

PATTERN QUERY
LANGUAGES FOR COMPLEX
DATA STRUCTURES

Simone Santini

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Work carried out with **Amarnath Gupta** (UCSD), **Xufei Xian** (SDSC), and **Cristina Bogdansch** (UAM)

Later on I will talk about expressivity of languages using a bunch of different logic systems, so I'd better tell you now from the start what I will use.

$FO[S_1, \dots, S_n]$	First order logic on the relations S_1, \dots, S_n
$FOREG[S_1, \dots, S_n]$	First order logic augmented with regular expressions.
$MSO[S_1, \dots, S_n]$	Monadic second order logic (set variables and quantification over sets).
$MSO_0[S_1, \dots, S_n]$	Monadic second order logic without first order variables (they are replaced with singletons; equivalent to MSO).
$EMSO[S_1, \dots, S_n]$	Existential monadic second order logic (equivalent to MSO on strings and trees)
$MPL[S_1, \dots, S_n]$	(for trees) Monadic path logic: MSO in which second order quantification is restricted to paths.
WS1S	Weak monadic second order logic (second order quantification restricted to finite sets) over the successor relation.
WS2S	Weak monadic second order logic over the two successor relations of binary trees (for finite binary trees it is the same as MSO).
Antichain	MSO in which second order quantification is restricted to sets whose elements are not related by the descendant relation.

WARNING!!! (for trees) Sometimes people distinguish between allowing *vertical* path expressions (along the branches of a tree) and *horizontal* path expressions (in the list of children of a node). I will try to keep things straight should the distinction arise (which, at the time I am writing this, I am not sure of). In particular, the "vertical" FOREG is the same thing as MPL. *I don't like the name FOREG.*

REGULAR EXPRESSIONS ON WORDS

(finite) word domain:

$$\begin{array}{ccc} \{0, \dots, n-1\} & & \{(i, j) | j > i\} \\ \uparrow & & \uparrow \\ \underline{w} = (\text{dom}(w), S, <, Q(a)_{|a \in A}) & & \\ & \swarrow & \downarrow \\ & \{(i, i+1)\} & \{i | a_i = a\} \end{array}$$

(non-deterministic) finite state automaton:

$$\begin{array}{ccc} \text{states} & & \text{final states} \\ \uparrow & & \uparrow \\ \mathcal{A} = (Q, A, q_0, \Delta, F) & & \\ & \downarrow & \\ & \subseteq Q \times A \times Q & \end{array}$$

ϕ : sentence in a logic system $X[S, <, Q(a)]$
(one of those in the previous slide)

$$L(\phi) = \{w \in A^* | \underline{w} \models \phi\}$$

Given $L \subseteq A^*$:

if there is a ϕ in X such that $L = L(\phi)$, then L is $X[S, <]$ -definable, or $X[S]$ -definable if $<$ is not used.

Theorem (Büchi, 1960) *A language of finite words is recognizable by a finite automaton if and only if it is $MSO[S]$ -definable.*

Tiling view: we can see a state machine as a function $\rho : \text{dom}(w) \rightarrow Q$, compatible with the state transition relation and with the input:

$$\begin{array}{cccccccccccc} \boxed{q_0} & \boxed{q_1} & q_2 & q_3 & q_4 & q_5 & \boxed{q_6} & \boxed{q_7} & q_8 \\ a_1 & \boxed{a_2} & a_3 & a_4 & a_5 & a_6 & \boxed{a_7} & a_8 & a_9 \\ & \boxed{q_1} & \boxed{q_2} & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 \end{array}$$

That is, for all i : $(q_{i-1}, a_i, q_i) \in \Delta$, the first state of the series is q_0 , and the last is a final state.

