Special tree-width and the verification of monadic second-order graph properties with edge quantifications

Bruno Courcelle

Institut Universitaire de France & Université Bordeaux 1, LaBRI

References : Graph structure and monadic second-order logic, book to be published by *Cambridge University Press*, see : http://www.labri.fr/perso/courcell/ActSci.html

On the model-checking of monadic second-order formulas with edge set quantifications. *Discrete Applied Maths*, to appear

Main topics of the lecture

Fixed-parameter tractable model-checking algorithms for monadic second-order (MS) sentences on graphs with respect to clique-width and tree-width.

Review of the method and introduction of *fly-automata*.

Introduction of *special tree-width*, a variant of tree-width, motivated by the case of MS sentences using edge quantifications.

Two ways of considering graphs

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

an element of an algebra of graphs

(Similar to words, elements of monoids)

2) A graph is a *logical structure* ;

graph properties can be expressed by logical formulas

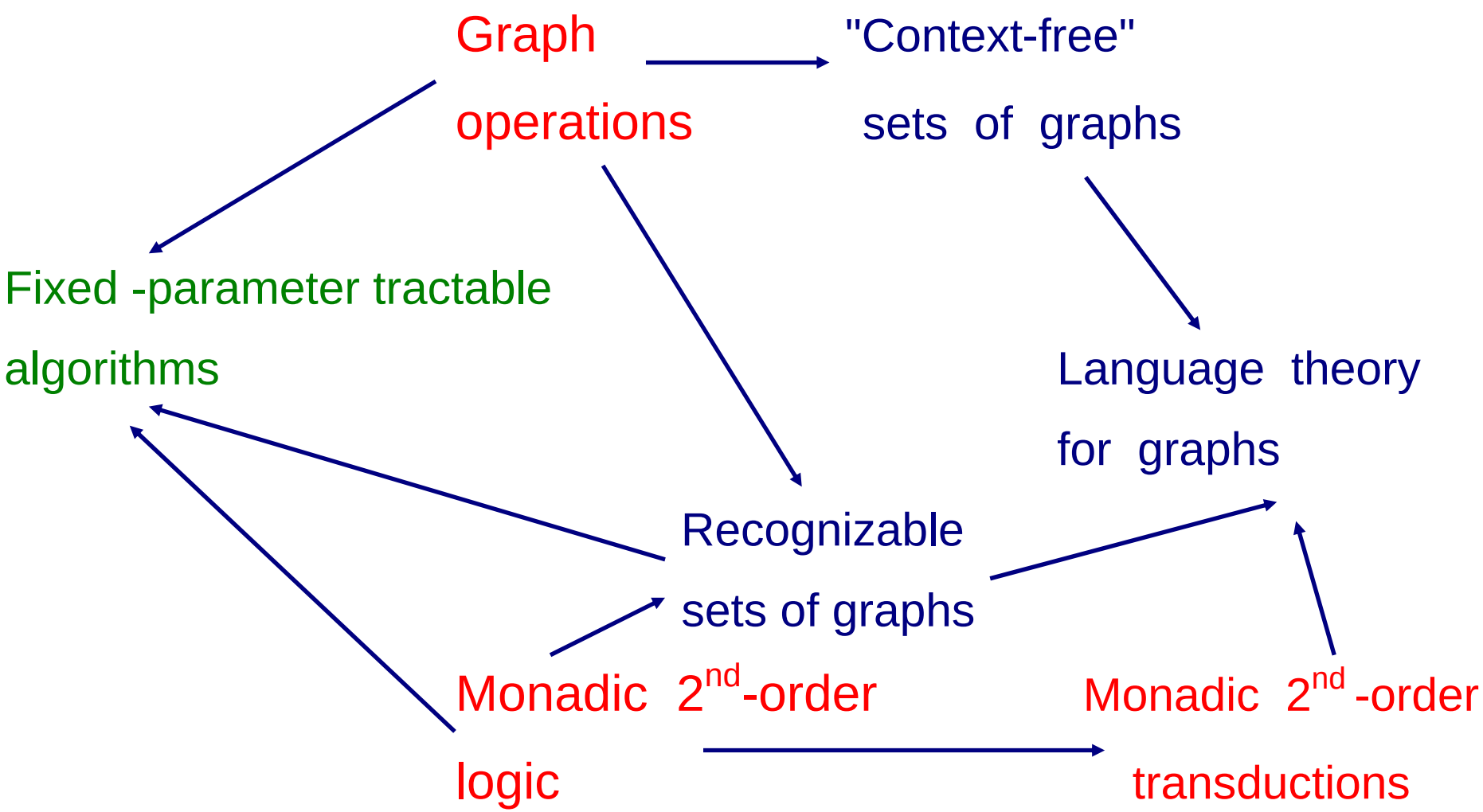
(FO = first-order, MS = monadic second-order, SO = second-order)

Consequences:

a) *Language Theory* concepts extend to graphs

b) *Algorithmic meta-theorems*

An overview chart



Some algorithmic meta-theorems

Language	*Graphs*	*Complexity class*
FO	All	P
FO	Bounded expansion	Linear (2010)
$\exists$SO	All	NP
MS_2 (edge quantif.)	Bounded tree-width	Linear
	Bounded tree-width	LogSpace (2010)
MS	Bounded clique-width	Cubic

Other meta-theorems based on MS logic :

Labelling schemes (or *indexing*), *Enumeration*

Kernelization (for FPT algorithms)

Summary of lecture

1) Monadic second-order (MS) model checking;

   Expressive power of monadic second-order logic

2) Two graph algebras, tree-width and clique-width

3) Special tree-width (*new*)

4) Automata constructed from MS formulas

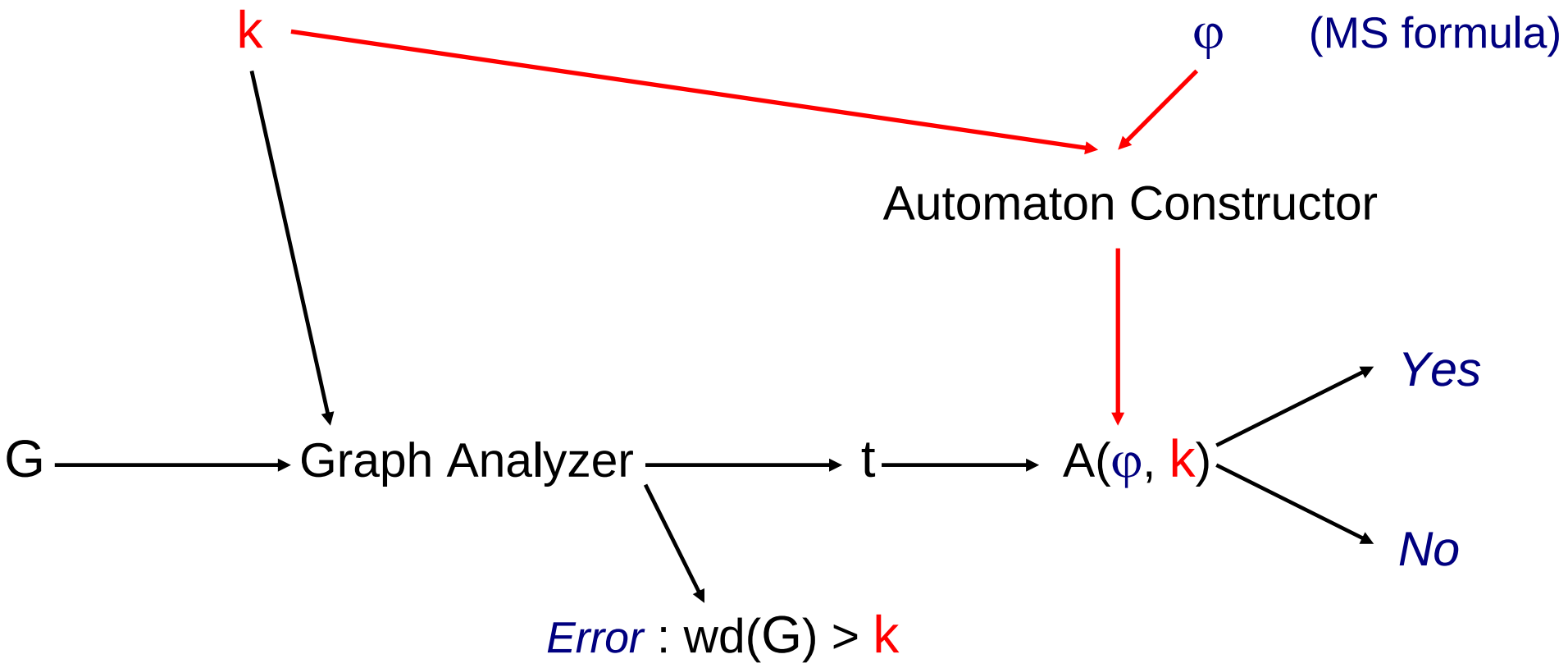
   Case of clique-width

   Case of special tree-width

   Difficulties with tree-width

5) Experiments with *fly*-automata (joint work with Irène Durand)

1. MS model-checking : the general scheme



Steps ⟶ done "once for all", independent of G

A(φ,k): finite automaton on terms (wd = tree-width or clique-width or equivalent)

FPT model-checking algorithms

MS formulas

MS_2 formulas

using edge quantifications

$G = (V_G , edg_G(.,.).)$

$Inc(G) = (V_G \cup E_G, inc_G(.,.))$

for G undirected : $inc_G(e,v) \Leftrightarrow$

v is a vertex (in V_G) of edge e (in E_G)

FPT for clique-width

FPT for tree-width

Expressive powers of logical languages

(Typical examples)

FO : maximal degree = 4 , diameter $\leq$ 6 , outdegree $\leq$ 3.

