La vérification de propriétés de graphes exprimées en logique du second-ordre monadique.

Verification of monadic second-order graph properties

Bruno Courcelle

Université Bordeaux 1, LaBRI & Institut Universitaire de France

References:

B.C. and J. Engelfriet : *Graph structure and monadic second-order logic*, book to be published by Cambridge University Press, May 2012.

B.C. and I. Durand: *Fly-automata, their properties and applications,* 16th CIAA, 2011, LNCS 6807, pp. 264 – 272.

All are readable on : http://www.labri.fr/perso/courcell/ActSci.html

Summary

1. Monadic Second-Order (MSO) logic

2. Graph decompositions, tree-width and clique-width

3. From MSO formulas to automata

4. Practical difficulties and (some) remedies

5. Tree-width and MSO formulas with edge quantifications

6. Open problems and conclusion.

Two ways of considering graphs

A graph (finite, up to isomorphism) is an *algebraic object*,

an element of an algebra of graphs

(similar to words, elements of monoids)

A graph is a *logical structure* ;

graph properties can be expressed by logical formulas

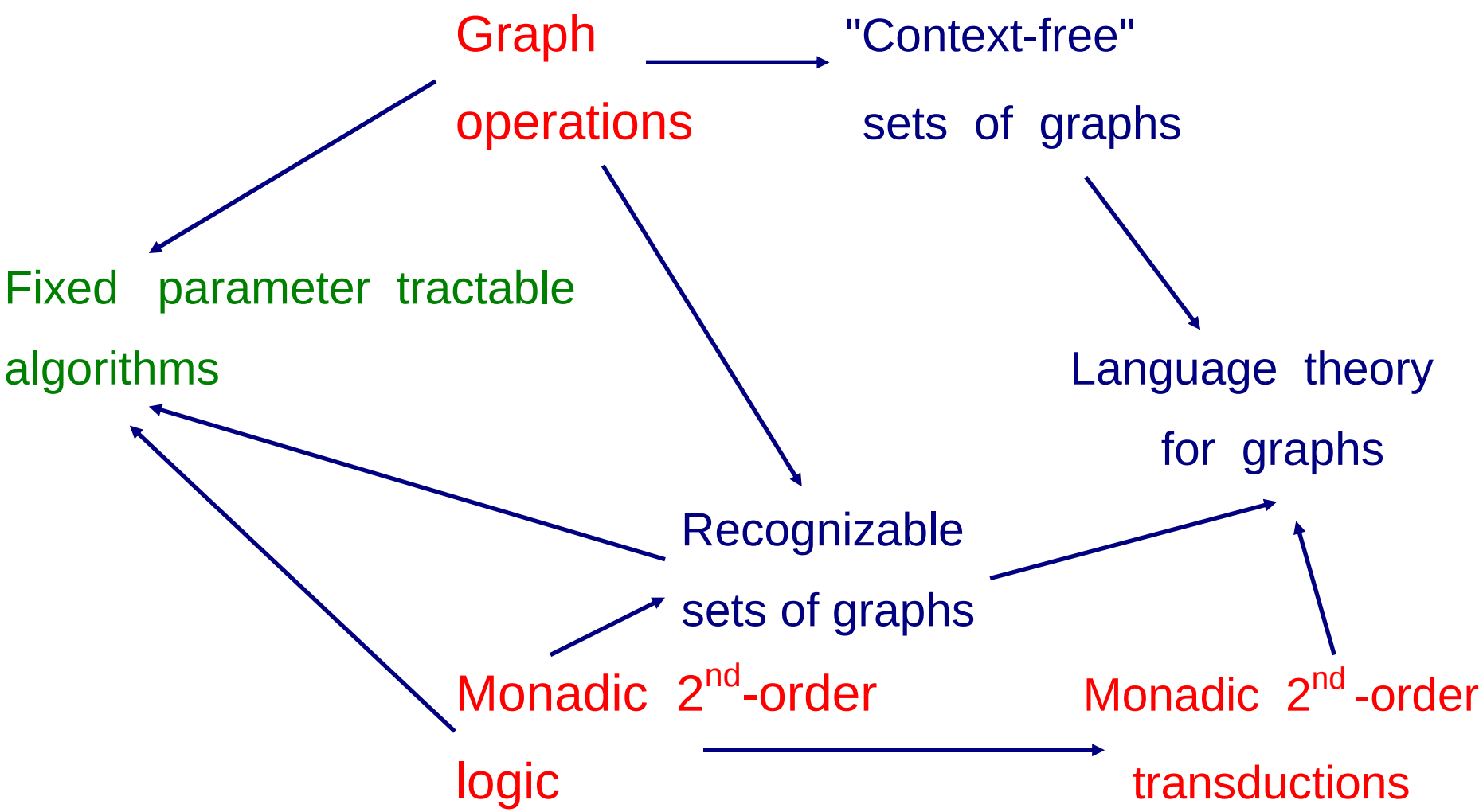
(FO = first-order, MSO = monadic second-order)

Consequences:

a) *Language Theory* concepts extend to graphs

b) *Algorithmic meta-theorems*

An overview chart



1. Monadic Second-Order (MSO) Logic

Graphs as logical structures: G is directed or undirected, and *simple*

G = (V_G , $edg_G(.,.)$) with $edg_G(u,v)$ $\Leftrightarrow$ there is an edge u $\rightarrow$ v (or u – v).

We cannot write an equality if G is not simple. Graphs with multiple edges will be considered later.

Monadic second-order logic

= First-order logic on power-set structures

= First-order logic extended with (quantified) set variables denoting subsets of the domains.

For graphs, set variables denote sets of vertices.

("An arbitrary set of edges" is here a binary relation over V_G).

MSO (expressible) properties : k-colorability, transitive closure, properties of paths, connectivity, planarity

Examples for G = (V_G , $edg_G(.,.)$), undirected

Syntax is clear; shorthands are used (example $X \cap Y = \emptyset$).

(1) *G is 3-colorable :*

$$\exists X,Y \, (X \cap Y = \emptyset \; \wedge$$
$$\forall u,v \, \{ edg(u,v) \Rightarrow$$
$$[(u \in X \Rightarrow v \notin X) \wedge (u \in Y \Rightarrow v \notin Y) \wedge$$
$$(u \notin X \cup Y \Rightarrow v \in X \cup Y) \;]$$
$$\})$$

(2) *G is not connected :*

$$\exists\, Z\, (\, \exists x \in Z \ \wedge\ \exists y \notin Z \ \wedge\ (\forall u,v\, (u \in Z \ \wedge\ \mathrm{edg}(u,v) \Rightarrow v \in Z)\ )$$

(3) *Transitive and reflexive closure* : **TC**(R ; x, y) :

$$\forall\, X\, \{\, \text{“X is R-closed”} \ \wedge\ x \in X \ \Rightarrow y \in X\ \}$$

where “X is R-closed” is defined by :

$$\forall u,v\, (u \in X \ \wedge\ R(u,v) \Rightarrow v \in X)$$

The relation R can be defined by a formula as in (where Y is free) :

$$\forall x,y\, (x \in Y \ \wedge\ y \in Y \Rightarrow\ \mathbf{TC}(\text{“}u \in Y \ \wedge v \in Y \wedge \mathrm{edg}(u,v)\text{”} ; x, y)$$

expressing Conn(Y) i.e. that G[Y] is connected.

(4) Minors : G contains a fixed graph H as a minor with $V_H = \{1,\ldots,p\}$:

there exist disjoint sets of vertices $X_1,\ldots, X_p$ in G

such that each $G[X_i]$ is connected and,

whenever $i - j$ in H, there is an edge between X_i and X_j.

(5) Planarity is expressible : no minor K_5 or $K_{3,3}$ (Kuratowski-Wagner).

(6) Has a cycle (for G without loops) :

$\exists x,y,z$ [edg(x,y) $\wedge$ edg(y,z) $\wedge$ "there is a path from x to z avoiding y"]

(7) Is a tree : connected without cycles.

Provably non-expressible properties

- G is isomorphic to $K_{p,p}$ for some p (*not fixed*; needs equal cardinalities of two sets, hence quantification over binary relations to find a bijection).

- G has a nontrivial automorphism, or has all vertices of same degree.

- Card(X) is a multiple of p. (But this is possible if the graph is linearly ordered or if some linear order is definable by an MSO formula).

Definition : Adding these *cardinality set predicates* to MSO logic gives Counting Monadic Second-Order (CMSO) logic: all good properties of MSO logic hold for it.

(Adding an *equicardinality* set predicate would spoil everything.)

Exercises

1) Write an MSO-formula expressing that a simple *directed* graph a rooted tree.

2) Write an MSO-formula with free variables x,y,z expressing that, in a tree (undirected and unrooted), if one takes x as root, then y is an ancestor of z.

3) A nonempty word over alphabet {a,b} can be considered as a directed path given with unary relations lab_a and lab_b representing the sets of occurrences of letters a and b. Prove that every regular language over {a,b} is MS-definable.

4) A complete bipartite graph $K_{n,m}$ has a Hamiltonian cycle iff n=m. Construct such a graph "over" any word in $\{a,b\}^+$ having at least one a and at least one b. (It follows that Hamiltonicty is not MS-expressible because the converse of 3) holds).

5) Write an MSO-formula expressing *acyclic p-colorability* : every two colors induce a forest.

Two problems for a class *C* of finite graphs and a logic:

Decidability? : Does a given sentence hold in some (or all) graphs of *C* ?

Model-checking (decidable) : Its time/space complexity ?

Language, class	*Decidability*	*Model-checking*
FO, all graphs	Undecidable	Polynomial-time
CMSO, clique-width < k	Decidable	Cubic-time
CMSO, unbdd cwd.	Undecidable	Conjecture : not FPT (*)

(*) A related fact is proved by S. Kreutzer (LICS 2010) for unbounded tree-width and MSO formulas with edge quantifications.

(The exact statement is very technical.)