TREES---SOME BACKGROUND

binary trees

Tree: $t : \text{dom}(t) \rightarrow A$
 $\underline{t} = (\text{dom}(t), S_0, S_1, <, Q(a)|_{a \in A})$

$\text{dom}(t) \subseteq \{0, 1\}^*$ such that:
 $\epsilon \in \text{dom}(t)$
 $w0 \in \text{dom}(t) \Rightarrow w \in \text{dom}(t) \wedge w1 \in \text{dom}(t)$
 $w1 \in \text{dom}(t) \Rightarrow w \in \text{dom}(t) \wedge w0 \in \text{dom}(t)$

$$S_0 = \{(w, w0)\} \quad S_1 = \{(w, w1)\}$$

$$< = \{(u, u \cdot v) | v \neq \epsilon\}$$

ϵ is the root of the tree.

unranked trees

Tree: $t : \text{dom}(t) \rightarrow A$
 $\underline{t} = (\text{dom}(t), C, <, Q(a)|_{a \in A})$

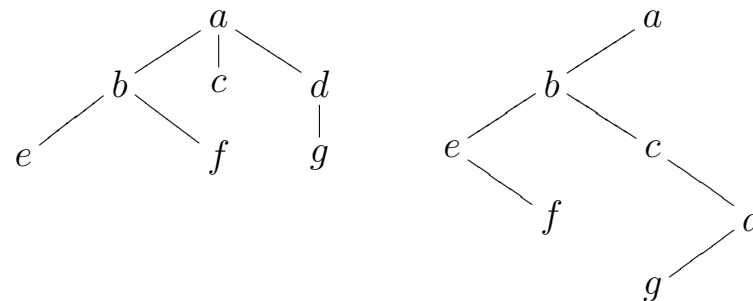
$\text{dom}(t) \subseteq \mathbb{N}^*$ such that:
 $\epsilon \in \text{dom}(t)$
 $u \cdot i \in \text{dom}(t) \Rightarrow u \in \text{dom}(t)$

$$C = \{(u, u \cdot i) | i \in \mathbb{N}\}$$

$$< = \{(u, u \cdot v) | v \in \mathbb{N}^* \wedge v \neq \epsilon\}$$

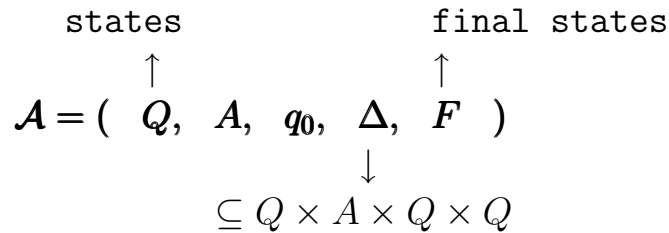
Unranked trees can be represented as binary trees using the "first child/next sibling" representation.

Example:

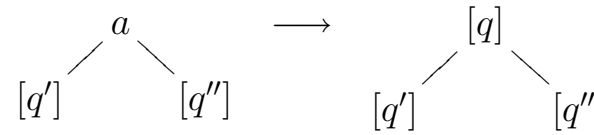


I will focus mainly on binary trees (things are simpler there, and it doesn't make that much conceptual difference)

(binary) tree automaton:

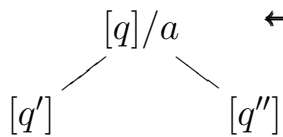
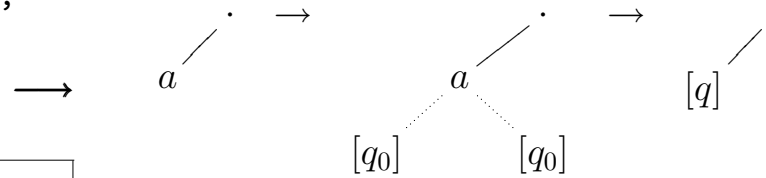


transition $(q, a, q', q'') \in \Delta$



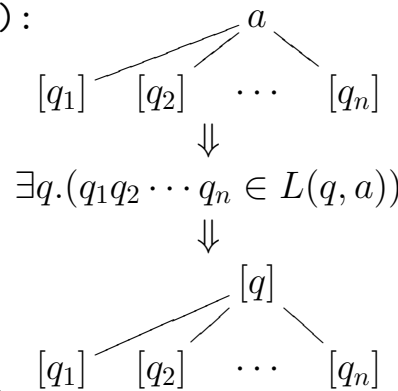
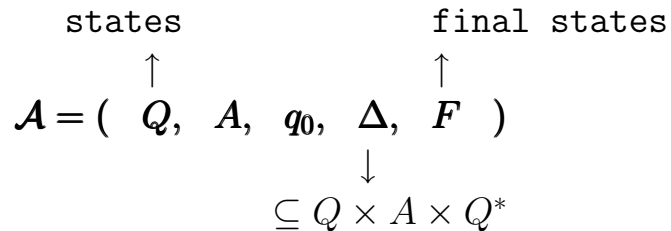
Run: $\rho : \text{dom}(t) \rightarrow Q$

Compatible with the state transition (a tiling, as in the case of strings), initialized at the leaves with transitions (q, a, q_0, q_0)



← For all portions of the tree:
it must be $(q, a, q', q'') \in \Delta$. The run is **accepting** if $\rho(\epsilon) \in F$.