MS properties that are *not FO* : *3-colorablility*

$\exists X,Y$ (”X,Y are disjoint” $\wedge$ $\forall u,v$ { edg(u,v) $\Rightarrow$

[$(u \in X \Rightarrow v \notin X) \wedge (u \in Y \Rightarrow v \notin Y) \wedge (u \notin X \cup Y \Rightarrow v \in X \cup Y$] })

Connectedness, negation of :

$\exists X$ ($\exists x \in X \wedge \exists y \notin X \wedge \forall u,v\,(u \in X \wedge$ edg(u,v) $\Rightarrow v \in X)$)

Planarity (via two forbidden minors K_5 and $K_{3,3}$)

Perfectness (via forbidden holes and anti-holes)

For a word or a term, membership in a fixed *regular language*

(FO property in certain cases)

Expressive powers of logical languages (continued)

MS_2 property that is *not MS* : has a *perfect matching* or a *Hamiltonian circuit* or a *spanning tree of degree* ≤ 3

SO property that is *not* MS_2 : has a *nontrivial automorphism*

For a word, is = $a^n b^n$ for some n *(nonregular language).*

2. Graph algebras and widths of graphs

Two (*not one*) and only two *robust* (in a precise sense) graph algebras:

the "HR" algebra → algebraic characterization of tree-width,
the "VR" algebra → definition of clique-width.

Note : "HR" refers to the "Hyperedge-Replacement (context-free) graph grammars"; they generate the equational sets of the "HR" algebra;

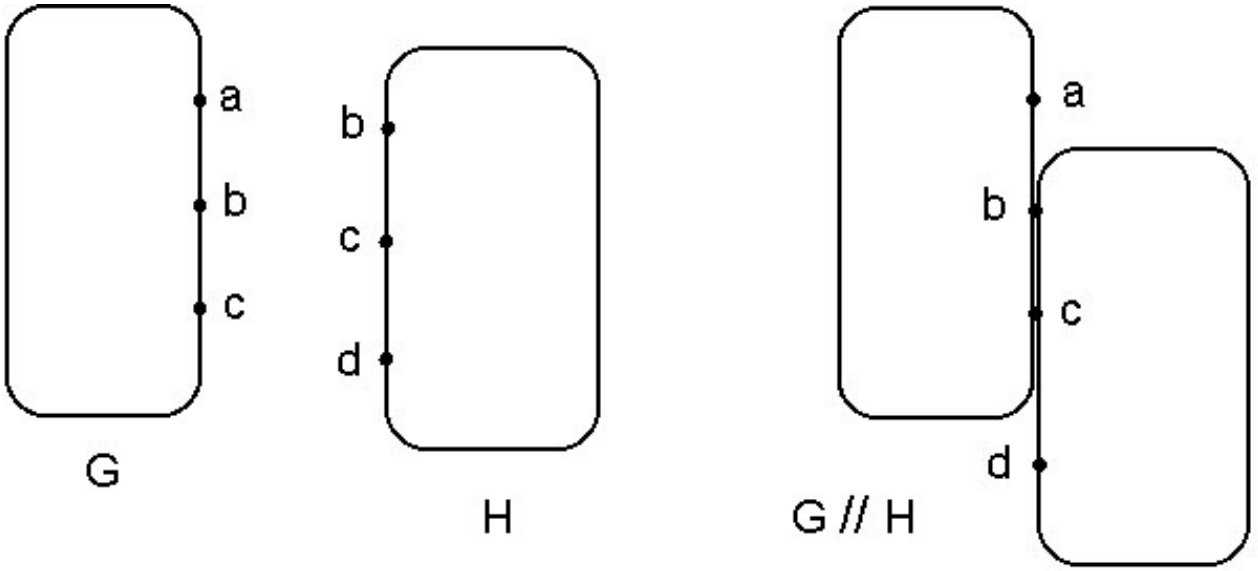
"VR" refers to the "context-free Vertex-Replacement graph grammars"; they generate the equational sets of the "VR" algebra

The "HR" algebra and tree-width

Graphs *with multiple edges*, equipped with distinguished vertices called *sources* (or *boundary vertices* or *terminals*) pointed to by *source labels* from finite sets {*a, b, ..., d*}.

Binary operation : *Parallel composition*

G **//** H is the disjoint union of G and H, where sources with same name are fused (If G and H are not disjoint, one takes 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 a distinguished vertex : it is made "internal".

Source renaming :

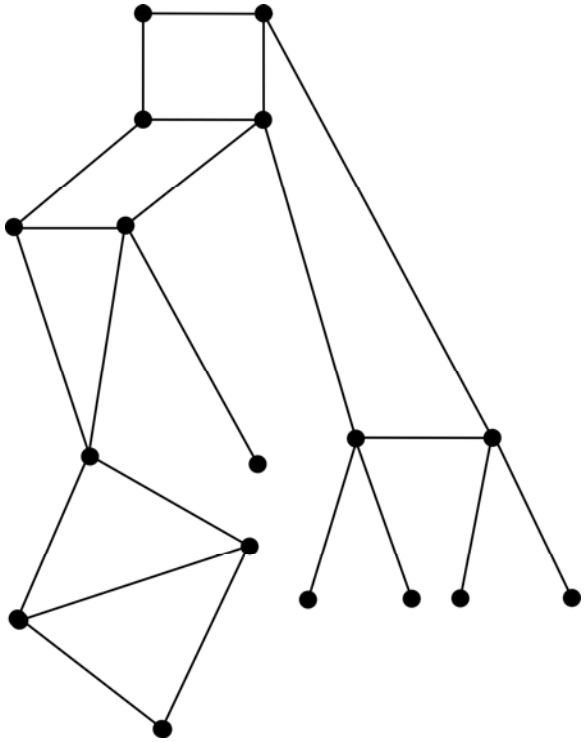
Ren$_{a \leftrightarrow b}$(G) exchanges source labels a and b

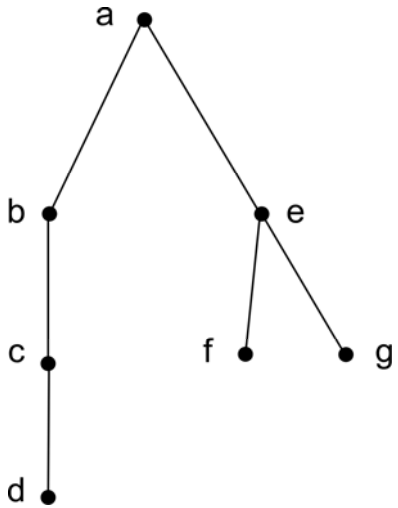
(replaces a by b if b is not the label of a source)

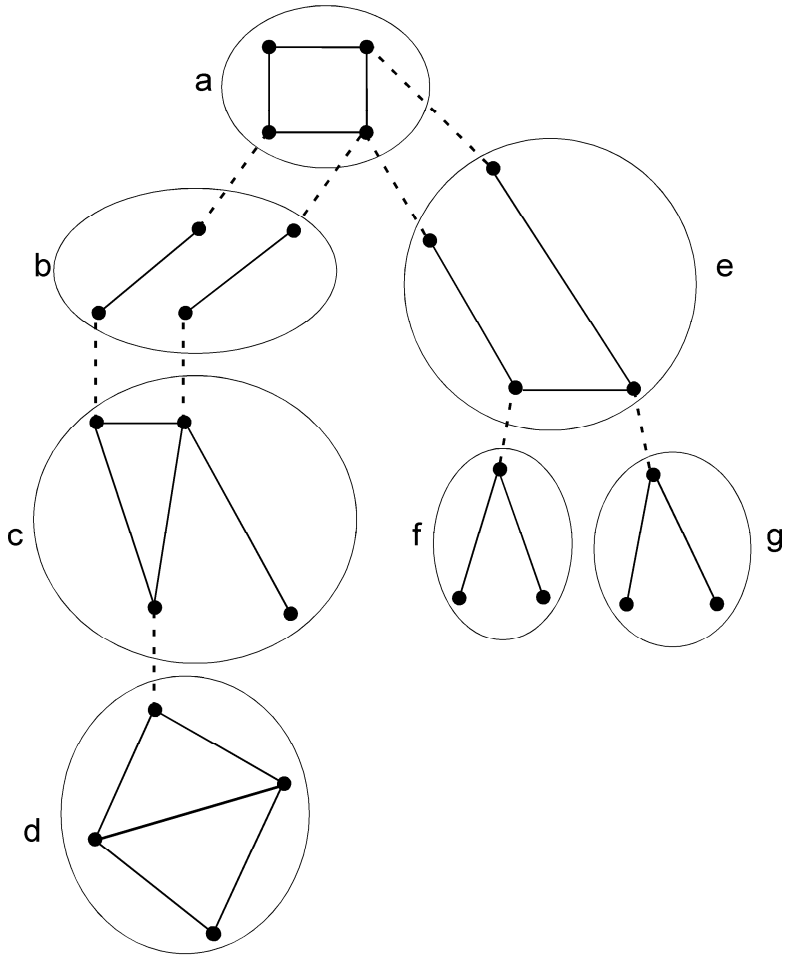
Nullary operations denote the most *elementary graphs* :

the connected graphs with at most one edge.

