The verification of monadic second-order properties of structured graphs

Bruno Courcelle

Université Bordeaux 1, LaBRI & Institut Universitaire de France

References:

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

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

Full version accepted for publication in *J. of Applied Logic*

For other references see : http://www.labri.fr/perso/courcell/ActSci.html

Summary

1. Logic and construction of algorithms ;

   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. Open problems and conclusion.

1. Logic and construction of algorithms.

Hard (NP-complete) graph problems :

3-vertex coloring, vertex cover,

SATisfiability of a propositional formula (can be viewed as a problem about labelled graphs).

These problems have polynomial-time algorithms for graphs of particular classes.

Our objective : Meta-theorems giving algorithms for *classes* of *structured* graphs and *classes* of *logically defined* problems.

Logical expression of graph properties

Graphs as logical structures:

Let G be directed or undirected, and *simple* (no parallel edges).

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

Logical languages : First-order logic

Examples : 1. Every vertex has an outgoing edge: $\forall u \, \exists v \; edg(u,v)$

2. G undirected has diameter at most k : here for k = 2 :

$\forall u, v \, (v \neq u \Rightarrow edg(u,v) \vee \exists w \, [edg(u,w) \wedge edg(w,v)])$.

FO logic expresses only local properties (in a precise technical sense).

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.

(“A set of edges of G ” is here a binary relation over V_G).

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

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 :*

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

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

$\forall$ X { "X is R-closed" $\wedge$ x $\in$ X $\Rightarrow$ y $\in$ X }

where "X is R-closed" is :

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

The relation R can be defined by a formula as in :

$\forall$x,y (x $\in$ Y $\wedge$ y $\in$ Y $\Rightarrow$ **TC**("u $\in$ Y $\wedge$ v $\in$ Y $\wedge$ edg(u,v)" ; x, y)

(where Y is free) to expressing 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 all vertices of same degree.

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
MSO, clique-width < k	Decidable	Cubic-time
MSO, unbdd cwd.	Undecidable	Conjecture : not FPT

Clique-width (cwd) is a graph parameter defined below.

FPT means here : takes time $f(k).n^c$ where k = *cwd* of input graph and

f(.) and c are fixed, depending on the property to check.

Model-checking problems

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

It is NP-complete for (even planar) graphs of degree ≤ 4 (Dailey, 1980). Hence the *degree* is *not* a good parameter for obtaining an FPT algorithm.

Tree-width and clique-width are good in this respect, and even for all MSO properties.

Both parameters are based on hierchical decompositions.

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

Hierarchical graph decompositions :

many notions;
all of them represent a graph as a tree of smaller components.

They are useful :

as preprocessings in algorithms,

for study of graph structure (modular decomposition for comparability graphs, tree-decompositions for the *Graph Minor Theorem*),

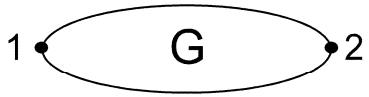
for graph grammars.

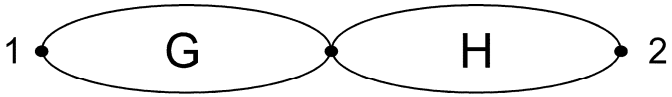
Here: formalized by *terms* over "graph concatenations".
→ graphs in an algebraic setting,
→ linear notation for graphs,

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

Graphs with distinguished vertices marked 1 and 2, generated from

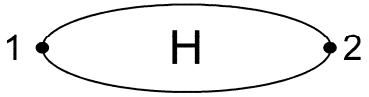
e = 1 → 2 by the operations of *parallel-composition* // and *series-composition* •

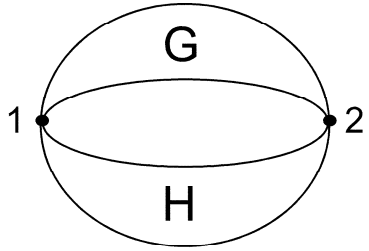




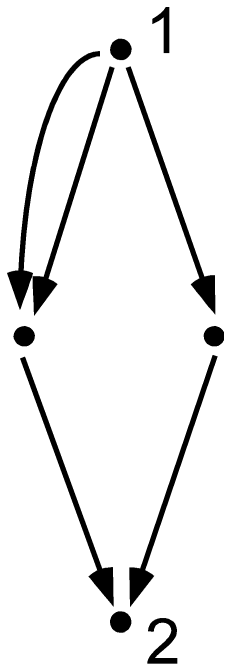
G • H

((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)