(unranked) tree automaton (more complicated):



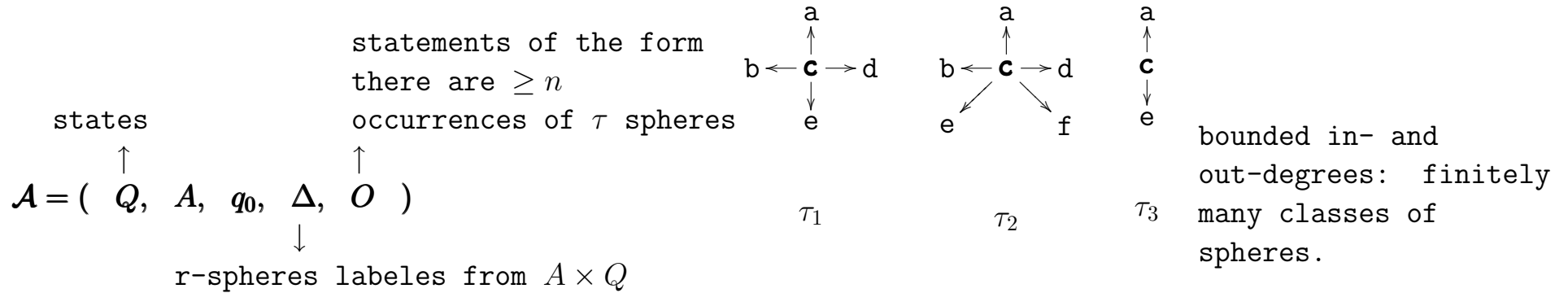
Run: $\rho : \text{dom}(t) \rightarrow Q$

for $v \in \text{dom}(v)$ with n children
 $\rho(v_1)\rho(v_2)\cdots\rho(v_n) \in L(\rho(v), t(v))$

Accepting run: $\rho(\epsilon) \in F$

$L(q, a) = \{u \in Q^* \mid (q, a, u) \in \Delta\}$ regular language

graph accepting automaton:



Run: $\rho : V[G] \rightarrow Q$ such that, given g_q :

$$V[g_q] = \{(v, \rho(v)) \mid v \in V[g]\}$$

$$E[g_q] = \{((v, \rho(v)), (u, \rho(u))) \mid (v, u) \in E[g]\}$$

each r-sphere of g_q is in Δ and their number satisfies O .

if a run exists, we say that the automaton a accepts the graph g , written $a \vdash g$.

THINGS WORK QUITE WELL FOR TREES...

Nondeterministic automata are equivalent to deterministic (Thomas, 1996)---for automata that work in "parallel" bottom-up mode.

On the other hand, *tree walking automata* (in which the control is on only one node at any given moment) are less expressive than finite tree automata. (Bojanczyk and Colcombet, 2005)

theorem: (Thatcher and Wright, 1968): A set of finite trees is recognizable by a finite tree automaton if and only if it is MSO definable.

Regular tree languages have the same closure properties as regular string languages.

theorem: (Potthoff and Thomas, 1993): A set of proper trees (without unary branchings) is recognizable by a finite tree automaton if and only if it is definable in antichain logic.

...NOT SO MUCH FOR GRAPHS

Even limiting ourselves to a narrow class of directed acyclic graphs, such as *pictures* (A picture can be seen as a rectangular grid of vertices, with directed edges joining each vertex with its neighbor on the left and with that below), we observe that non-deterministic automata are strictly more expressive than deterministic ones, that the class of regular languages is not closed under complementation, and that the emptiness problem is undecidable (Thomas, 1996).