Model-checking problems

The 3-Coloring problem is NP-complete and MSO-expressible.

Fixed-Parameter Tractability (FPT) :

Definitions: (1) An algorithm taking time $f(k).n^c$ for some fixed function f and constant c is *FPT*.

The value k is a *parameter*: degree, diameter, tree-width, clique-width, etc. The "hard part" of the time complexity depends on some function f (arbitrary in the definition; in practice, it must be "limited") whereas the size of the input is n. So, the "main factor" is polynomial.

(2) An algorithm is in the class *XP* if it takes time $O(n^{g(k)})$.

It takes polynomial time for fixed k, with a degree depending on k.

3-colorability again :

It is NP-complete for (even planar) graphs of degree ≤ 4 (Dailey, 1980)

Hence the *degree* is *not* a good parameter.

Tree-width is good because the time complexity is $O((2^{32.k.k.k} + 3^k).n)$.

(The term $2^{32.k.k.k}$ is for constructing a tree-decomposition).

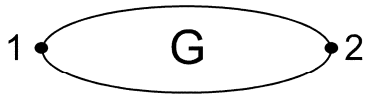
For MSO properties the good parameters are tree-width and clique-width : both are based on hierchical decompositions.

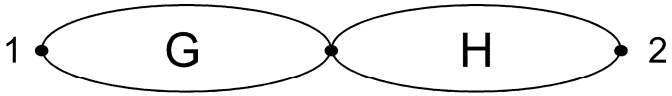
2. Graph decompositions : tree-width and clique-width.

Example 1: Directed series-parallel graphs (tree-width 2 ; trees have tree-width 1)

Graphs with distinguished vertices marked 1 and 2, generated from

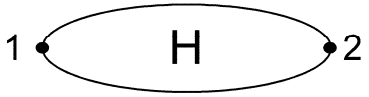
e = 1 → 2 by the operations of *parallel-composition* // and *series-composition* •

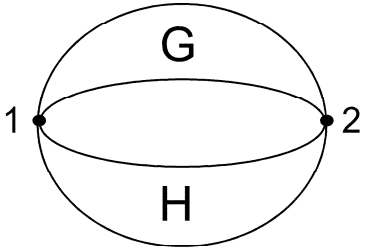




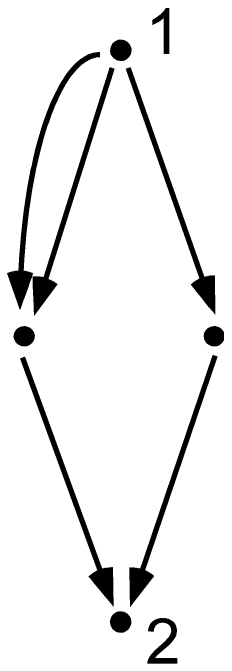
G • H

((e // e) • e) // (e • e)





G//H



The defining equation is S = S // S ∪ S • S ∪ { e }

Example 2 : Cographs (clique-width 2 ; trees have clique-width ≤ 3)

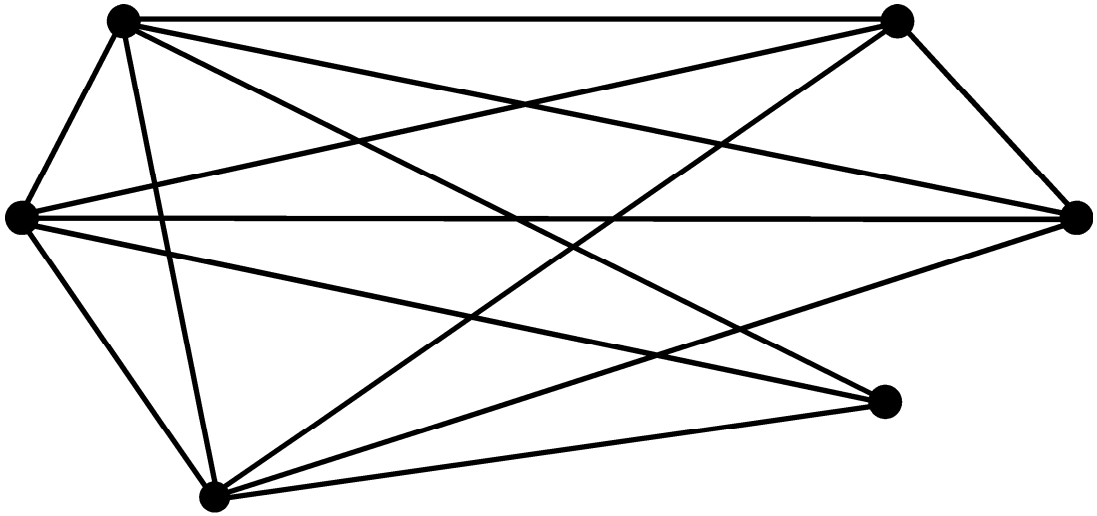
Undirected graphs generated by $\oplus$, *disjoint union* and $\otimes$, *complete join* from **a**, a vertex without edges (up to isomorphism); $\otimes$ is defined by :

$G \otimes H = G \oplus H$ with "all possible" undirected edges between G and H,

Cographs are recursively defined by : $C = C \oplus C \cup C \otimes C \cup \{ \mathbf{a} \}$

Example :

$(\mathbf{a} \otimes \mathbf{a} \otimes \mathbf{a}) \otimes ((\mathbf{a} \otimes \mathbf{a}) \oplus \mathbf{a})$



Definition : *Clique-width*

More powerful than *tree-width* ; the construction of automata is easier.

Graphs are *simple*, directed or not.

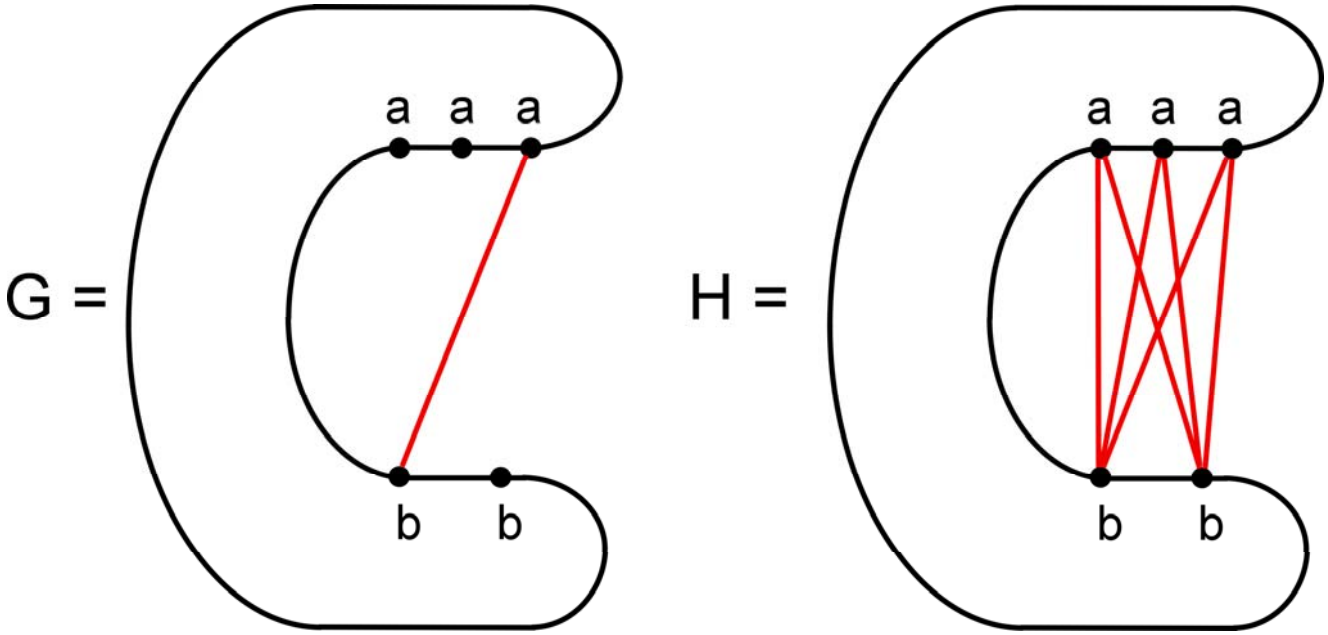
We use labels: *a, b, c, ..., d*. Each vertex has one and only one label ; *several* vertices may have the same label. We call x labelled by *a* an *a-port*

One binary operation : *disjoint union* : $\oplus$

Remark : If G and H are not disjoint, we replace H by an isomorphic disjoint copy to define G $\oplus$ H. Hence G $\oplus$ H is well-defined *up to isomorphism*. No such problem in a “decomposition approach”.

Unary operations: Edge-addition denoted by $Add_{a,b}$

Addition of undirected edges: $Add_{a,b}(G)$ is G augmented with edges *between* every *a*-port and every *b*-port.



H = $Add_{a,b}(G)$; only 5 edges added

The number of added edges depends on the argument graph.

Addition of directed edges: $\overrightarrow{Add}_{a,b}(G)$ is G augmented with edges *from* every *a*-port to every *b*-port.