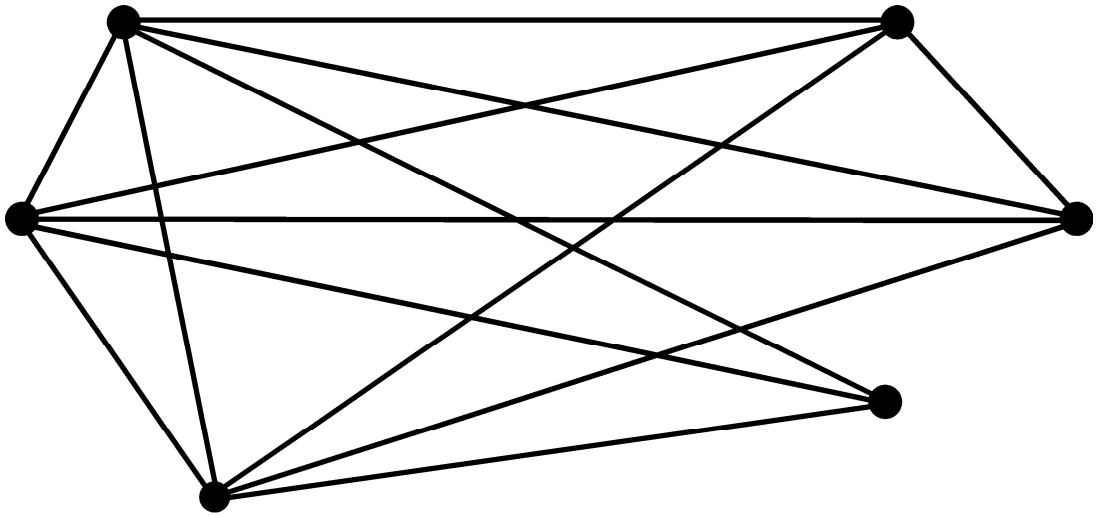
Undirected graphs generated by $\oplus$, *disjoint union* and $\otimes$, *complete join* from **a**, a vertex without edges (up to isomorphism); $\otimes$ is defined by :

$G \otimes H = G \oplus H$ 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.

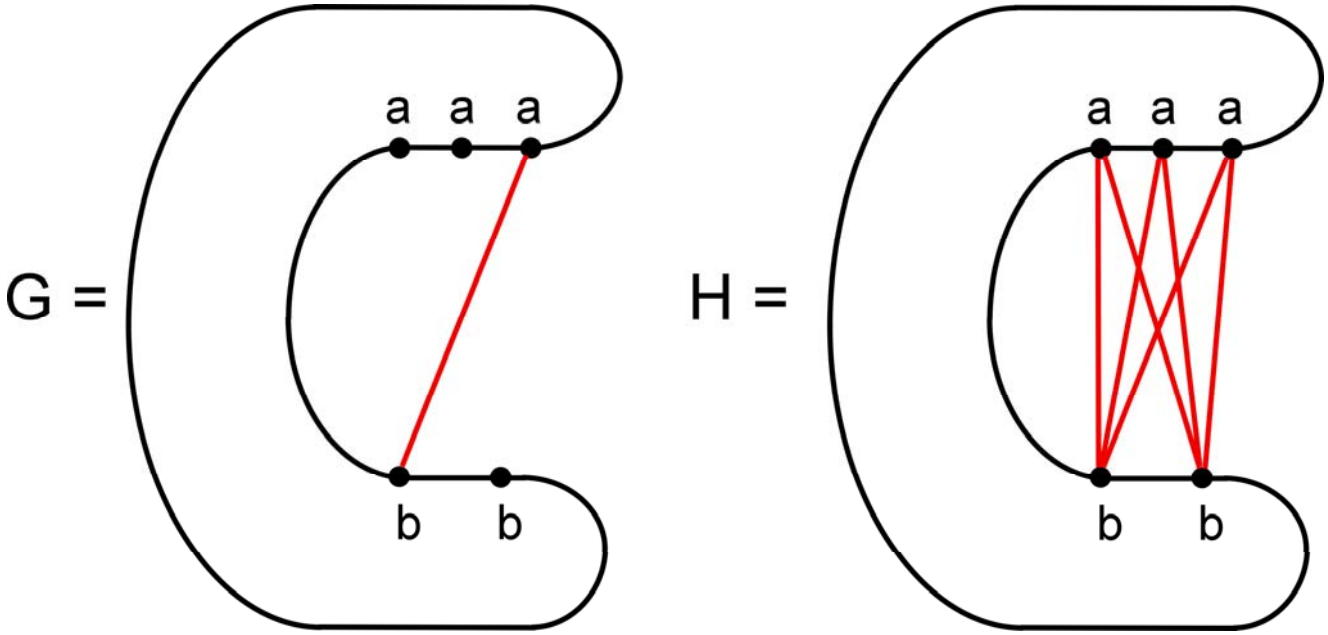
We use labels: *a*, *b*, *c*, ..., *d*. Each vertex has *one* label ; *several* vertices may have the same label. A vertex labelled by *a* is an *a-port*