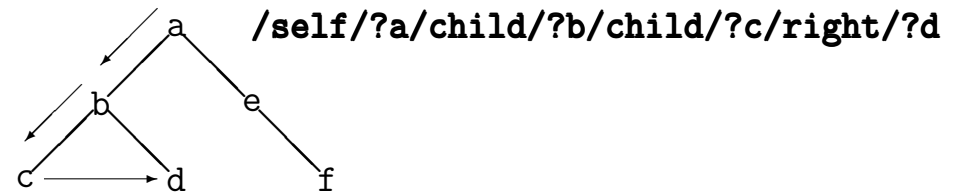
but...

theorem: (Thomas, 1996): A family of graph recognized by a graph recognition automata is definable in existential monadic second order logic.

(Note: on graphs EMSO is strictly less expressive than MSO.)

Core xpath: (xpath without the W3C frills)

```
step ::= child | parent | right | left
pf   ::= step | pf/pf | pf ∪ pf | step* | ?nf
nf   ::=  $p_i$  |  $\top$  | [pf] |  $\neg$ nf | nf ∨ nf | nf ∧ nf
```



based on paths but, because of the axes, a path can "twist around" a tree.

xpath doesn't have a *true* transitive closure operator ("*****"): only transitive closure on the child, parent, right, and left relations (in the actual syntax each one of these closures is a keyword)

Relaxations on the use of the star (Marx, 2005):

conditional xpath:

```
step ::= child | parent | right | left
pf   ::= step | pf/pf | pf ∪ pf | step/?nf* | ?nf
nf   ::=  $p_i$  |  $\top$  | [pf] |  $\neg$ nf | nf ∨ nf | nf ∧ nf
```

regular xpath:

```
step ::= child | parent | right | left
pf   ::= step | pf/pf | pf ∪ pf | pf* | ?nf
nf   ::=  $p_i$  |  $\top$  | [pf] |  $\neg$ nf | nf ∨ nf | nf ∧ nf
```

Expressivity results:

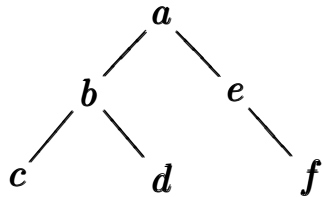
Core xpath (and the W3C kludge): not quite clear to me. Any xpath formula can be translated in a FO[child*, right*] formula in two free variables (Gottlob et al., 2002), but the reverse is not true (Kamp, 1968).

Conditional xpath is equivalent to FO[child*, right*] (Marx, 2005)

regular xpath is obviously more expressive: (child/child)* is not FO; it is, indeed, MPL.

...BUT LET ME TAKE ON A DIFFERENT PROBLEM

xpath is rather inconvenient if you want to take beyond simple paths. Match this structure:



Two ways: using a tortuous path

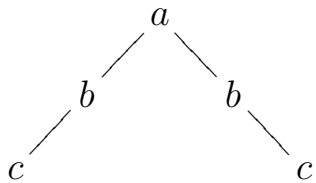
```
/self/?a/child/?b/child/?c/right/?d/parent/?b/right/?e/child/?f
```

(which one has to go back to (almost) the root)

using path filters

```
/self/?a/[?child/?e/child/?f]/child/?b/child/?c/right/?d
```

But.... beware of the filters! The filter expression that matches

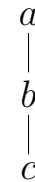


is

```
/self/?a/[child/?b/child/?c]/child/?b/child/?c
```

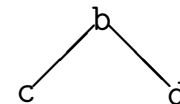
which also matches

→



Moreover... xpath only returns, as a results, the *final* nodes of the paths that satisfy the pattern. What happens if we want to return a whole **structure**?

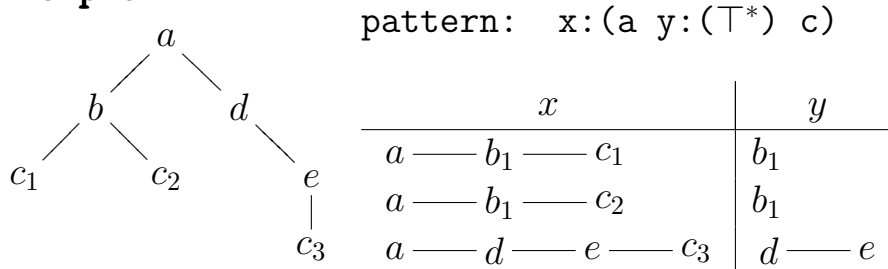
For instance, in the first example, whenever the path matches the tree, it should return the fragment on the side here:



SECOND PROBLEM FIRST!