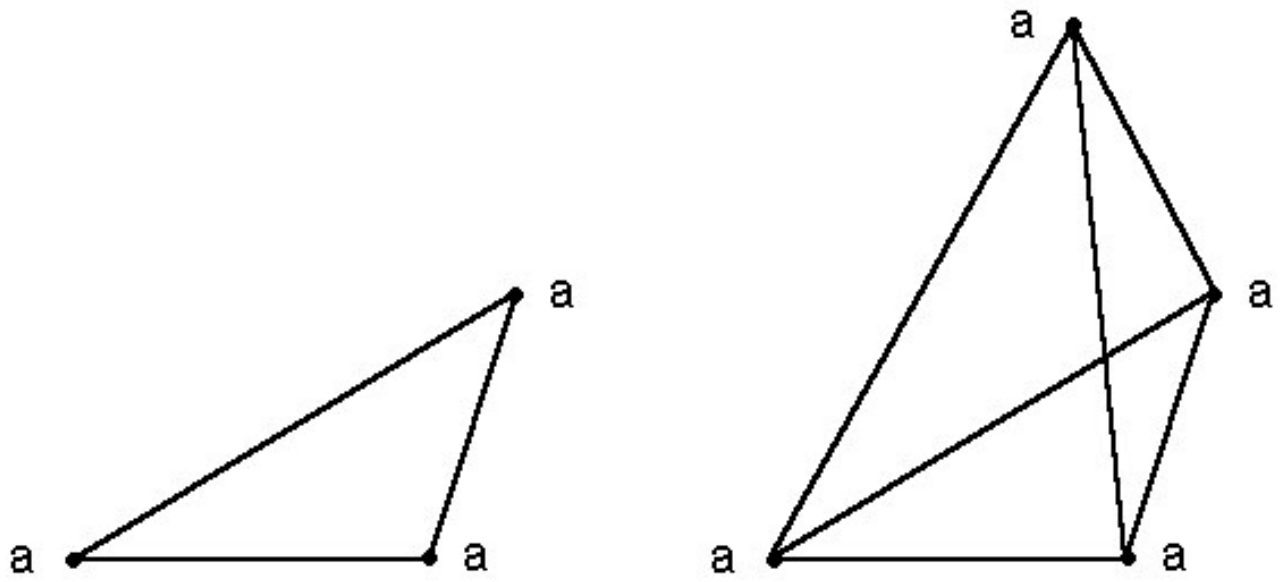
Vertex relabellings :

$Relab_{a \longrightarrow b}(G)$ = G with every vertex labelled by *a* relabelled into *b*

Basic graphs : those with a single vertex.

Definition: A graph G has clique-width $\leq k \Leftrightarrow$ it can be constructed from basic graphs with the operations $\oplus$, $Add_{a,b}$ (or $\overrightarrow{Add}_{a,b}$) and $Relab_{a \longrightarrow b}$ by using $\leq k$ labels. Its clique-width *cwd*(G) is the smallest such *k*

Example : Cliques have clique-width 2 (and *unbounded tree-width*)



K_n is defined by t_n where $t_{n+1} = Relab_b \longrightarrow_a (Add_{a,b}(t_n \oplus \mathbf{b}))$

Cliques are defined by the equation :

$$K = Relab_b \longrightarrow_a (Add_{a,b}(K \oplus \mathbf{b})) \cup \mathbf{a}$$

Examples of bounded clique-width:

An undirected graph is a cograph $\Leftrightarrow$ it has clique-width at most 2.

Distance hereditary graphs have clique-width at most 3.

Examples of unbounded clique-width:

Planar graphs (even of degree 3),

Interval graphs.

Fact : Unlike tree-width, clique-width is sensible to edge directions :

Cliques have clique-width 2 but tournaments have *unbounded clique-width*.

Exercises

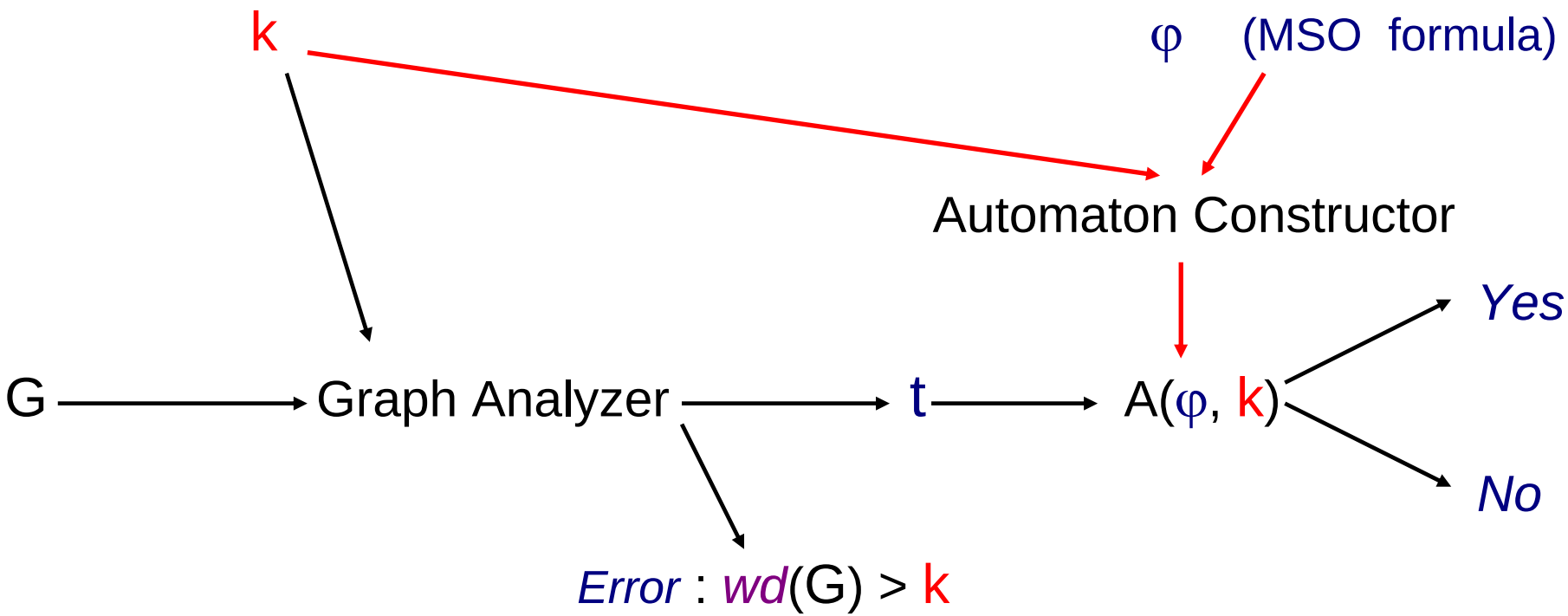
1) Write optimal terms that define grids.

(cwd($G_{n \times n}$) = n+1 ; m+1 $\leq$ cwd($G_{n \times m}$) $\leq$ m+2 if n > m, by Golumbic & Rotics).

2) Give an upper bound to the clique-width of a graph whose biconnected components have clique-width at most k.

3) Give an upper bound to the clique-width of a series-parallel graph.

3. From MSO formulas to automata



Steps ⟶ are done "once for all", independently of G

A(φ,k): finite automaton on terms t

wd = tree-width or clique-width or equivalent,

Tree-decompositions also have algebraic expressions.

Construction of A(φ, k) for "clique-width" terms

k = the number of vertex labels = the bound on clique-width

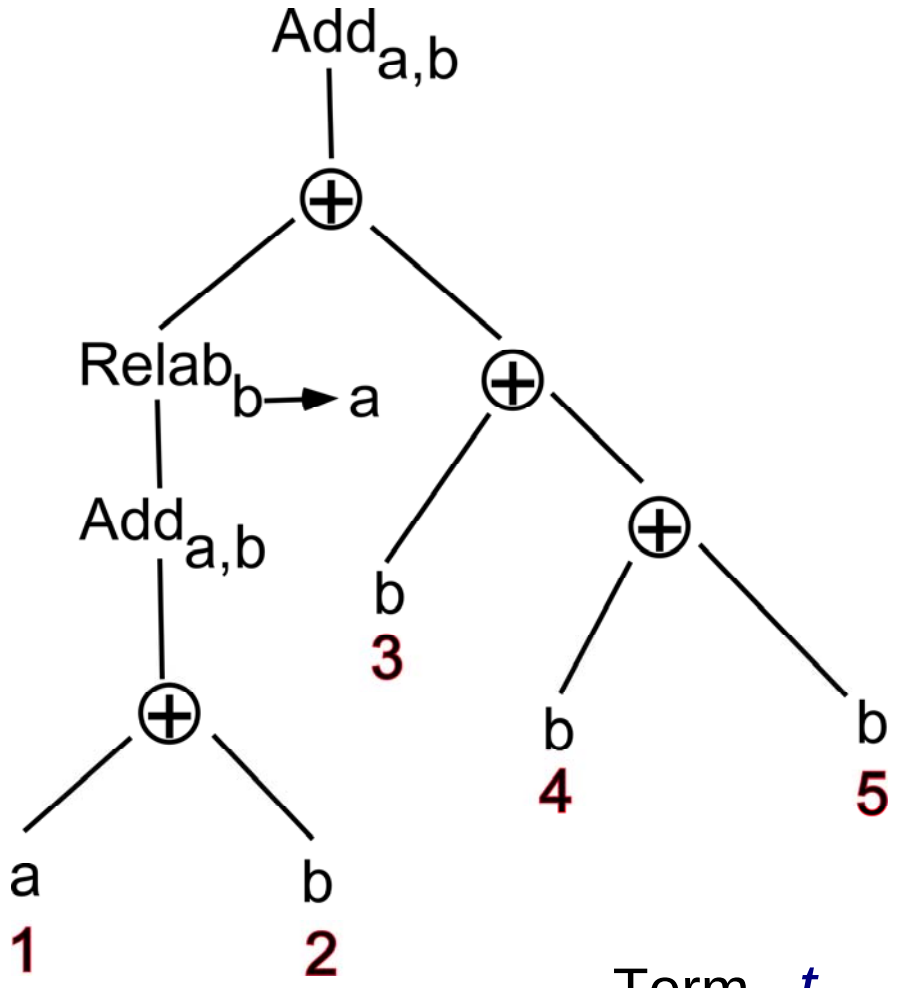
F = the corresponding set of operations and constants :

$\mathbf{a}$, $\varnothing$, $\oplus$, $Add_{a,b}$, $\overrightarrow{Add}_{a,b}$, $Relab_{\,a \longrightarrow b}$

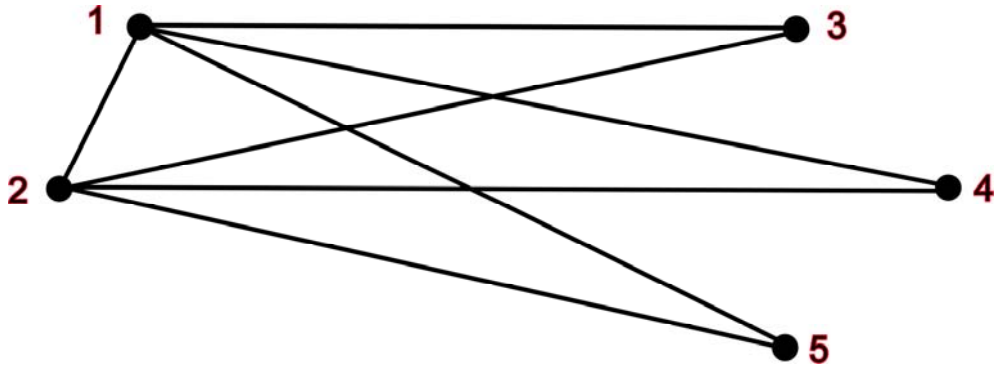
G(*t*) = the graph defined by a term *t* in **T**(F).