Tree-decompositions







a decomposition of width 3

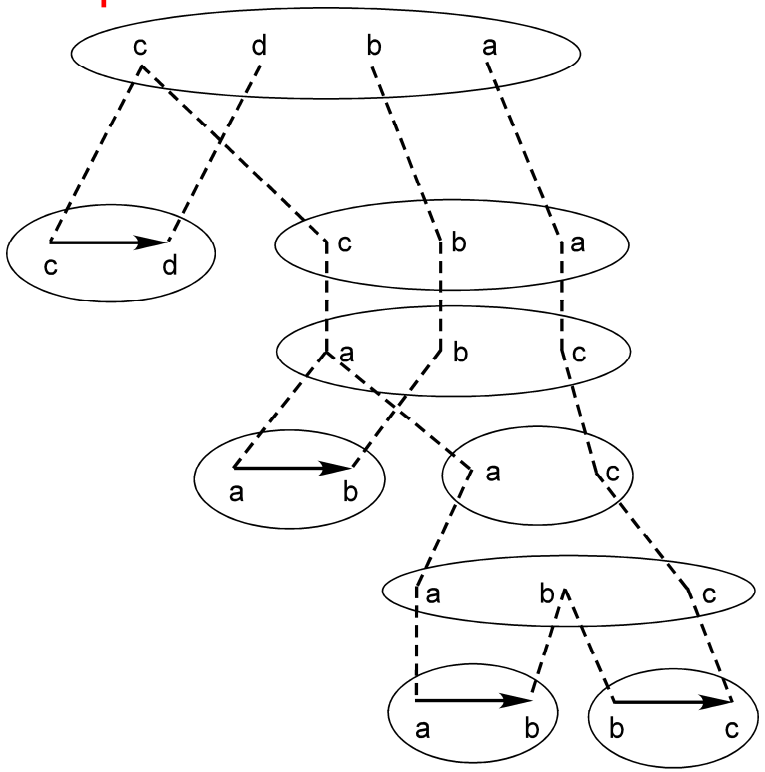
Proposition: A graph has tree-width ≤ k ⇔ it can be constructed from basic graphs with ≤ k+1 labels by using the operations // , $Ren_{a \leftrightarrow b}$ and $Forget_a$

From an algebraic expression to a tree-decomposition

Example : cd // $Ren_{a \leftrightarrow c}$ (ab // $Forget_b$(ab // bc))

(the constant ab denotes an edge from source a to source b)

The tree-decomposition associated with this term



The "VR" algebra and clique-width.

Clique-width was originally defined for *simple* graphs, but we extend the definitions to graphs with *multiple edges*.

Graphs are loop-free (just to simplify notation).

They have vertex labels : *a,b,c ,...* Each vertex has a single label, and each label designates a *set of vertices* (not a unique one as in HR)

Binary operation : *disjoint union* : $\oplus$

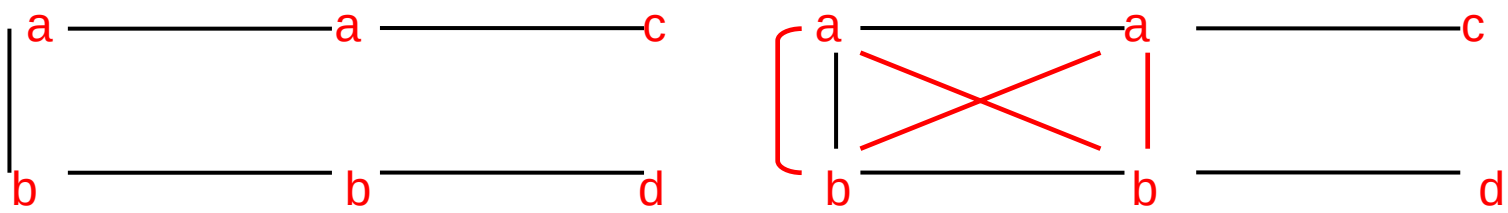
Well-defined up to isomorphism; one takes disjoint copies

→ G $\oplus$ G is *not* equal to G

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

$Add_{a,b}(G)$ is G augmented with undirected edges between every *a*-labelled vertex and every *b*-labelled vertex.

Multiple edges may be created.



The directed version of $\overrightarrow{Add}_{a,b}$ adds directed edges *from* every *a*-labelled vertex *to* every *b*-labelled vertex.

Vertex relabellings

$Relab_{a \to b}(G)$ is G with every label *a* changed into *b*

Variant : $Relab_{h}(G)$ is G with every label *a* changed into *h(a)* for some function $h : C \to C$; C is the *finite* set of labels.

Basic graphs

a : one vertex labelled by a, for each a in C

$\varnothing$: the empty graph (yes, it will be useful !)

Definition : A graph G (*not necessarly simple*) has clique-width $\leq$ k

$\Leftrightarrow$ it can be constructed from basic graphs with the operations

$\oplus$, $\overrightarrow{Add}_{a,b}$, $Add_{a,b}$, $Relab_{a \longrightarrow b}$ and constants **a** with labels *a, b* in a set C of k labels.

Its (exact) clique-width *cwd*(G) is the smallest such k.

Note : It is NP-complete to check if *cwd*(G)=k (input : (G,k)) (Fellows *et al.*)

Cubic approximation algorithms have been given (Oum, Hlineny, Seymour).

Bounded clique-width : cliques, cographs, distance hereditary graphs, every class of bounded tree-width

Unbounded clique-width : tournaments, planar graphs (even square grids).

Comparison with tree-width

For G undirected (Corneil and Rotics) :

$$cwd(G) \leq 3 \cdot 2^{twd(G)-1}$$

For G directed :

$$cwd(G) \leq 7 \cdot 4^{twd(G)-1} - 3^{twd(G)} < 2^{2.twd(G)+1}$$

No polynomial bound : $cwd(G) \leq \textit{poly}(twd(G))$

In both cases :

$$cwd(G) \leq pwd(G) + 2$$

pwd = path-width = tree-width with paths instead of trees

FPT model-checking algorithms

For MS properties, the parameter is clique-width.

For MS_2 properties, the parameter is tree-width and cannot be clique-width.

By Kreutzer, Makowsky *et al.* MS_2 model-checking *needs* restriction to bounded tree-width unless P=NP, ETH, Exptime=NExptime *etc*…

The case of MS_2 *formulas reduces to that of* MS *ones:*

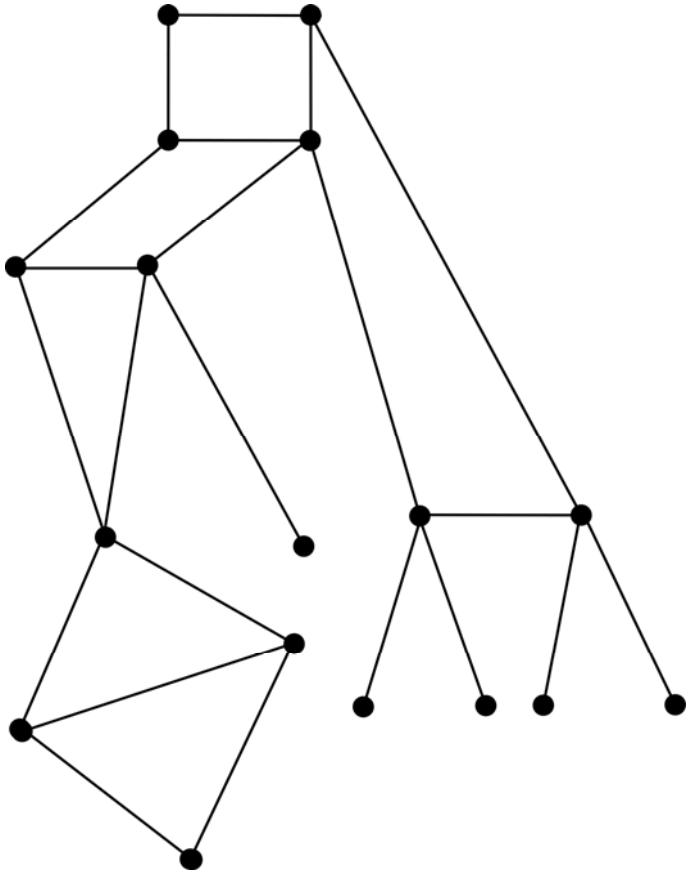
G of tree-width k $\geq$ 2 → Inc(G) has tree-width k,