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.

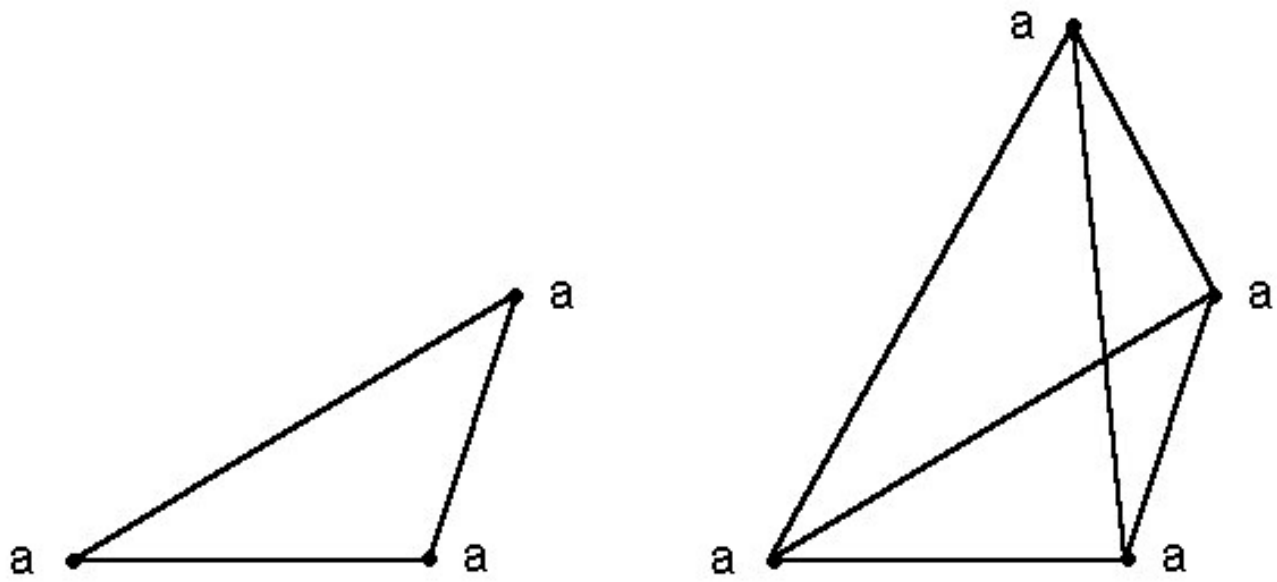
Vertex relabellings :

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

Basic graphs : those with a single labelled 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 cwd 2 (and *unbounded tree-width*)



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

Cliques are defined by the equation :

$$K = Relab_{b \to a}(Add_{a,b}(K \oplus \mathbf{b})) \cup \mathbf{a}$$

Examples of bounded clique-width:

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

Trees and *distance hereditary graphs* have clique-width at most 3.