Its vertices are (in bijection with) the occurrences of the *constants* (the nullary symbols) in *t* that are not $\varnothing$

Example



Term *t*



Graph G(*t*)

Terms are equipped with Booleans that encode assignments of vertex sets $V_1,\ldots,V_n$ to the free set variables $X_1,\ldots,X_n$ of MSO formulas (*formulas are written without first-order variables*):

1) we replace in F each constant **a** by the constants $(\mathbf{a}, (w_1,\ldots,w_n))$ where $w_i \in \{0,1\}$: we get $F^{(n)}$

(only constants are modified);

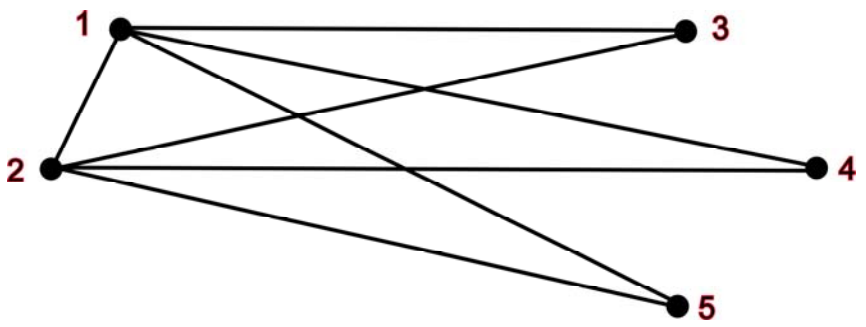
2) a term *s* in $\mathbf{T}(F^{(n)})$ encodes a term *t* in $\mathbf{T}(F)$ and an assignment of sets $V_1,\ldots,V_n$ to the set variables $X_1,\ldots,X_n$:

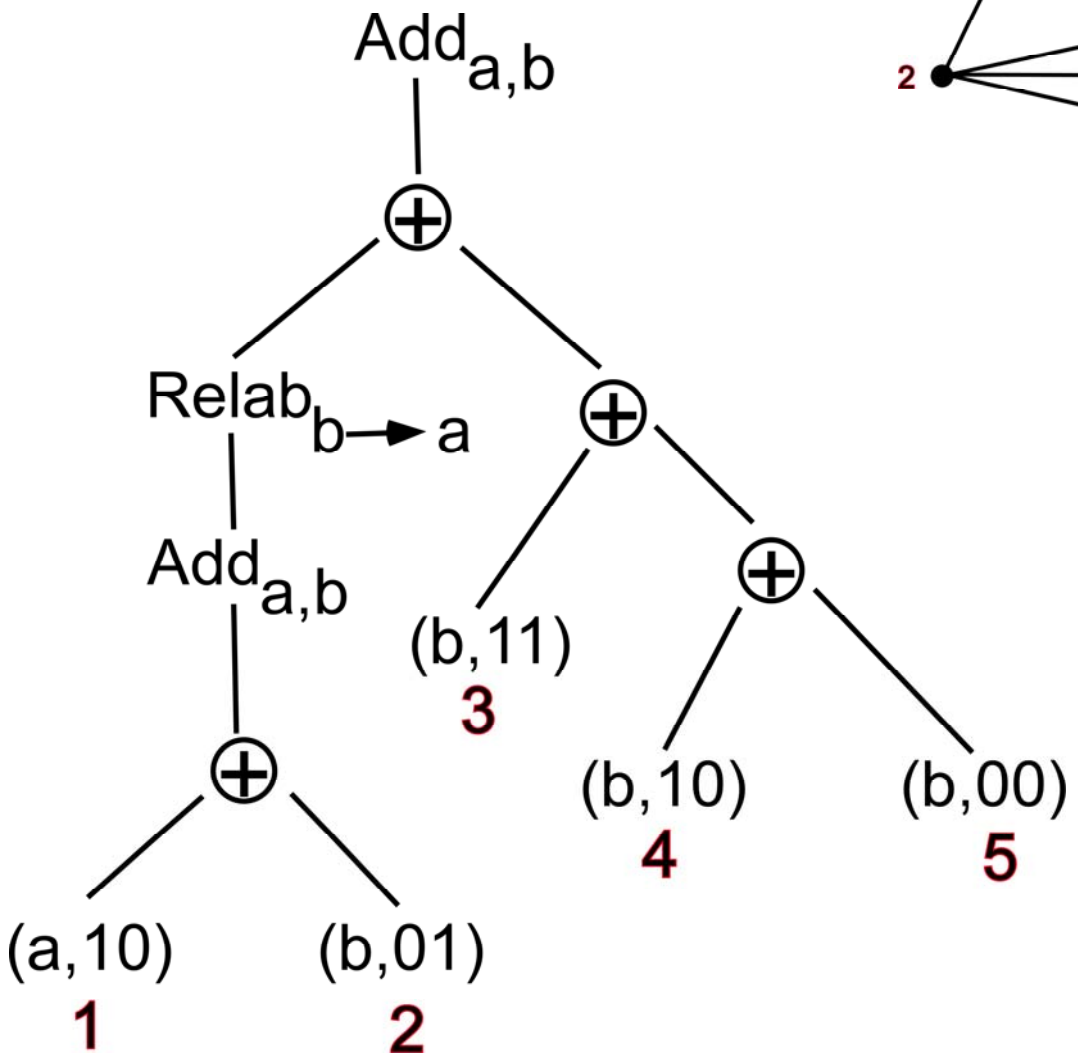
if *u* is an occurrence of $(\mathbf{a}, (w_1,..,w_n))$, then

$w_i = 1$ if and only if $u \in V_i$.

3) *s* is denoted by $t * (V_1,\ldots,V_n)$

Example (continued)





$V_1 = \{1,3,4\}, \; V_2 = \{2,3\}$

Term $t * (V_1,V_2)$

By an induction on φ, we construct for each $\varphi(X_1,\ldots,X_n)$ a finite (bottom-up) deterministic automaton $A(\varphi(X_1,\ldots,X_n), k)$ that recognizes:

$$L(\varphi(X_1,\ldots,X_n)) := \{ t * (V_1,\ldots,V_n) \in \mathbf{T}(F^{(n)}) \ / \ (G(t), V_1,\ldots,V_n) \mid = \varphi \}$$

Theorem : For each sentence φ, the automaton $A(\varphi, k)$ accepts in time $f(\varphi, k).\mid t \mid$ the terms t in $\mathbf{T}(F)$ such that $G(t) \mid = \varphi$

It gives a *fixed-parameter linear* model-checking algorithm for input t, and a *fixed-parameter cubic* one if the graph has to be *parsed*. (The parameter is clique-width; the parsing algorithm is based on results by Oum, Seymour, Hlineny and myself; it uses *rank-width*, equivalent to clique-width for undirected graphs).

The inductive construction of $A(\varphi, k)$

Atomic formulas : discussed below.

For $\wedge$: product of two automata (deterministic or not)

For $\vee$: union of two automata (or product of two *complete* automata; product preserves determinism)

For *negation* : exchange accepting / non-accepting states

for a complete *deterministic* automaton

Quantifications: Formulas are written without $\forall$

$$L(\exists X_{n+1} . \varphi(X_1, ..., X_{n+1})) = pr(L (\varphi(X_1, ..., X_{n+1}))$$

$$A(\exists X_{n+1} . \varphi(X_1, ..., X_{n+1}), k) = pr(A (\varphi(X_1, ..., X_{n+1}), k)$$

where pr is the *projection* that eliminates the last Boolean;

→ a *non-deterministic* automaton.

Some tools for constructing automata

Substitutions and inverse images (*cylindrifications*).

1) If we know $A(\varphi(X_1, X_2), k)$, we get $A(\varphi(X_4, X_3), k)$ because :

$$L(\varphi(X_4, X_3)) = h^{-1}(L(\varphi(X_1, X_2)))$$ where

h maps $(\mathbf{a}, (w_1, w_2, w_3, w_4))$ to $(\mathbf{a}, (w_4, w_3))$. We take

$$A(\varphi(X_4, X_3), k) = h^{-1}(A(\varphi(X_1, X_2)), k)$$

This construction preserves determinism and the number of states.

2) From $A(\varphi(X_1, X_2), k)$, we get $A(\varphi(X_3, \overbrace{X_1 \cup (X_2 \setminus X_4)}^{\textit{Set term}}), k)$ by h^{-1}

with h mapping $(\mathbf{a}, (w_1, w_2, w_3, w_4))$ to $(\mathbf{a}, (w_3, w_1 \vee (w_2 \wedge \neg w_4)))$

Relativization to subsets by inverse images.