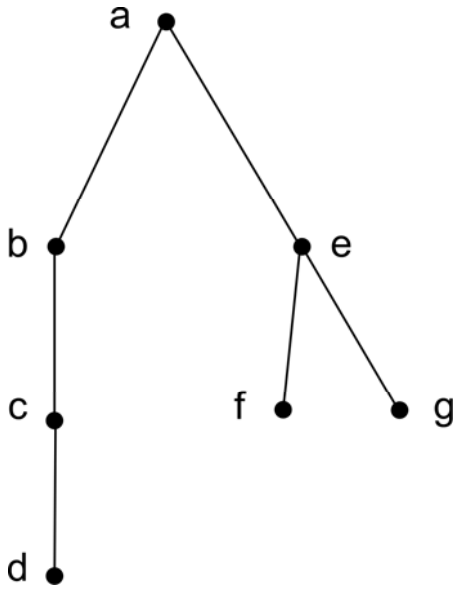
hence, clique-width $\leq 2^{O(k)}$ (exponential blow-up)

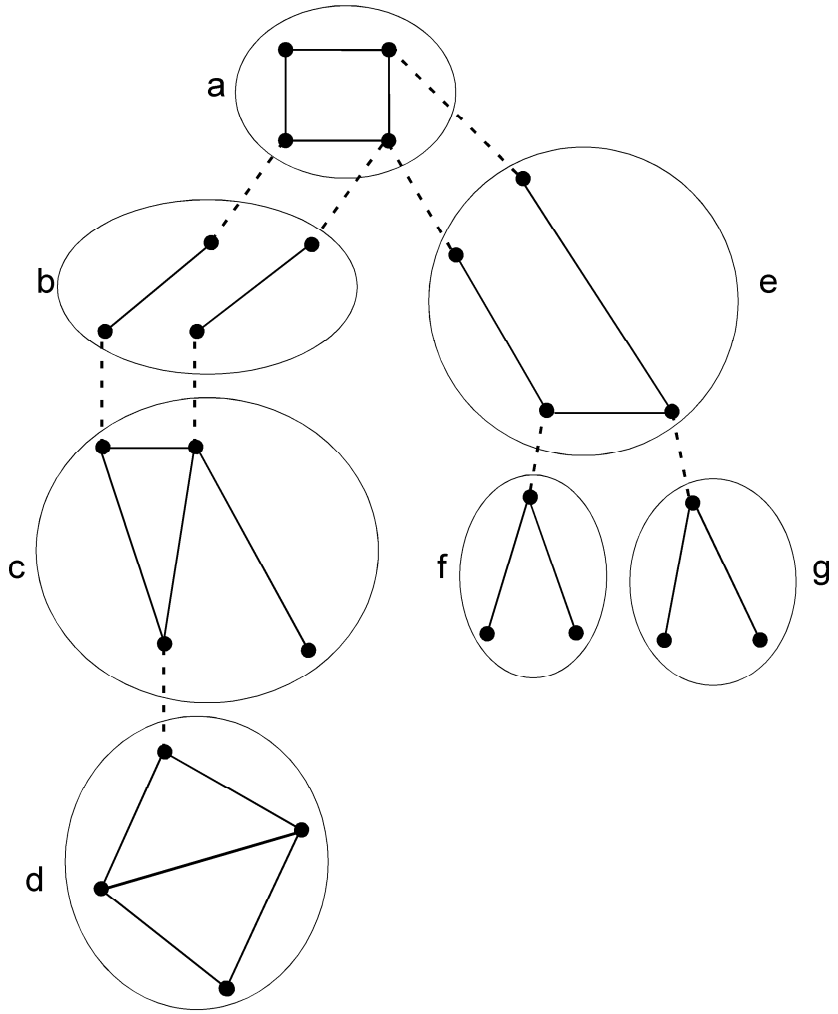
every MS_2 property of G is an MS property of Inc(G)

3. Special tree-width

Tree-decompositions





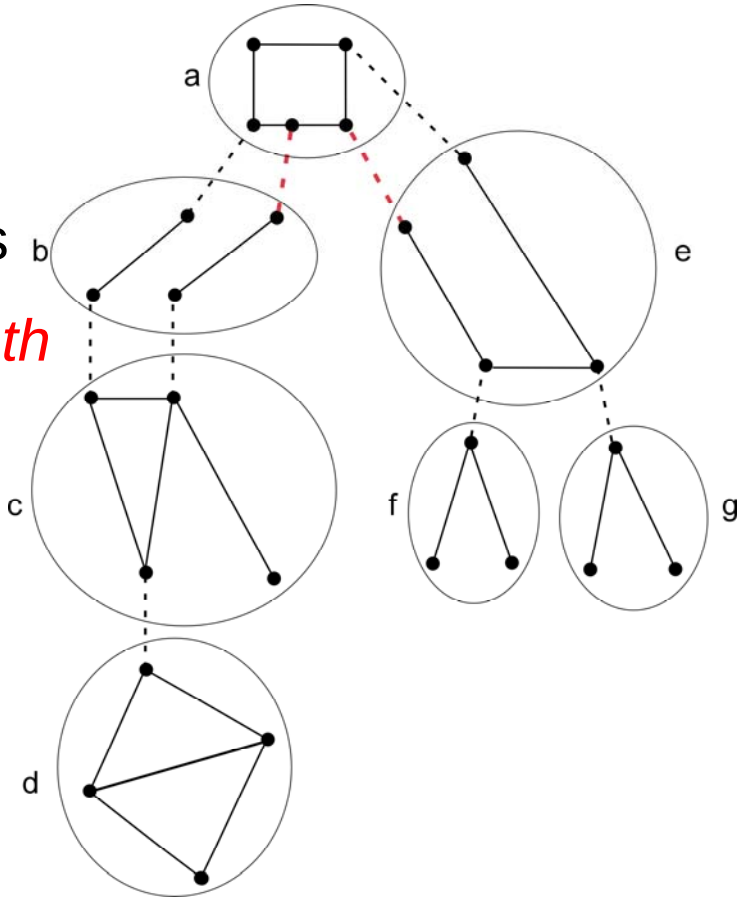


a decomposition of width 3 (= 4 – 1).

Definition: *Special tree-width* is the minimal width of a *special tree-decomposition* (T,f) where :

(a) T is a rooted tree,

(b) the set of nodes whose boxes contain any vertex is a *directed path*



Motivations : (1) Comparison with clique-width (no exp. blow-up)

(2) The automata for checking adjacency are exponentially smaller than for bounded tree-width

Properties of special tree-width

twd = tree-width ; pwd = path-width ; sptwd = special tree-width ;
cwd = clique-width.

1) $twd(G) \leq sptwd(G) \leq pwd(G)$

2) $cwd(G) \leq sptwd(G) + 2$ (for G simple).

whereas $cwd(G) \leq 2^{2.twd(G)+1}$ (exponential is not avoidable)

3) $sptwd(G) \leq 20\,(twd(G)+1).\, MaxDegree(G)$

(for a set of graphs of bounded degree, bounded special tree-width is *equivalent* to bounded tree-width).

4) Trees have special tree-width 1 (= tree-width) but graphs of tree-width 2 have *unbounded special tree-width*.

5) The class of graphs of special tree-width $\leq$ k is closed under:

- reversals of edge directions,
- taking *topological minors* (subgraphs and smoothing vertices) but *not under taking minors*.

Graphs of tree-width 2 have *unbounded special tree-width*

Proof sketch: If $G \otimes *$ (= G augmented with a universal vertex $*$) has special tree-width k, then it has path-width $\leq$ k.

Let G be any tree : $G \otimes *$ has tree-width 2.

If $G \otimes *$ has special tree-width $\leq$ k, then G has path-width $\leq$ k.

But trees have *unbounded path-width*, hence graphs of tree-width 2 have unbounded special tree-width.

Terms that characterize special tree-width;

and construction of automata for MS_2 properties.

Definition: *Special terms*

They use the graph operations that define clique-width for *graphs with multiple edges* (Key point : no "vertex fusion" is needed)

1) The set C of labels contains $\perp$ (to mean "terminated vertex")

2) Operations $Relab_{a \longrightarrow c}$ and $Add_{a,b}$ only if $a, b \neq \perp$

3) Subterms define graphs with $\leq$ 1 vertex labelled by a if $a \neq \perp$

4) $Add_{a,b}(t)$ allowed as subterm only if G(t) has one vertex **x** labelled by a and one vertex **y** labelled by b. Similar definitions for directed graphs.

Edges are added "one by one" and are in bijection with the occurrences of the operations $Add_{a,b}$, that can define multiple edges.