Tree-width $\leq$ k implies clique-width $\leq$ f(k).

Examples of unbounded clique-width:

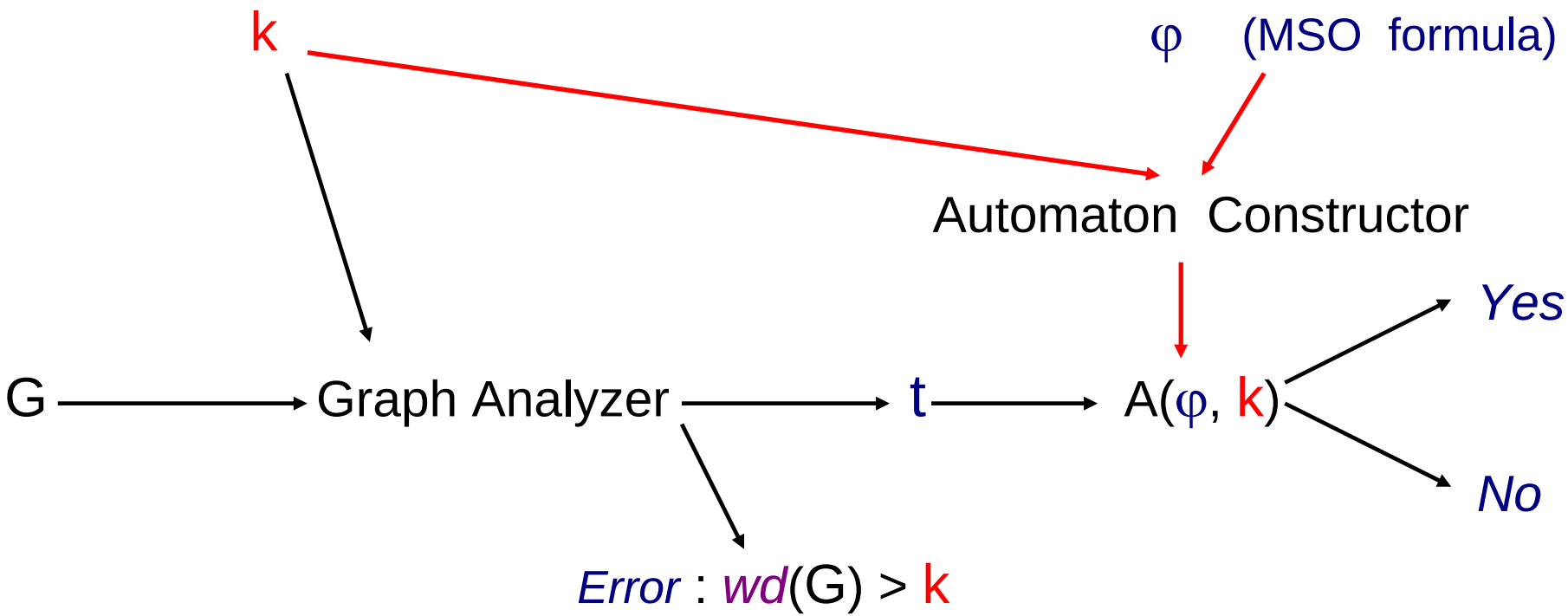
Planar graphs (even of maximum degree 3),

Interval graphs.

Fact : Clique-width is sensible to edge directions :