If φ is a closed formula expressing a graph property *P*, its relativization $\varphi\,[X_1]$ to X_1 expresses that the subgraph induced on X_1 satisfies *P*. To construct it, we replace recursively

$$\exists\, y.\ \theta \quad \text{by} \quad \exists\, y.\ y \in X_1 \wedge \theta, \ \text{etc…}$$

However, there is an easy transformation of automata :

Let h map (**a** , 0) to $\varnothing$ and (**a** ,1) to **a**.

$$L(\varphi\,[X_1]) = h^{-1}(L(\varphi))$$

Hence:

$$A(\varphi\,[X_1],\ k) := h^{-1}(A(\varphi),\ k)$$

The inductive construction (continued) :

Complete *deterministic* automata for atomic formulas and basic graph properties : automaton over $F^{(n)}$ recognizing the set of terms

$$t * (V_1,\ldots,V_n) \quad \text{in} \quad L(\varphi(X_1,\ldots,X_n))$$

Intuition : in all cases, the *state* reached at node u represents a *finite information* q(u) about the graph G(*t* / u) and the restriction of $V_1,\ldots,V_n$ to the vertices below u (vertices = leaves)

1) if u = f(v,w), we want that q(u) is defined from q(v) and q(w) by a fixed function : → the *transition function* ;

2) whether (G(*t*), $V_1,\ldots,V_n$) satisfies $\varphi(X_1,\ldots,X_n)$ must be checkable from q(*root*) : → the *accepting states*.

Atomic and basic formulas :

$X_1 \subseteq X_2$, $X_1 = \varnothing$, Single(X_1),

Card $_{p,q}$ (X_1) : cardinality of X_1 is = p mod. q,

Card $_{< q}$ (X_1) : cardinality of X_1 is < q.

→ Easy constructions with small numbers of states :

respectively 2, 2, 3, q, q+1.

Example : for $X_1 \subseteq X_2$, the term must have no constant (**a**, 10).

Atomic formula : edg(X_1,X_2) for directed edges

edg(X_1,X_2) means : $X_1 = \{ x \} \wedge X_2 = \{ y \} \wedge x \longrightarrow y$

Vertex labels belong to a set C of k labels.

k^2+k+3 *states* : 0, Ok, a(1), a(2), ab, Error, for a,b in C , $a \neq b$

Meaning of states (at node u in t ; its subterm t/u defines G(t/u) $\subseteq$ G(t)).

0 : $X_1 = \varnothing$, $X_2 = \varnothing$

Ok *Accepting state* : $X_1 = \{v\}$, $X_2 = \{w\}$, edg(v,w) in G(t/u)

a(1) : $X_1 = \{v\}$, $X_2 = \varnothing$, v has label a in G(t/u)

a(2) : $X_1 = \varnothing$, $X_2 = \{w\}$, w has label a in G(t/u)

ab : $X_1 = \{v\}$, $X_2 = \{w\}$, v has label a, w has label b (hence $v \neq w$)
and $\neg$edg(v,w) in G(t/u)

Error : all other cases

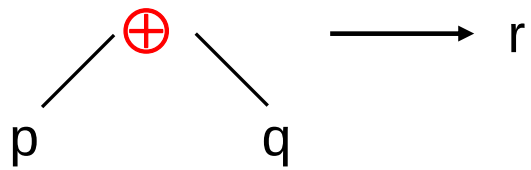
Transition rules

For the constants based on **a** :

(**a**,00) → 0 ; (**a**,10) → a(1) ; (**a**,01) → a(2) ; (**a**,11) → Error

For the binary operation ⊕:

(p,q,r are states)



If p = 0 then r := q

If q = 0 then r := p

If p = a(1), q = b(2) and a ≠ b then r := ab

If p = b(2), q = a(1) and a ≠ b then r := ab

Otherwise r : = Error

For unary operations $\overrightarrow{Add}_{a,b}$ $\overrightarrow{Add}_{a,b}$ ⟶ r

p

If p = ab then r := Ok else r : = p

For unary operations $Relab_{a \longrightarrow b}$

If p = a(i) where i = 1 or 2 then r : = b(i)

If p = ac where c ≠ a and c ≠ b then r : = bc

If p = ca where c ≠ a and c ≠ b then r : = cb

If p = Error or 0 or Ok or c(i) or cd or dc where c ≠ a

then r : = p

4. Practical difficulties and (some) remedies.

Parsing :

Case of clique-width:

Checking if a graph has clique-width $\leq k$ is NP-complete (with k in the input ; Fellows *et al.*)

The *cubic approximate* parsing algorithm (by Oum *et al.*) based on *rank-width* is not (directly) implementable.

Remark: For certain classes of graphs of bounded clique-width defined by forbidden induced subgraphs, optimal clique-width terms can be constructed in polynomial time, by using in many cases, *modular decomposition*.

Case of tree-width: *(bounded tree-width implies bounded clique-width)*

Checking if a graph has tree-width $\leq$ k is NP-complete (with k in the input ; Arnborg *et al.*)

The *linear-time exact* parsing algorithm by Bodlaender (for tree-width) takes time $O(2^{32.k.k.k} . n)$. There are usable algorithms for (non-random) graphs with 50 vertices and tree-width $\leq$ 35 (Bodlaender & Koster).

Specific algorithms : Flow-graphs of structured programs have tree-width at most 6 and tree-decompositions are easy to get from the parse trees of programs (Thorup).

Sizes of automata :

The number of states of $A(\varphi, k)$ is bounded by an h-iterated exponential where h is the number of quantifier alternations of φ .

There is no alternative construction giving a fixed bound on nestings of exponentiations (Meyer & Stockmeyer, Weyer, Frick & Grohe).

The construction by induction on the structure of φ may need intermediate automata of huge size, even if the *unique minimal deterministic* automaton equivalent to $A(\varphi, k)$ has a manageable number of states.

Soguet *et al.* using MONA have constructed automata for the following cases ; no success for clique-width 4 :

	Clique-width 2	*Clique-width 3*
MaxDegree≤3	91 states	Space-Out
Connected	11 states	Space-Out
IsConnComp(X)	48 states	Space-Out
Has≤4-VertCov	111 states	1037 states
HasClique ≥ 4	21 states	153 states
2-colorable	11 states	57 states

One can avoid the inductive construction and construct "directly" deterministic automata for basic properties : NoEdge, Connected, NoCycle

Property	Partition $(X_1,\ldots,X_p)$	edg(X,Y)	NoEdge	Connected, NoCycle for degree $\leq p$	Path(X,Y)	Connected, Nocycle
Number of states N(k)	2	k^2+k+3	2^k	$2^{O(p.k.k)}$	$2^{O(k.k)}$	$2^{2^{O(k)}}$

Examples of automata too large to be constructed, i.e., "compiled":

for k = 2 : 4-colorability, 3-acyclic-colorability, NoCycle (i.e., is a forest)

for k = 4 : connectedness, for k = 5 : 3-colorability, clique.

The *minimal deterministic* automaton for Conn(X) has more than $2^{2^{k/2}}$ states.

An issue : *Fly-automata*

States and transitions are not listed in huge tables :

they are *specified* (in uniform ways for all k) by "small" programs.

Can be nondeterministic.

Closure properties:

- Union (for $\vee$)
- Product (for $\wedge$)
- Image (for $\exists$)
- Inverse image (for substitutions and relativization)
- Determinization.

Example of a state for connectedness :

q = { {a}, {a,b}, {b,c,d}, {b,d,f} },

a,b,c,d,f are vertex labels; q is the set of *types* of the connected components of the current graph. (type(H) = set of labels of its vertices)

Some transitions :

$Add_{a,c}$: q ⟶ { {a,b,c,d}, {b,d,f } },

$Relab_{a \to b}$: q ⟶ { {b}, {b,c,d}, {b,d,f } }

Transitions for ⊕ : union of sets of types.

Using fly automata works for formulas with no quantifier alternation that use “new” atomic formulas for “basic” properties

Examples : p-*acyclic colorability*

$\exists X_1,\ldots,X_p$ (Partition($X_1,\ldots,X_p$) $\wedge$ NoEdge(X_1) $\wedge$ ..... $\wedge$ NoEdge(X_p) $\wedge$ ...
..... $\wedge$ NoCycle($X_i \cup X_j$) $\wedge$ ...)

(all $i < j$; set terms $X_i \cup X_j$ avoid some quantifications).

Minor inclusion : *H* simple, loop-free. *Vertices(H)* = { $v_1,\ldots,v_p$ }

$\exists X_1,\ldots,X_p$ (Disjoint($X_1,\ldots,X_p$) $\wedge$ Conn(X_1) $\wedge$ ... $\wedge$ Conn(X_p) $\wedge$ ...
... $\wedge$ Link(X_i , X_j) $\wedge$ ...)

Existence of “holes” : odd induced cycles (to check *perfectness* ; one checks “anti-holes” on the edge-complement of the given graph).

Some experiments, by Irène Durand.

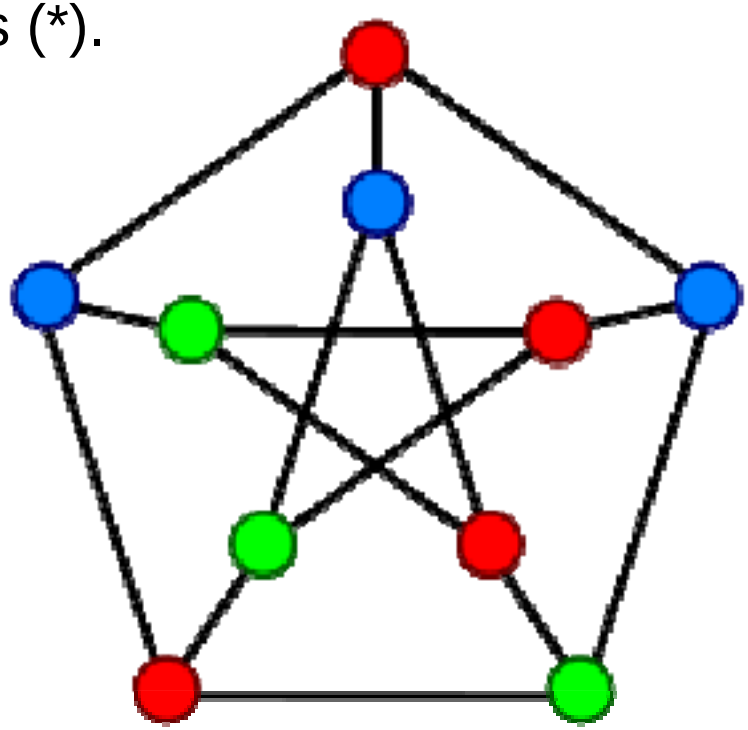
3-colorability of the 6 x 7 grid (of clique-width 8) in 7 minutes,
of the 6 x 33 grid (of clique-width 8) in 90 minutes.