Our first language... let us call it LT1 (we haven't got around to giving names to them yet): path language based on regular expressions; we simplify it (for the moment) giving it only one axis (down, or following-in-a-branch) so that, rather than a path language, it is really a **branch** language.

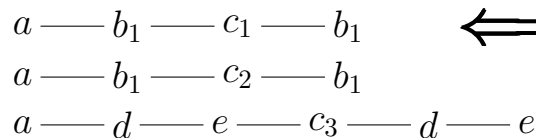
Example:



$$\cup \{x^2 | x \leftarrow [1, 2, 3]\} = \{2, 4, 9\}$$

$$++ \{x^2 | x \leftarrow [1, 2, 3]\} = [2, 4, 9]$$

$$\cup \{x \cdot y | g \vdash x : (a y : (T)^* c), g \leftarrow G\}$$



Syntax (quite simple):

$$\pi ::= C \mid T \mid \perp \mid \pi\pi \mid \pi^* \mid \pi|\pi \mid v:\pi \mid (\pi)$$

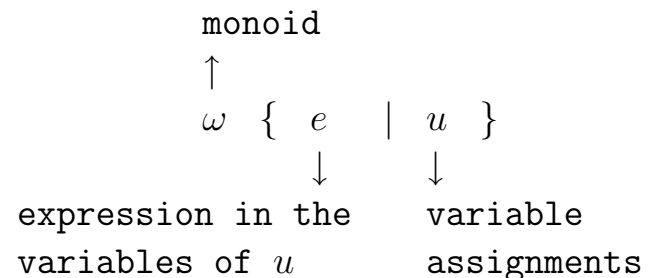
$v:\pi$ assigns the results of matching the pattern π to the variable v . If there are several variables the matcher produces a tuple of structures corresponding to the same match of the pattern.

\perp is the expression that doesn't match anything: useful to express the fact that a path must terminate with a leaf.

Apart from the variables, the language is regular expressions on paths. It is, in fact, equivalent to MPL.

Interlude: how do we use the variable values?

The pattern language is embedded in the *monoid comprehension calculus* (Fegaras and Maier, 1998):

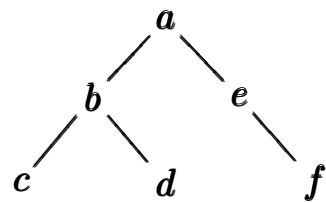


...AND NOW LET'S START TO MAKE THINGS INTERESTING

Branching operator:

$a[\pi_1, \pi_2]$ matches

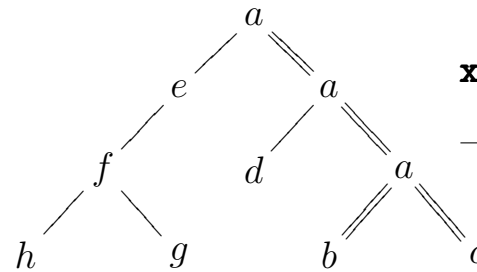
Now we can match the tree



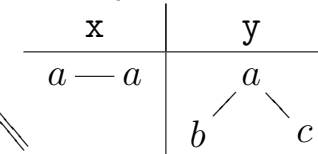
with the expression $a[b[c,d], e[\perp, f]]$

Note: So far the use of the star is restricted to paths: in the second example we don't match the complete tree, only its path components. The monoid comprehension gives us a way to recompose the tree from its pieces using the *merge* operator. Still, the question remains: is it possible to create a "structural" star and, if so, will it give us the expressive power of finite tree automata?

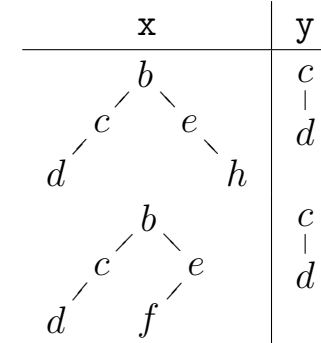
Three easy pieces:



