ is EXPTIME-complete.

If a model $\mathcal{M}$ is given by, or can be effectively converted into, a nested word automaton, then model-checking is:

- polynomial in the size of $\mathcal{M}$;
- exponential in the size of φ.

The 2-variable fragment

We know that FO=FO^3 over nested words? Can we capture FO^2?

Over ω-words, one enriches FO with the successor operator and captures the X,F-fragment of LTL with FO^2.

A similar result is true for nested words.

Problem: many different notions of successor!

So we have to enrich FO with some of them, and then the next–eventually fragment of NWTL captures FO^2 (AABEIL).

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.