3-colorability of the Petersen graph (clique-width 7) in 1.5 second,
its 4-acyclic-colorability in 4 minutes (*).

(3-colorable but not acyclically;

red and **green** vertices

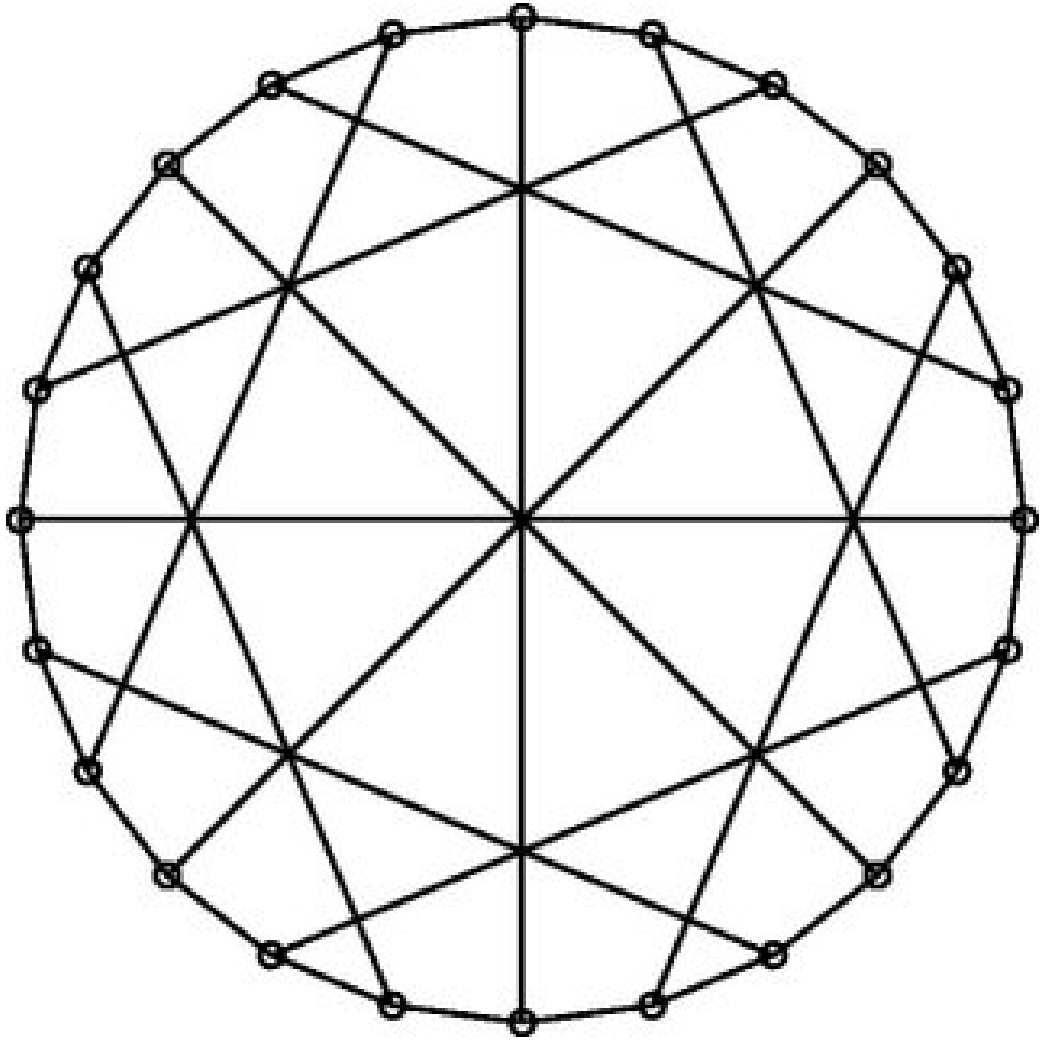
induce a cycle).



(*) For a term with *annotations*.

The McGee graph

24 vertices,
36 edges,
clique-width ≤ 10.



3-colorability in 7 minutes,

3-AC-colorability in 12 hours.

Annotations : an additional tool

At some positions in the given term, we attached some (finite) contextual information.

Example :

At position u in t, we attach the set

$ADD_t(u)$ = the set of pairs (a,b) such that some operation

$Add_{c,d}$ *above* u (hence, in its "context") adds edges between the (eventual) vertices *below* u labelled by a and b.

The sets $ADD_t(u)$ can be computed in *linear time* by means of a top-down traversal of t.

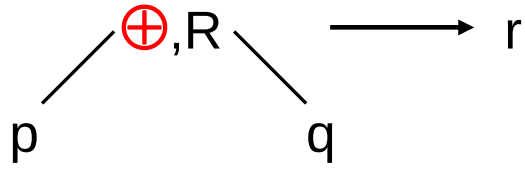
Motivation: Certain automata on annotated terms may have less states.

Example : *edg*(X_1, X_2) : 2k+3 states instead of k^2 +k +3 (cf. page 34):

0, Ok, a(1), a(2), Error, for a in C.

Transitions for $\oplus$ annotated by R :

(p, q, r are states)



If p = 0 then r := q ; if q = 0 then r := p ;

if p = a(1), q = b(2) and (a , b) $\in$ R $\wedge$ then r := Ok ;

and if (a , b) $\notin$ R $\wedge$ then r := Error ;

if p = b(2), q = a(1) : *idem* ;

otherwise r : = Error.

Other examples :

For *Clique*(X) meaning that X induces a clique :

$2^k + 2$ states instead of $2^{O(k.k)}$.

For *Connectedness* : same states but they "shrink" quicker :

cf. the rules for $Add_{a,c}$ on page 43

Has been tested successfully

5. Edge set quantifications and tree-width

Incidence graph of G undirected, **Inc**(G) = ($V_G \cup E_G$, $inc_G(.,.)$.)

$inc_G(v,e) \Leftrightarrow$ v is a vertex of edge e.

Monadic second-order formulas written with inc can use quantifications on sets of edges :

they define MSO_2 –*expressible graph properties*.

The existence of a perfect matching or a Hamiltonian circuit is MSO_2 -expressible but not MSO-expressible.

Exercises

1) Write an MSO_2-sentence expressing that a graph has a Hamiltonian cycle.

2) Write an MSO_2-sentence expressing that a graph has a spanning tree of degree ≤ 3. (This property is not MSO-expressible).

MSO_2 model-checking

- For an FPT algorithm, the parameter is tree-width and cannot be clique-width. By Kreutzer, Makowsky *et al.* MSO_2 model-checking *needs* restriction to bounded tree-width unless P=NP, ETH, Exptime=NExptime.

- MSO_2- model-checking *reduces to* MSO- model-checking:

  Given G (can have mult. edges) we build a tree-decomposition of G, then a tree-decomposition of its Inc(G) of same width k (easy), then a "clique-width" term for Inc(G) of width $\leq 2^{O(k)}$ (exponential blow-up, not avoidable).

  An MSO_2 property is MSO on Inc(G): hence, we can apply the previous algorithm.

  Problem : the exponential clique-width of Inc(G)

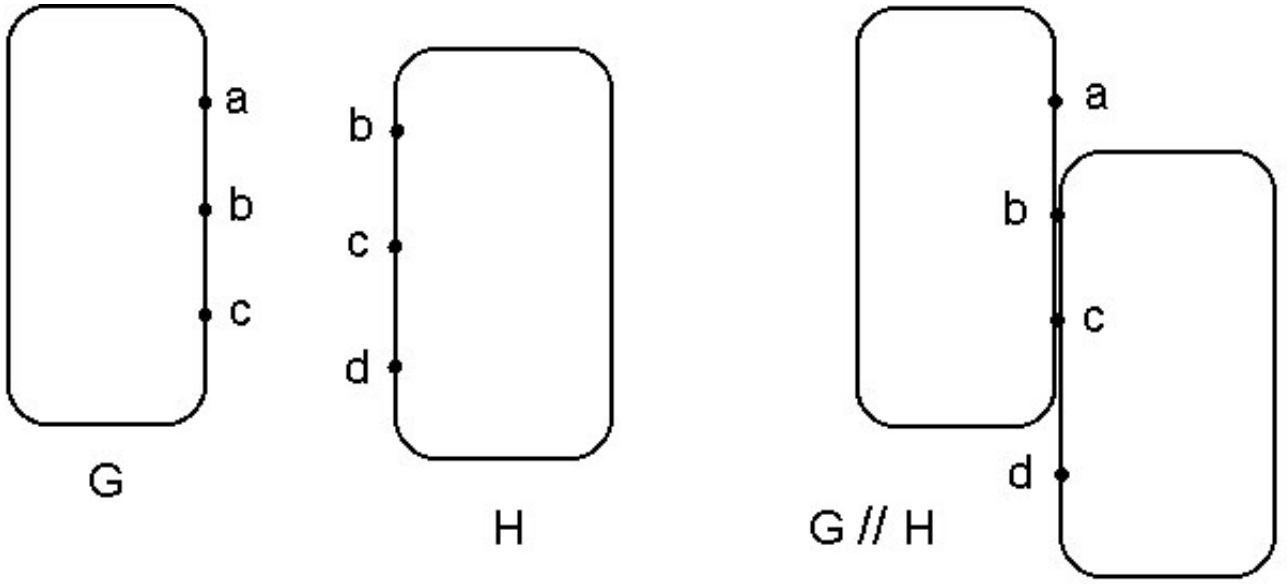
Graph operations that characterize tree-width

Graphs have distinguished vertices called *sources,* (or *terminals* or *boundary vertices*) pointed to by source labels from a finite set : {*a, b, c, ..., d*}.