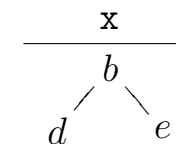
$x: (a^*)y: a[b, c]$



$a[x:b[T^*, T^*], y:cd\perp]$

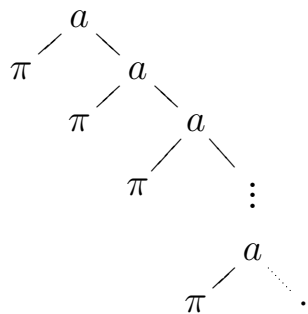


$x: (T[T\perp, T])$



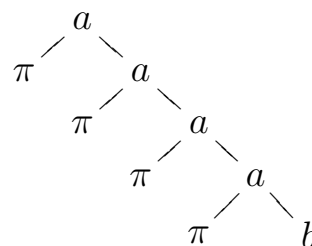
We will need two stars: one for recursion on the left child, one for recursion on the right one (this is **LT2**).

a[π , *]



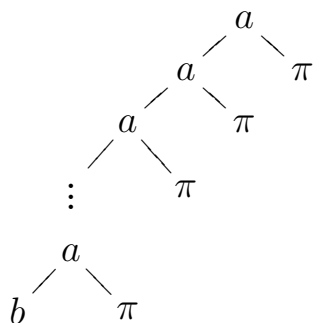
Note: the final element of the tree is left "dangling": it is necessary to finish a pattern with something else to close the tree:

a[π , *]b



there is, of course a symmetric one:

a[* , π]b



stipulation: ϵ (the empty tree) satisfies the pattern.

the equational theory of the "structural star" is very similar to that of the star in regular languages:

$$\epsilon \mid \mathbf{a}[\mathbf{a}[* , \pi]] = \mathbf{a}[* , \pi]$$

$$\mathbf{a}[* , \pi] \mathbf{a}[* , \pi] = \mathbf{a}[* , \pi]$$

Expressiveness: ? but we know that it is at least MPL (on paths we have a regular language) and the equational theory is rich enough to make us hope for translation of finite automata, that is, WS2S or, equivalently on finite trees, MSO (Genevès and Layaïda, 2007).

Some examples...

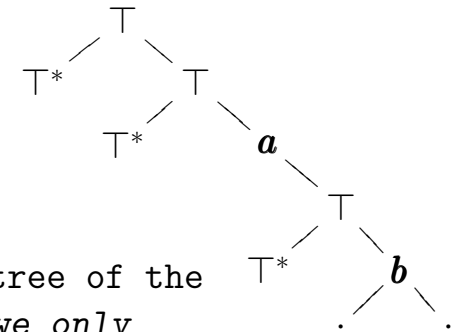
All the subtrees that start with a node **a**:

$x:a[*,*]\perp$

(note the \perp after the recursion: it causes the previous statement to match only *complete* subtrees, up to the leaves.

the common ancestor of all nodes **b** and **c** that are not directly comparable

$x:T[T^*b,T^*c]$

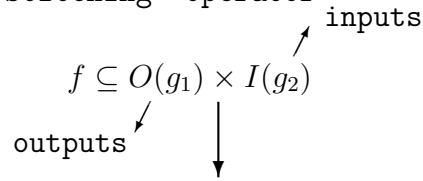


trees in which the same sub-tree of the root contains **a** and **b** where we *only return the path that contains them*:
 $T[x:(T^*aT^*b), T^*] \mid T[x:(T^*bT^*a), T^*] \mid$
 $T[T^*, x:(T^*aT^*b)] \mid T[T^*, x:(T^*bT^*a)]$
 (this we can actually answer with xpath...)

trees in which the same sub-tree of the root contains **a** and **b** where we *return the whole sub-tree containing the paths*
 $x:T[*,T^*]. a . T[*,T^*] . b . T[*,*] \mid$
 $T[T^*,*]. a . T[T^*,*] . b . T[*,*] \mid$
 $T[*,T^*]. b . T[*,T^*] . a . T[*,*] \mid$
 $T[T^*,*]. b . T[T^*,*] . a . T[*,*]$

Trivial point that will come handy shortly: clearly given a specific tree T it is always possible to define a path π such that $T \models \pi$. And this is true even in xpath... (!)

for the sequencing of graphs we have to define a suitable "stitching" operator



...AND LET US MOVE ON TO (acyclic) GRAPHS

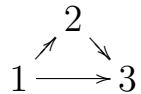
LG1: $\pi ::= \mathbf{C} \mid \pi \cdot \pi \mid \pi^* \mid \pi[\pi, \dots, \pi]\pi$

LG2: $\pi ::= \mathbf{C} \mid \pi \cdot \pi \mid \pi|\pi \mid \pi^* \mid \pi[\pi, \dots, \pi]\pi$
(LG1 with disjunction)

beware! For graphs, the trivial point of the previous slide is no longer true.