Proposition: (1) G has *special tree-width* $\leq$ k $\Leftrightarrow$ it is defined by a *special term* using $\leq$ k + 2 labels (including the particular label $\perp$)

(2) cwd(G) $\leq$ sptwd(G) + 2

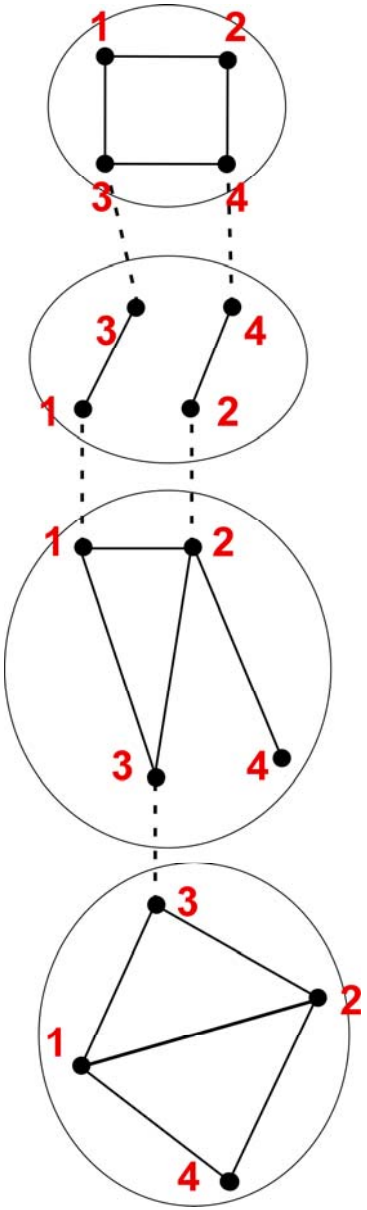
We will compare:

path-width and clique-width,

tree-width and clique-width,

special tree-width and clique-width

Comparing path-width and clique-width :

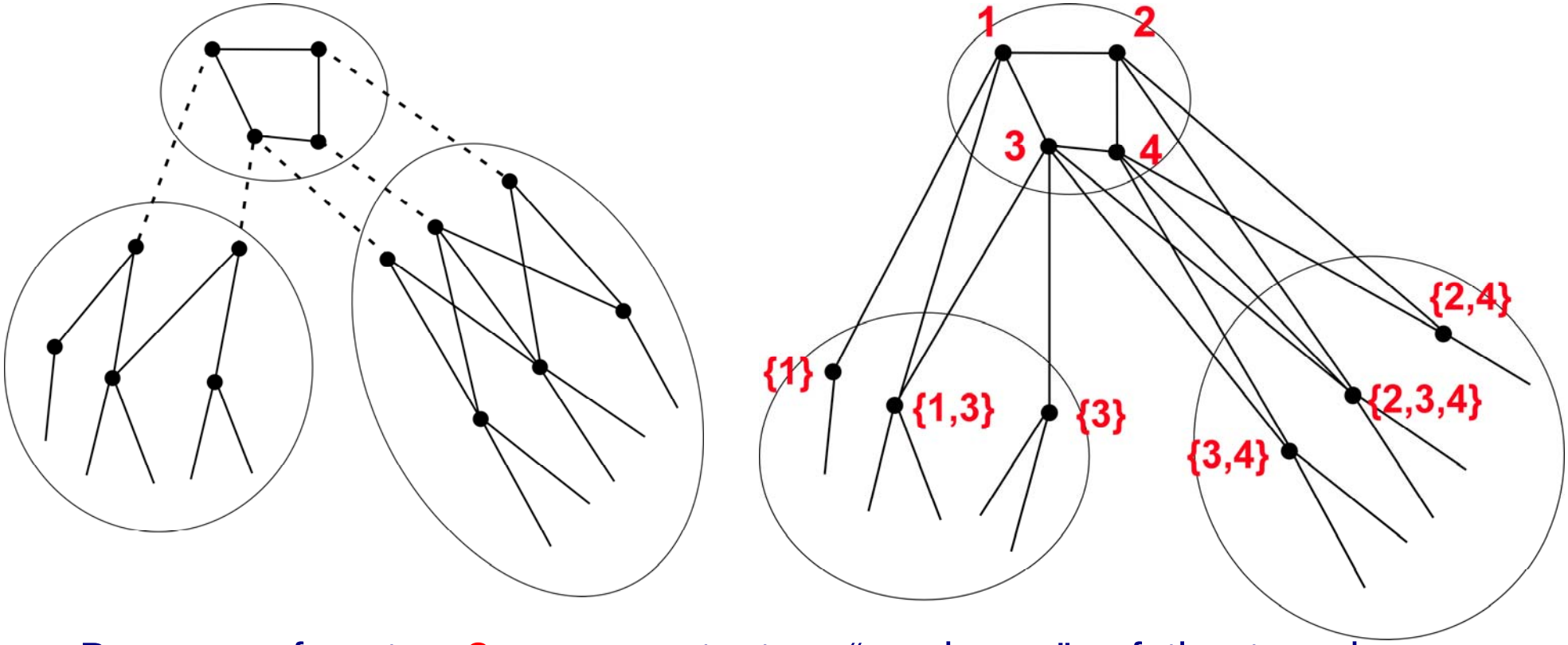


$$cwd(G) \leq pwd(G) + 2$$

Idea : By traversing bottom-up the path decomposition, by using 4 colors + $\perp$, the clique-width operations can add, *one by one*, new vertices (using $\oplus$ **i**) and new edges (using $Add_{a,b}$ or $\overrightarrow{Add}_{a,b}$).

$\perp$ is for "terminated vertices".

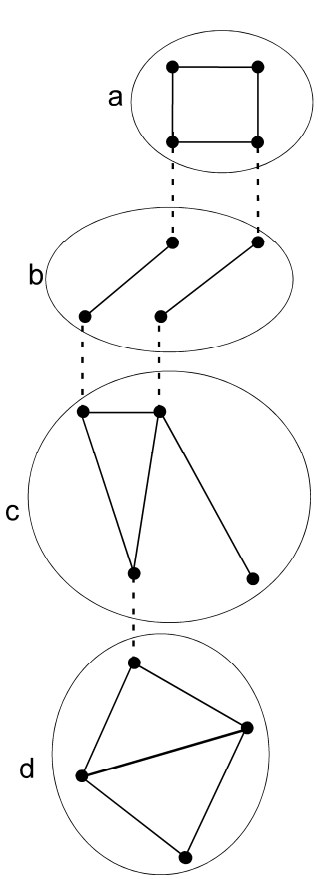
For tree-width : $cwd(G) \leq 2^{2.twd(G)+1}$

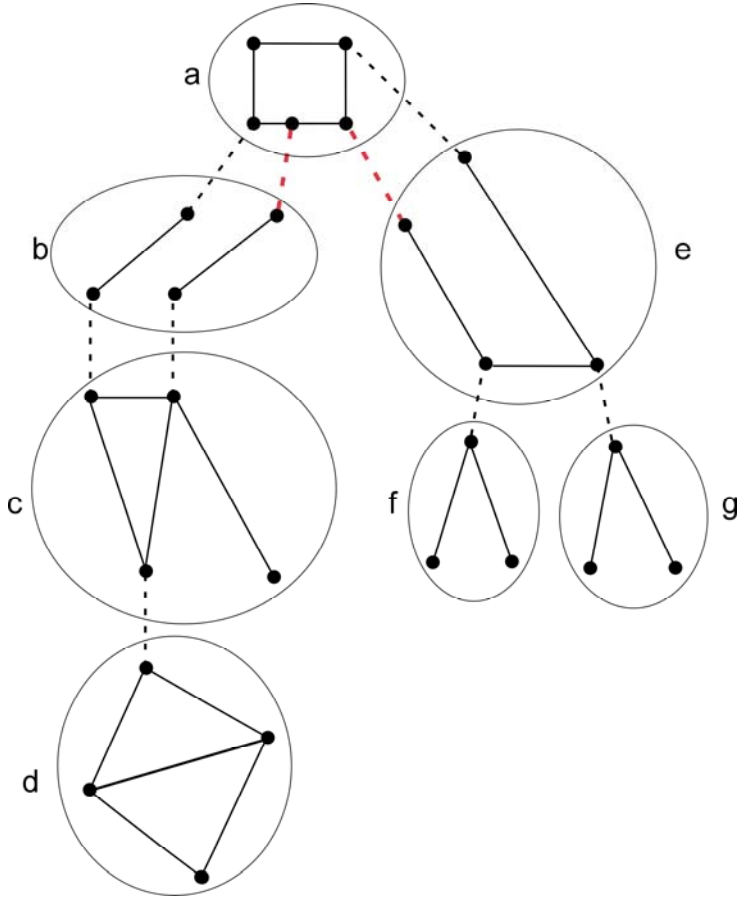


Because of vertex **3**, common to two "son boxes", of the tree-dec, the previous method does not work. (It does not allow fusion of vertices).

If a box of the tree-decomposition has k vertices, then 2^k-1 labels are necessary to specify how the vertices below it are linked to its vertices. ($2^{2k} - 1$ for directed graphs).

For special tree-width (as for path-width) : cwd(G) ≤ sptwd(G)+2





The red dotted edges are not incident.

Two “brother” boxes (*b, e*) are disjoint.