Binary operation(s) : *Parallel composition*

G // H is the disjoint union of G and H and sources with same label are fused.

(If G and H are not disjoint, we use a copy of H disjoint from G).



Unary operations :

Forget a source label

Forget$_a$(G) is G without *a*-source: the source is no longer distinguished (it is made "internal").

Source renaming :

Ren$_{a \leftrightarrow b}$(G) exchanges source labels *a* and *b* (replaces *a* by *b* if *b* is not the label of any source)

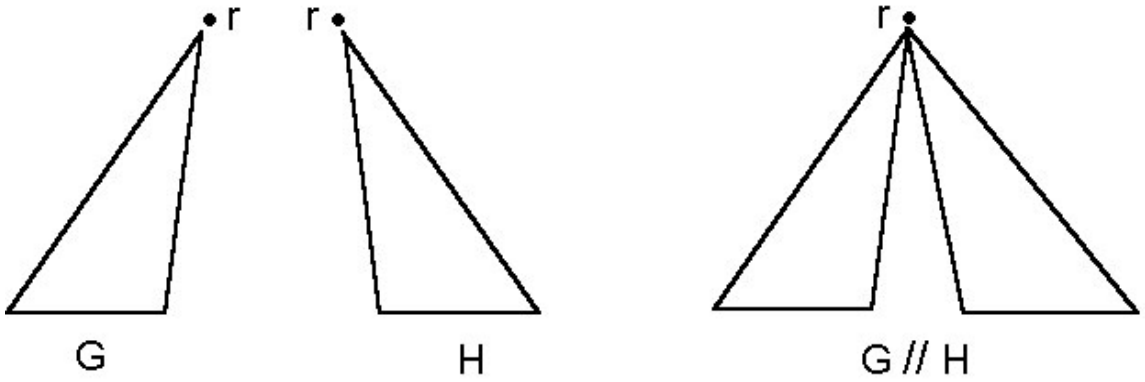
Nullary operations denote *basic graphs* : 1-edge graphs, isolated vertices.

Terms over these operations *define* (or *denote*) graphs (with or without sources). They can have parallel edges.

Example : Trees

Constructed with two source labels, *r* (root) and *n* (new root).

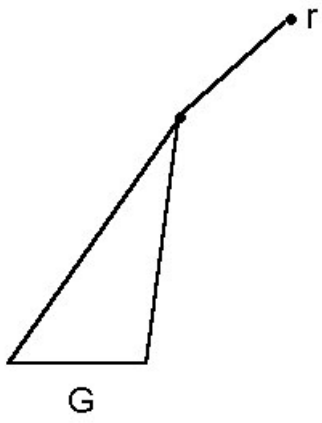
Fusion of two trees at their roots :



Extension of a tree by parallel composition with a new edge, forgetting the old root, making the "new root" as current root :

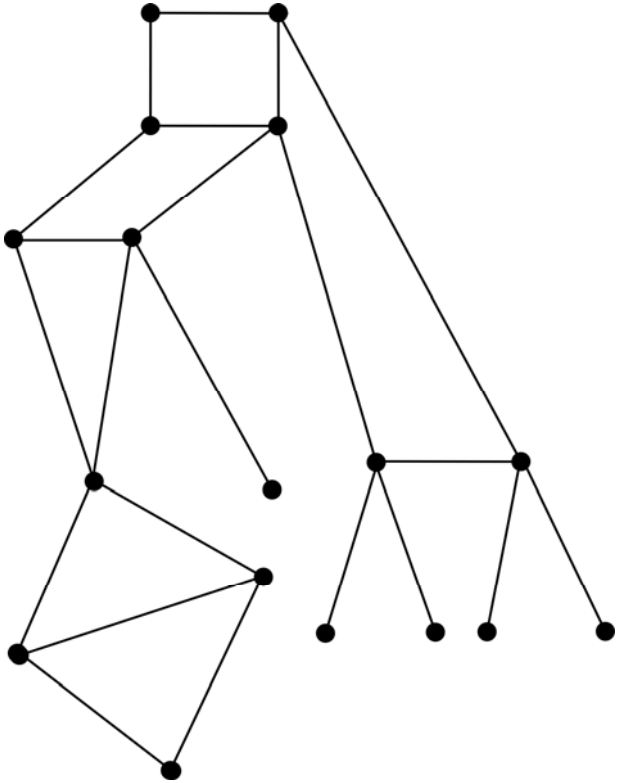
e = ***r*** ●——————● ***n***

$Ren_{n \leftrightarrow r}\,(Forget_r\,(G \,//\, e\,))$

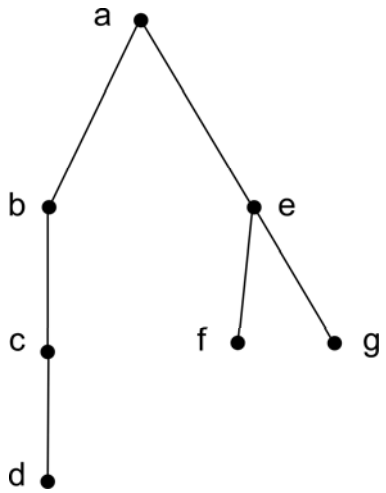


Trees are defined by : T = T // T ∪ extension(T) ∪ ***r***

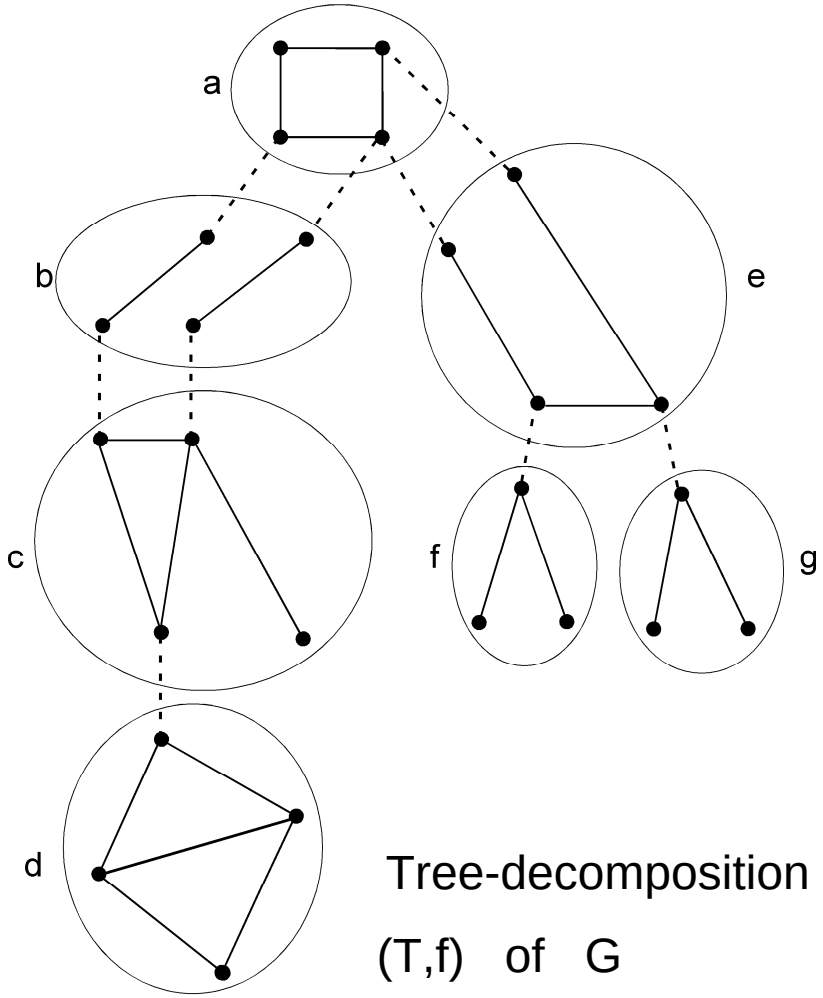
Relation to tree-decompositions and tree-width



Graph G



Tree T



Tree-decomposition (T,f) of G

Dotted lines - - - - link *copies* of a same vertex.