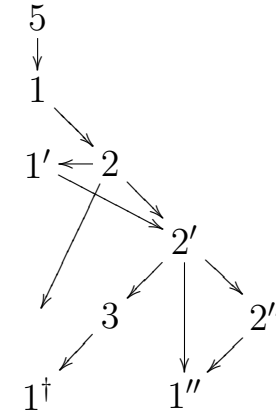
$$L(\pi) = \{g : g \models \pi \wedge \forall g'. (g' \models \pi \wedge g' \subseteq g \Rightarrow g' = g)\}$$

There are graphs that do not belong to any $L(\pi)$, for any π . General property: count the number of patterns on n nodes and the number of acyclic graphs with n nodes. If the empty pattern is not allowed (as it is not so far), then the graph



is not in any $L(\pi)$

Example:



The pattern $1 \cdot 2^* \cdot 1$ matches:

$$1 \rightarrow 2 \rightarrow 1'$$

$$1 \rightarrow 2 \rightarrow 1^\dagger$$

$$1 \rightarrow 2 \rightarrow 2' \rightarrow 1''$$

$$1 \rightarrow 2 \rightarrow 2' \rightarrow 2'' \rightarrow 1''$$

Theorem: The class of graphs that are expressible by an LG1 pattern is the class of connected minimal vertex series parallel graphs

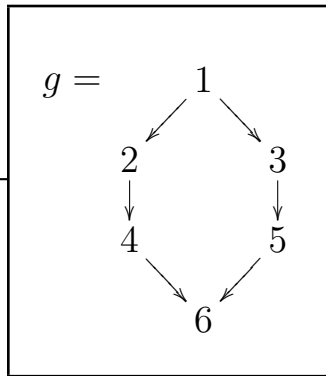
Theorem: The class of graphs that are expressible by an LG2 pattern is the class of minimal vertex series parallel graphs

theorem: given a pattern π , there is a graph automaton that recognizes $L(\pi)$, that is, an automaton $a(\pi)$ such that for all graphs g , $a(\pi) \vdash g$ if and only if $g \models \pi$.

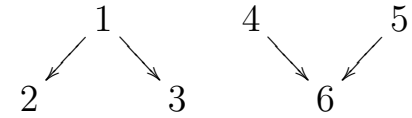
corollary: let G be a set of graphs. If there is a pattern π such that $G = L(\pi)$, then G is definable in existential monadic second order logic.

theorem: the emptiness problem for $L(\pi)$ is decidable.

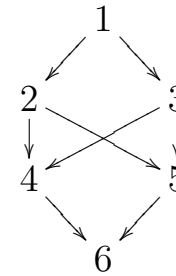
strong suspicion: Patterns are at least as expressive as MPL on graphs.



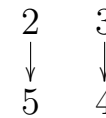
$$\cup \{(p, q) | g \vdash p : (1[2, 3]), g \vdash q : ([4, 5]6), g \leftarrow \text{DB}\}$$



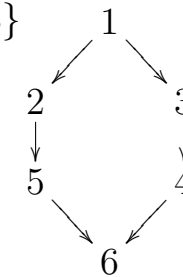
$$\cup \{p \cdot q | g \vdash p : (1[2, 3]), g \vdash q : ([4, 5] \cdot 6), g \leftarrow \text{DB}\}$$



$$\cup \{(r \cdot s, t \cdot w) | g \vdash p : ([1, 2] \cdot 3), g \vdash q : ([4, 5] \cdot 6), g \leftarrow \text{DB}\}$$



$$\cup \{[g] \{p, q, r, s, t, w\} | g \vdash p : ([r : 1, t : 2] \cdot 3), g \vdash q : ([w : 4, s : 5] \cdot 6), g \leftarrow \text{DB}\}$$