This is the characteristic property of *special tree-decompositions*

Special tree-width is interesting for model-checking of MS_2 properties (as we will see) but the *parsing* problem is open :

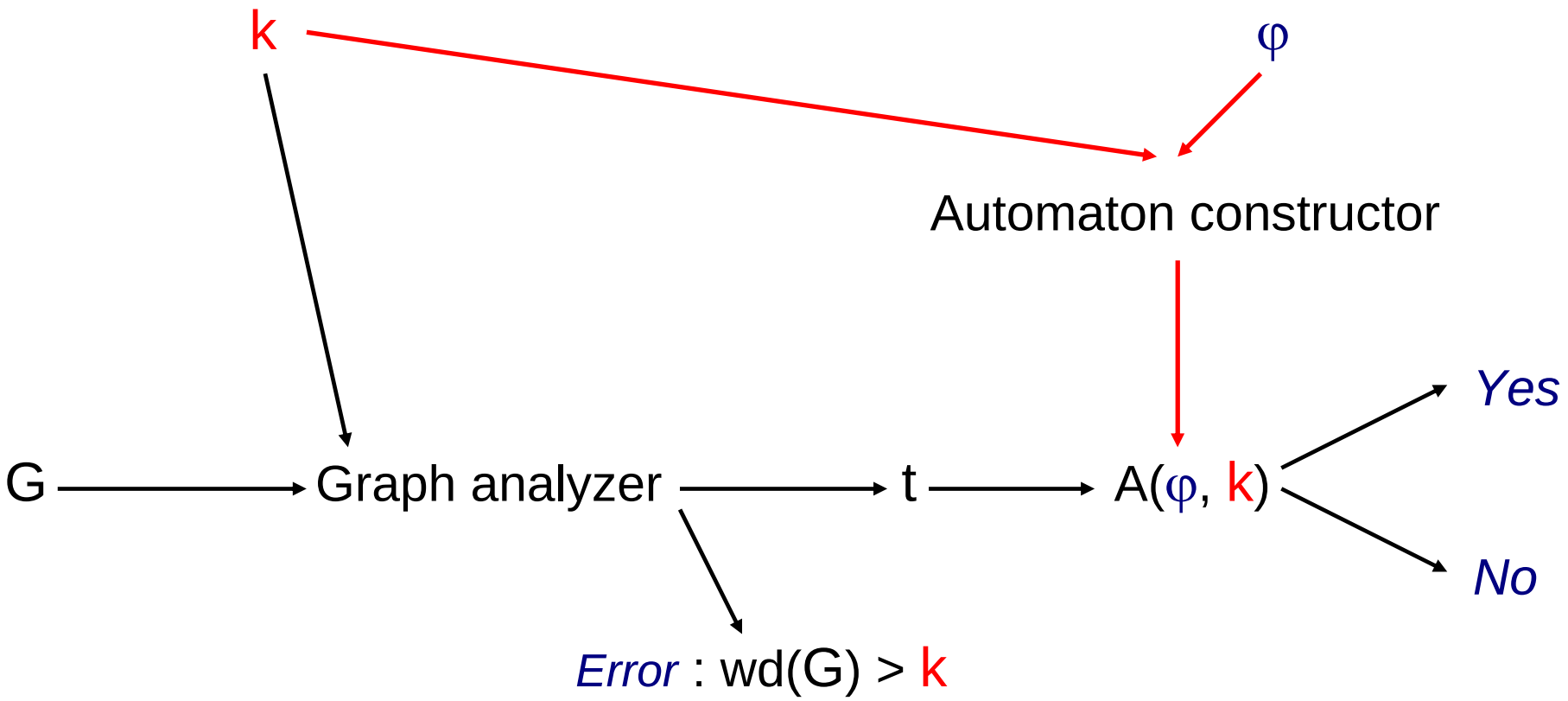
Can one find an $O(n^{g(k)})$ algorithm ?:

- that reports that the input graph G (with n vertices) has special tree-width more than k or

- outputs a special tree-decomposition witnessing that the special tree-width of G is $\leq f(k)$ (for a fixed function f hopefully not exponential).

Note: We can use the algorithms producing path-decompositions

4. Automata for MS model-checking



Steps ⟶ done "once for all", independent of G

A(φ, k): automaton on terms (wd = tree-width or clique-width or equivalent)

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

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

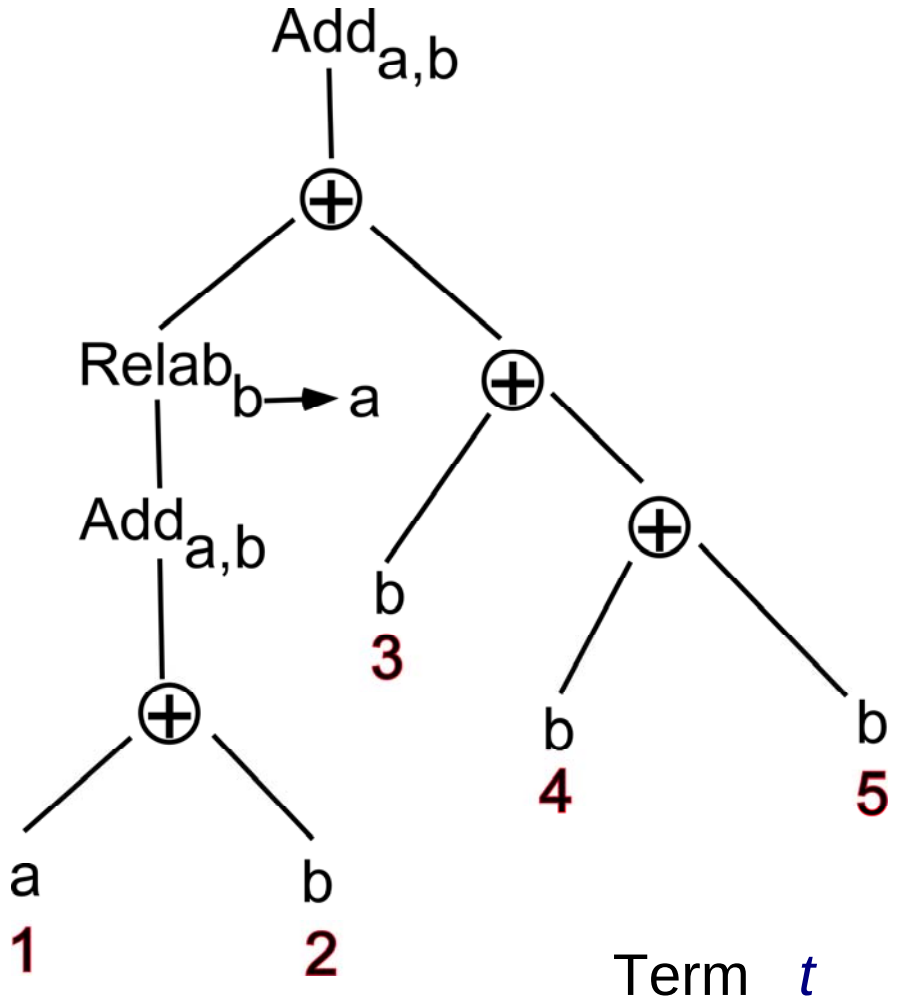
F = the corresponding set of operations and constants :

a , $\varnothing$, $\oplus$, $Add_{a,b}$, $Relab\ a \longrightarrow b$

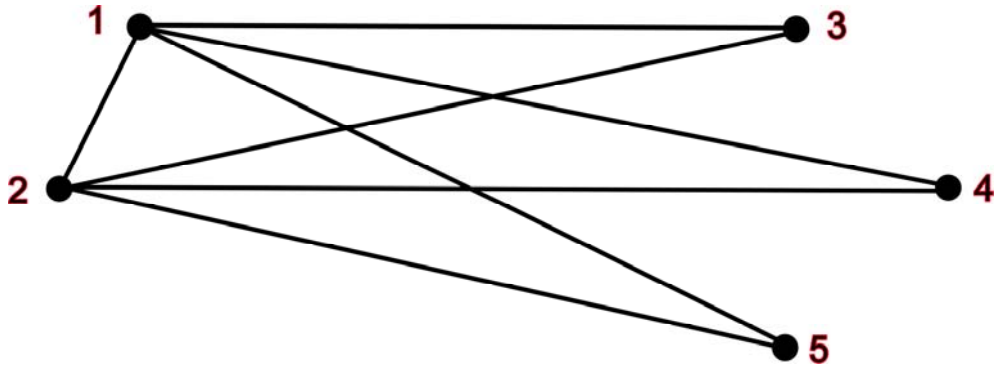
G(*t*) = the graph defined by a term *t* in **T**(F).