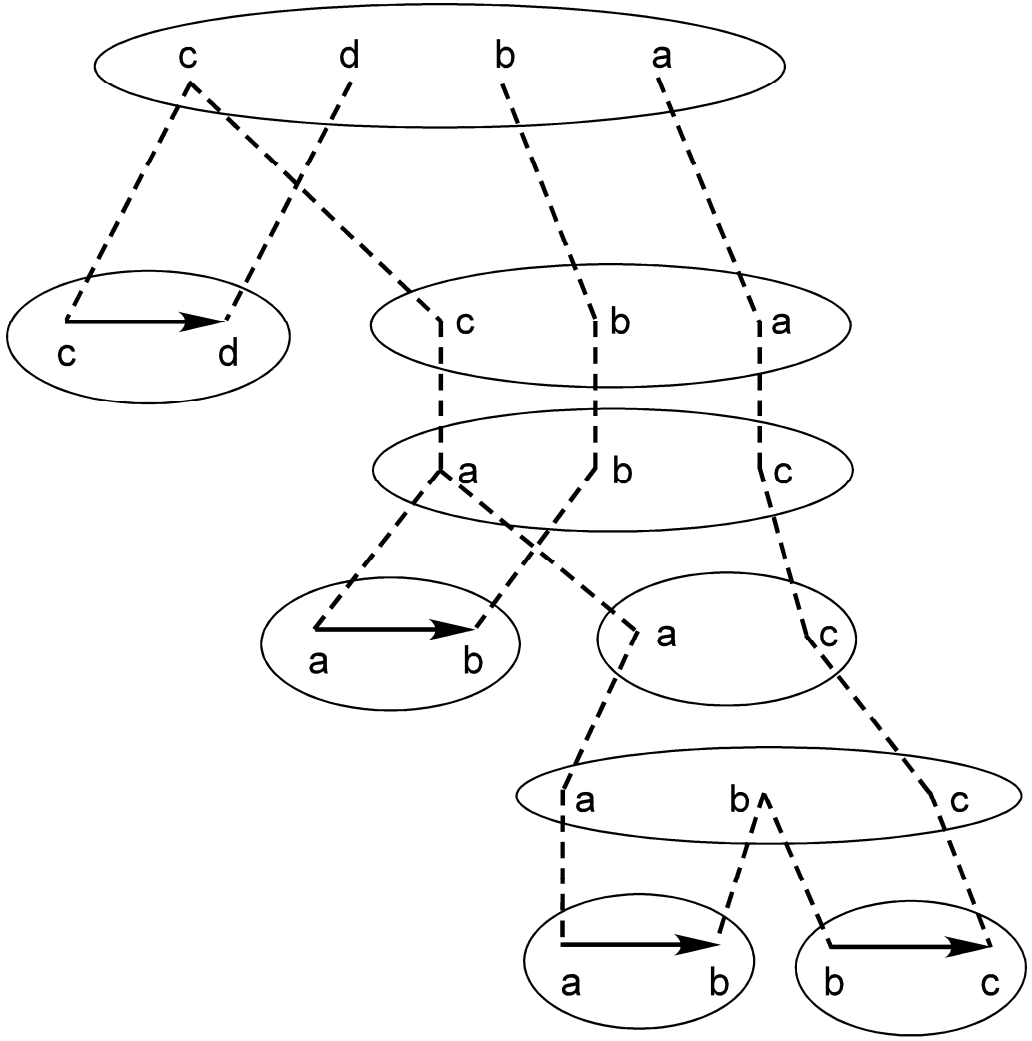
Width = max. size of a box -1. Tree-width = min. width of a tree-dec.

Proposition: A graph has tree-width $\leq k$ $\Leftrightarrow$ it can be constructed from edges by using the operations $//$, $Ren_{a \leftrightarrow b}$ and $Forget_a$ with $\leq k+1$ labels a,b,….

Proposition : Bounded tree-width implies bounded clique-width (cwd(G) $\leq 2^{2twd(G)+1}$ for G directed), but not *conversely*.

From an algebraic expression to a tree-decomposition

Example : cd // $Ren_{a \leftrightarrow c}$ (ab // $Forget_b$(ab // bc)) (ab denotes an edge from a to b)



The tree-decomposition associated with this term.

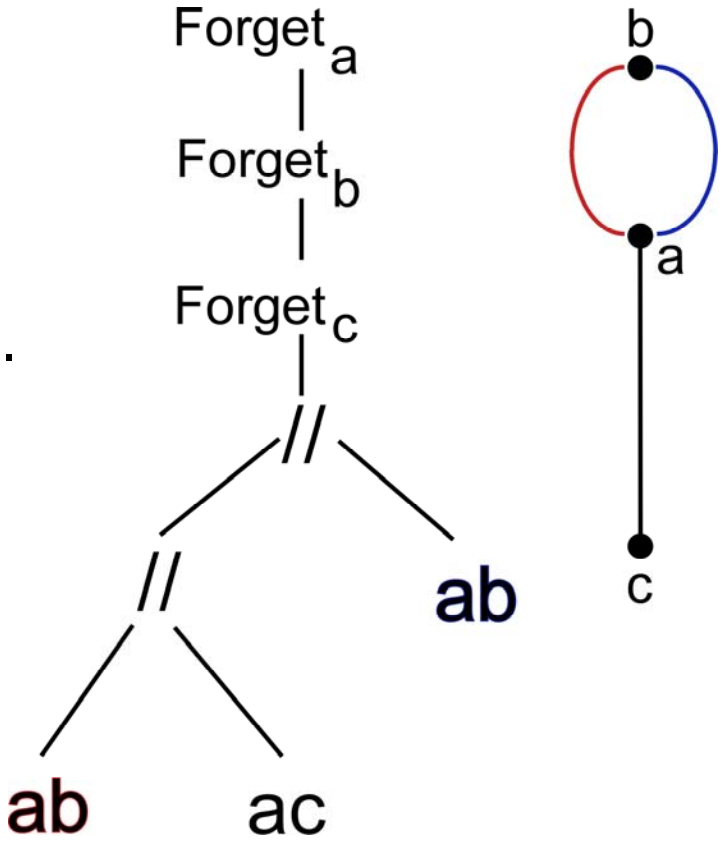
Automata for the model-checking of MSO_2 formulas

To extend the method used for bounded clique-width, we need a representation of vertices *and edges* by occurrences of operations and constants in the terms representing tree-decompositions.

Undirected graphs of tree-width $\leq$ k-1 are denoted by terms over the operations : $//$, $Forget_a$ and the constants a, ab for a,b $\in$ [k]={1,…,k}, *without renamings of labels*.

First method

Vertices are represented at the occurrences of *Forget* operations. The edges are at the leaves of the tree, *below* the nodes representing their ends.



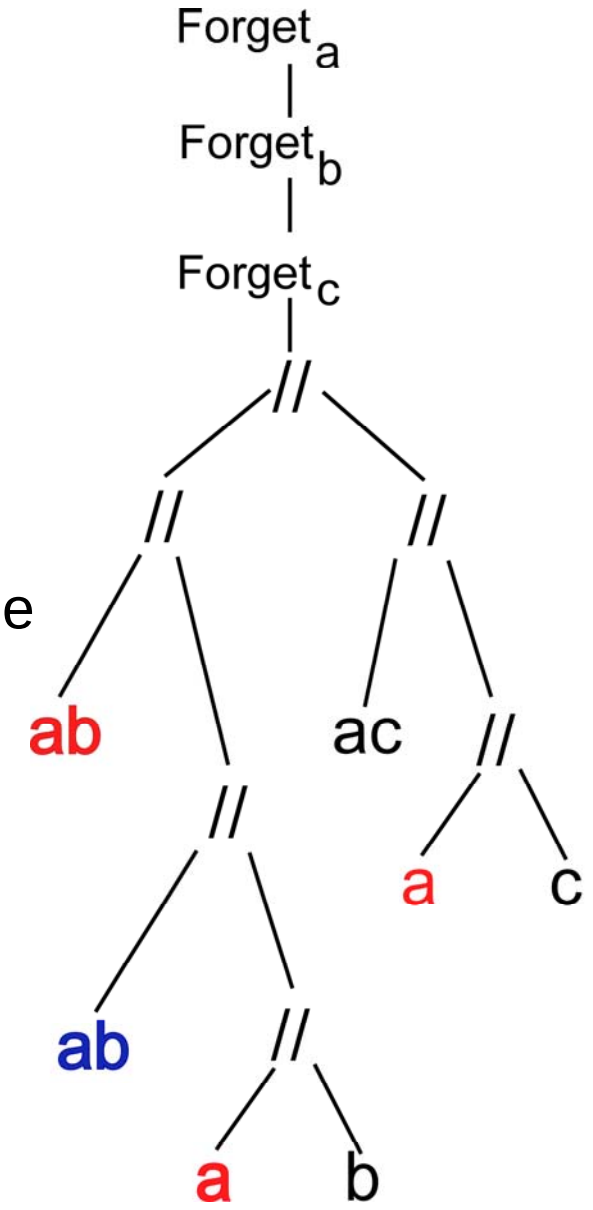
The automaton for *edg(X,Y)* has $2^{\Theta(k.k)}$ states (compare with $O(k^2)$, cf. page 34

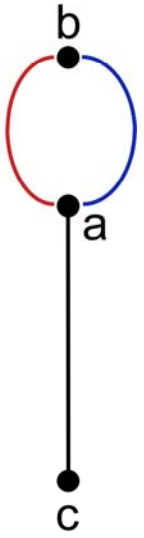
Second method

Vertices are represented at the leaves, the edges are at nodes *close to* those representing their ends.

Because of **//** which fuses some vertices, each vertex is represented by several leaves.

On the figure, vertex a is represented by two leaves.





Equality of vertices is an equivalence relation $\simeq$ on leaves.

Hence: there exists a set of vertices X such that …

is expressed by:

there exists a set of leaves X, saturated for $\simeq$ such that …

Same exponential blow up as with the first possibility.

Note : The responsible is //.

Improving the first method by annotations

Recall: the vertices are in bijection with the occurrences of the *Forget* operations.

The annotation : at each occurrence u of $Forget_a$ representing a vertex x is attached the set of labels b such that the first occurrence of $Forget_b$ above u represents a vertex adjacent to x.

The automaton for *edg*(X,Y) has $2^{2k}+2$ states (instead of $2^{\Theta(k.k)}$).

The automaton for *in*(X,Y) has $k(k-1)/2 + 3$ states.

Incidence and adjacency are handled separately on "redundant" representations of graphs by terms (*edg* by using the annotation).

Conclusion

1. Using automata for model-checking of MS sentences on graphs of bounded tree-width or clique-width is not hopeless if we use fly-automata, built from (possibly non-deterministic) "small" automata for basic graph properties (and their negations), and for sentences without quantifier alternation.

2. More tests on significant examples are necessary, and also comparison (theory and practice) with other approaches : games, monadic Datalog, specific problems, "Boolean width".

3. One can adapt fly-automata to counting and optimization problems ? However, this extension should be tested.