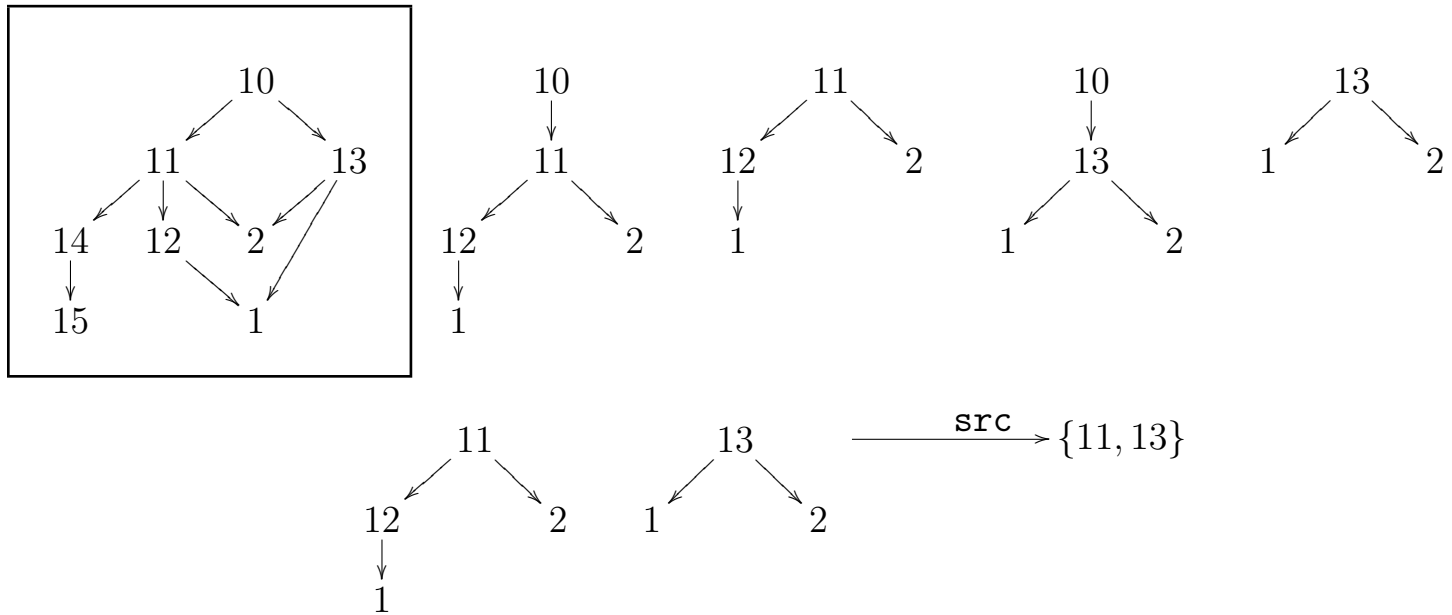


$$\cup \{p | g \vdash p : (1[2 \cdot 5, 3 \cdot 4] \cdot 6), g \leftarrow \text{DB}\}$$

Graph operations: *gmin*, *gmax*: min and max in the sense of graph inclusion
 $\cdot \cdot$: graph append;
 $[g]$: union of the node sets with the edges that connect them.
 src: returns the nodes with no incoming edges;
 snk: returns the nodes with no outgoing edges.

least common ancestors of the nodes with id=1 and id=2

$$\cup\{\text{src}(x) \mid x \leftarrow \text{gmin}\{a \mid x \vdash a : (\top[\top^* \cdot (\text{id} = 1), \top^* \cdot (\text{id} = 2)]), x \leftarrow \text{DB}\}$$



5 in the case discussed here: we have identified a recursively defined class of graphs and created a pattern language that generates the corresponding patterns (OK... we did it the other way around...)

2 xpath & friends did not take on the lessons of pattern languages for strings (weird, for an organism obsessed with standardization...)

3 There is no reason why a tree pattern language should be limited to paths: we have the theoretical instruments to define and study languages based on tree pattern matching.

4 on graphs the situation is stickier; for one thing, it seems as if there are too many graphs for a (recursively defined) pattern language to define all possible patterns.

1 the "standard" patterns languages on trees are, I daresay, a disaster; they have limited an uncertain expressive power, and contain bizarre and arbitrary limitations.

PEARLS OF WISDOM

6 Note that we did not try to define a "surface" query language with a *select*, a *from*, and a *where*; these days the queries are complicated, and even in a surface language they are a mess; most of them are written by a program so... why bother?