Its vertices are (in bijection with) the occurrences of the constants 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 MS 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);

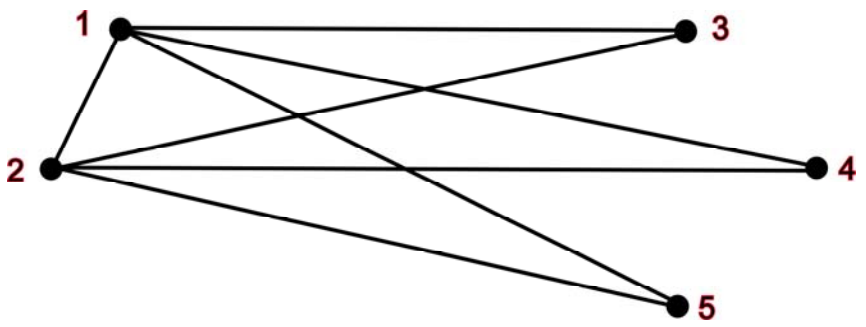
2) a term *s* in $\mathbf{T}(F^{(n)})$ encodes a term *t* in $\mathbf{T}(F)$ and an assignment of sets $V_1,\ldots,V_n$ to the set variables $X_1,\ldots,X_n$:

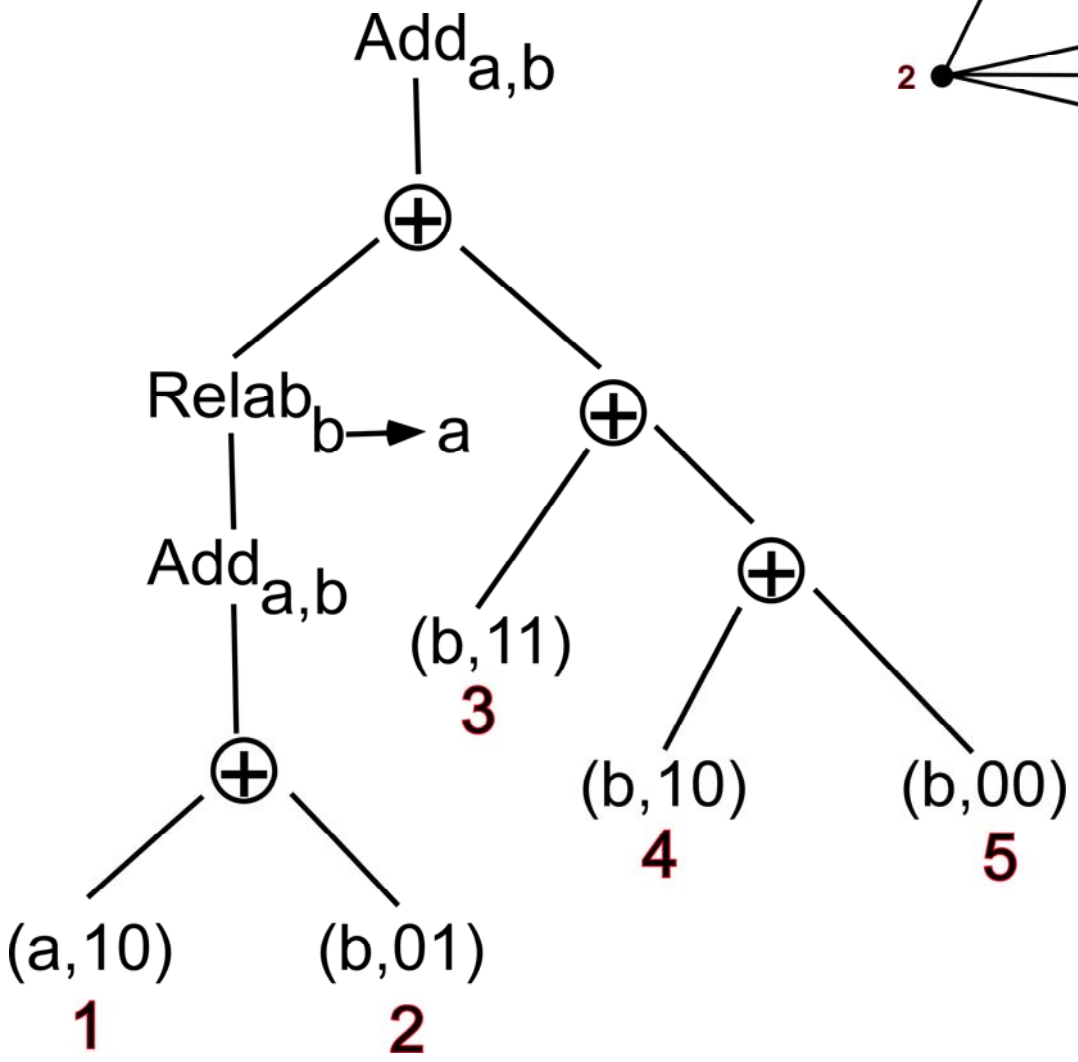
if *u* is an occurrence of $(\mathbf{a}, (w_1,..,w_n))$, then

$w_i = 1$ 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:

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

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

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, or, for undirected graphs, the equivalent graph complexity measure *rank-width*)

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

Atomic formulas : discussed below.

For $\wedge$ and $\vee$: product of two *complete* automata

(deterministic or not)

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

for a complete *deterministic* automaton

Quantifications: Formulas are written without $\forall$

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

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

where pr is the "*projection*" that eliminates the last Boolean.

One obtains a *nondeterministic* automaton.

oOo

The number of states is an h-iterated exponential,

where h = maximum nesting of negations.

Tools for constructing automata

Substitutions and inverse images ("cylindrifications").

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

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

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

We take

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

This construction preserves determinism and the number of states.

2) From $A(\varphi(X_1, X_2))$, we can get $A(\varphi(X_3, \overbrace{X_1 \cup (X_2 \setminus X_4)}^{\textit{Set term}}))$ 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 :

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

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

Hence:

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

The inductive construction (continued) :

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

$t * (V_1,\ldots,V_n)$ in $L(\varphi(X_1,\ldots,X_n))$

Intuition : in all cases, the *state* 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*), giving the *accepting states*.

Atomic formulas (1) : edg(X_1,X_2) for directed edges

Vertex labels are from a set C of k labels.

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

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 : $X_1 = \emptyset$, $X_2 = \emptyset$

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

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

a(2) : $X_1 = \emptyset$, $X_2 = \{w\}$, 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

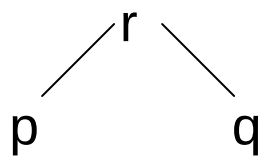
Transition rules

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 $\overrightarrow{Add}_{a,b}$

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

Other atomic or basic formulas (2)

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

Card $_{p,q}$ (X_1) : cardinality of X_1 is p modulo q,

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

→ Easy constructions with small numbers of states : 2, 2, 3, q, q+1.

Sizes of some deterministic automata : k = bound on clique-width

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	$2^{O(p.k.k)}$	$2^{O(k.k)}$	$2^{2^{O(k)}}$

For connectedness, the minimal (deterministic) automaton has more than $2^{2^{k/2}}$ states.

Other constructions yield *nondeterministic* automata for connectedness and for its negation with $2^{O(k.k)}$ states

Difficulties in practice :

Parsing : construction of terms (based on tree-decompositions or other graph decompositions). The *linear-time exact* parsing algorithm by Bodlaender (for tree-width) and the *cubic approximate* parsing algorithm by Hlineny & Oum (for clique-width via *rank-width*) are not implementable.

Bodlaender reports about usable algorithms for (non-random) graphs with 50 vertices and tree-width $\leq$ 35

Specific algorithms : (1) Flow-graphs of structured programs have tree-width at most 6 and tree-decompositions are easy from the parse trees of programs (Thorup).

(2) For certain graph classes of bounded clique-width defined by forbidden induced subgraphs, optimal clique-width terms can be constructed in polynomial time (by using *modular decomposition*).

The *sizes* of the automata A(φ, k).

They may be too large to be practically compiled.

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*(φ ,k) has a manageable number of states.

Examples : 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

Examples of automata too large to be constructed, i.e., “compiled”, even “directly”, without using the general construction.

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

for k = 5 : 3-colorability, clique

for k = 4 : connectedness.

This is not avoidable :

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 an upper bound with a bounded nesting of exponentiations (Meyer & Stockmeyer, Weyer, Frick & Grohe).

Remedy : using *fly-automata.*

States and transitions are not listed in huge tables.

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

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.

Some transitions :

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

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

Transitions for $\oplus$: union of sets of types.

This method works for formulas with no quantifier alternation but using the "basic formulas".

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($X_i \cup X_j$) $\wedge$ .....)

with NoCycle($X_i \cup X_j$) for every $i < j$.

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) $\wedge$ ...)

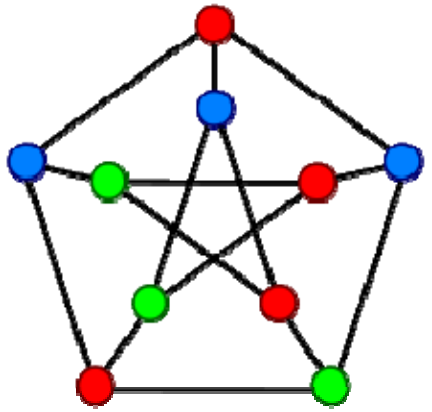
with Link(X_i , X_j) for each edge v_i ---- v_j of H ; means that there exists an edge between X_i and X_j, and Conn(X_i) means that X_i induces a connected graph.

Some experiments with fly-automata (by Irène Durand, LaBRI)

3-colorability of the 6 x 300 grid (of clique-width 8) in less than 2 hours,

4-acyclic-colorability of the Petersen graph (clique-width 7) in 17 minutes.

(3-colorable but not acyclically;
red and **green** vertices
induce a cycle).



New tool : Annotations

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

Example :

At position u in a term *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.

These sets can be computed in *linear time* by means of a top-down traversal of *t*.

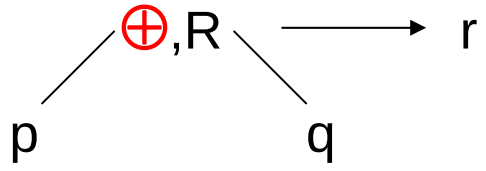
Certain automata on annotated terms may have less states.

Example : *edg*(X_1, X_2) : 2k+3 states instead of k^2 +k +3 (cf. page 44):

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 53.

4.2 Automata for the model-checking of MS_2 formulas

We need :

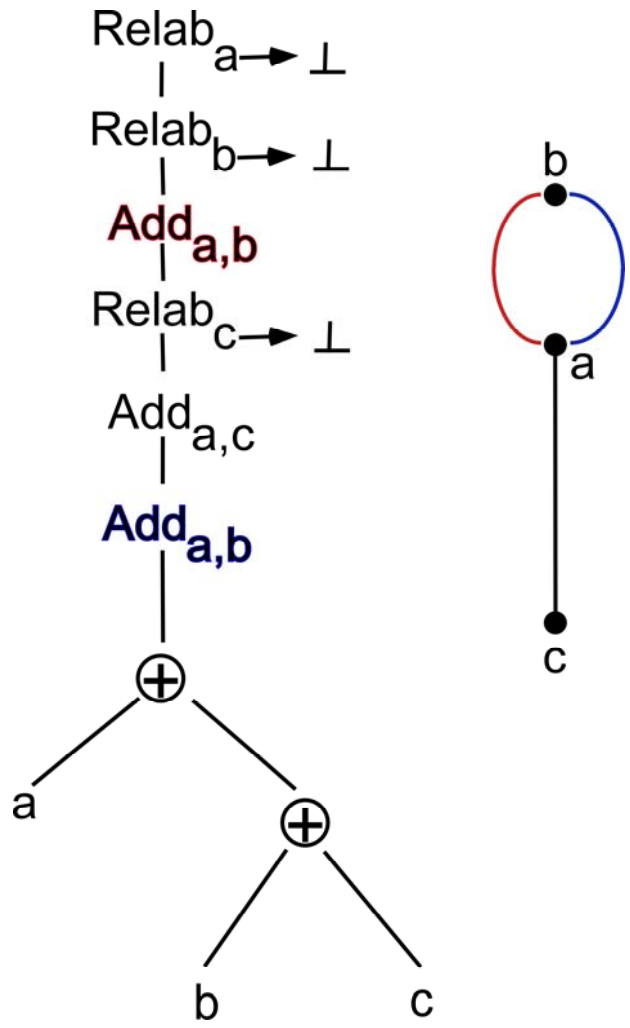
1) Terms to represent graphs, over appropriate operations.

2) A representation of vertices *and edges* by occurrences of operations and constants in these terms.

2.1 : For "clique-width" terms : we have *no* good representation of edges because each occurrence of $Add_{a,b}$ may add simultaneously an unbounded number of edges.

2.2 : For special terms : each edge is produced by a unique occurrence of $Add_{a,b}$. This gives what we want for graphs of bounded *special tree-width* (but not for bounded tree-width).

Using special terms :



The leaves represent the vertices.

The nodes labelled $Add_{a,b}$ and $Add_{a,c}$ represent the edges ; each occurrence of $Add_{a,b}$ represents one of the two parallel edges

The automata for edg(X,Y) and inc(X,Y) (incidence) have $O(k^2)$ and O(k) states respectively for *sptwd* at most k.

2.3 : Case of terms characterizing tree-width

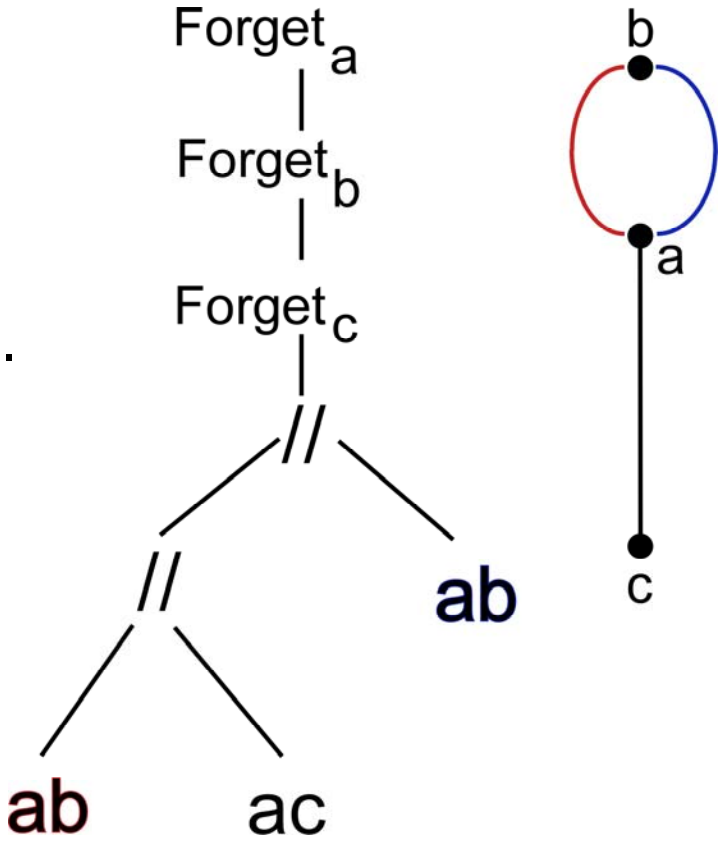
First idea : make them into "clique-width terms" for the *incidence graph*. But:

clique-width $\leq$ $2^{O(\text{tree-width})}$ → too large automata.

Second idea : handling them "directly", as for "clique-width terms"

The difficulty is to have a bijection between nodes in the term and the vertices and edges of the graph.

First possibility



Vertices are in bijection with 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 $O(k^2)$ for sptwd).

Too bad for a basic property !

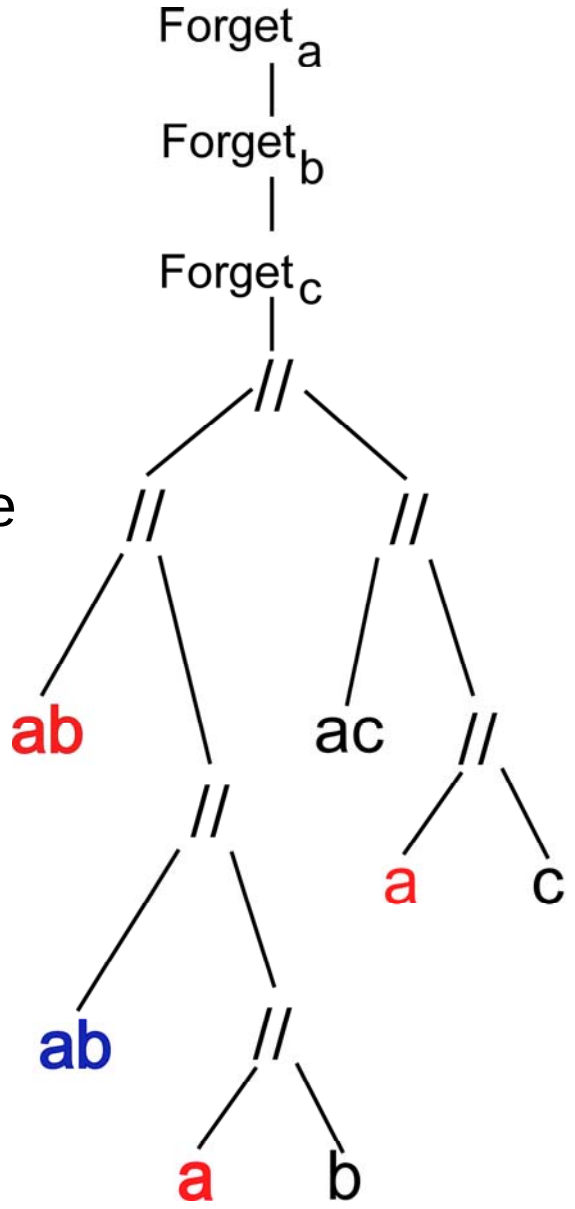
Second possibility

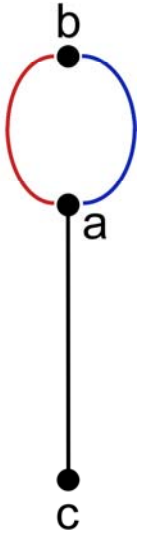
Vertices are at the leaves,

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 second possibility.

The responsible is // (that is not needed for representing special tree-decompositions).

An improvement using annotations

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

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 $2^{\Theta(k.k)}$).

Remarks :

incidence : in(X,Y) uses k^2 + 3 states (for undirected graphs) (only k+3 states for directed graphs).

adjacency : edg(X,Y), can be written $\exists Z$ (in(Z,X) $\wedge$ in(Z,Y)) (for undirected graphs) which gives a deterministic automaton with $2^{O(k.k)}$ states.

With this annotation, incidence and adjacency are handled separately on “redundant” representations of graphs by terms.

Conclusions

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 (in order to keep flexibility, by allowing variations on the input sentences).

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. Can one adapt fly-automata to counting and optimization problems?

4. Special tree-width is less powerful than tree-width, but the constructions of automata are simpler. The parsing problem is open.

5. In many cases (in particular bounded degree) special tree-width is linearly bounded in tree-width.