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

3. From MSO formulas to automata



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

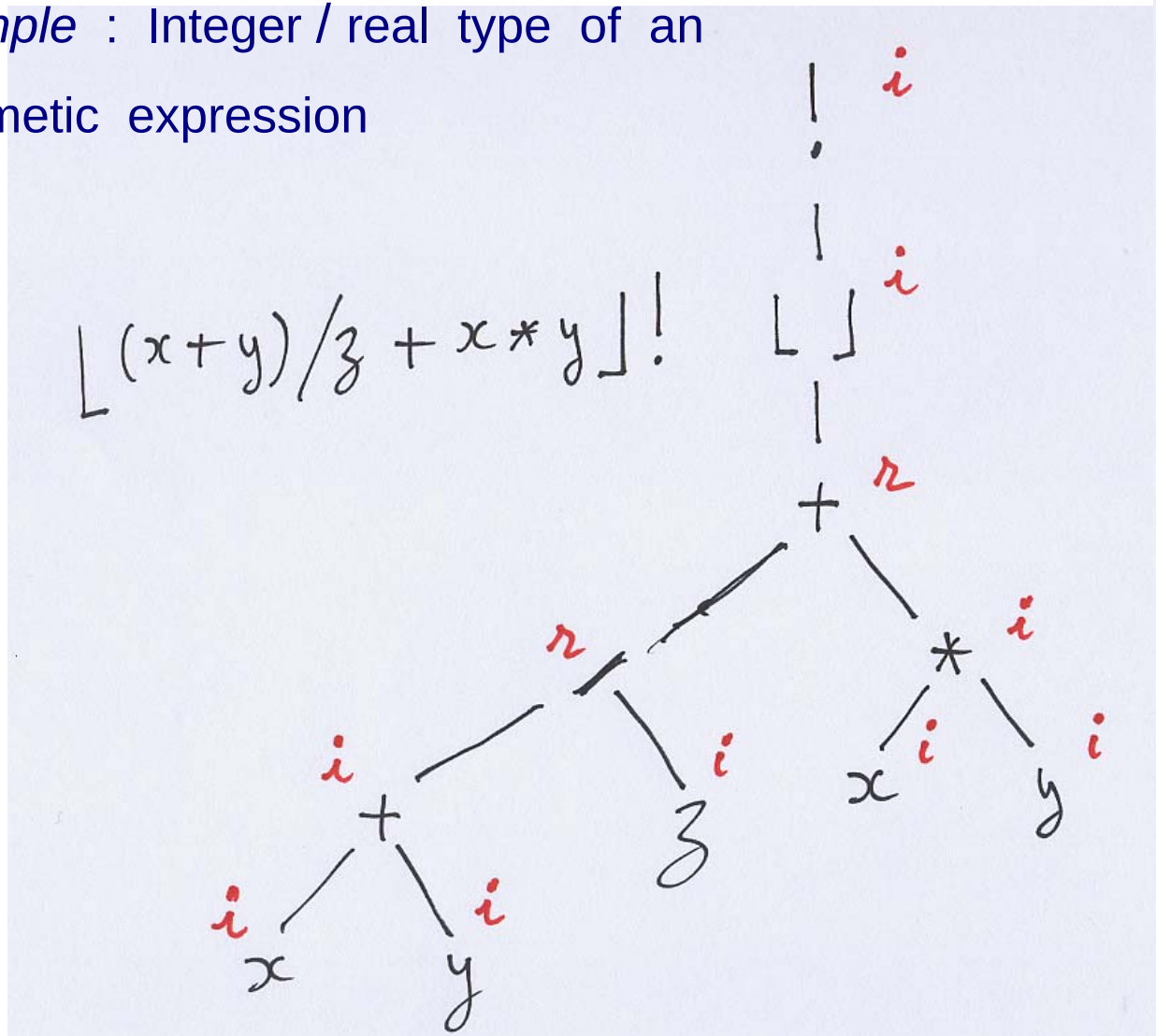
A(φ,k): finite automaton on terms t

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

(Tree-decompositions also have algebraic expressions).

Finite automata on terms

Example : Integer / real type of an arithmetic expression



Bottom-up computation of i / r / Error type using the rules above.

Construction of automaton 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 :

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

G(t) = the graph defined by a term t in $\mathbf{T}$(F). Its vertices are (in bijection with) the occurrences of the nullary symbols in t .

By induction on the structure of φ, one constructs a finite (bottom-up) deterministic automaton A(φ, k) that recognizes:

$$\{ t \in \mathbf{T}(F) \ / \quad G(t) \ \models \varphi \}$$

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 term t defining the input graph must be constructed. (This construction is similar to the parsing step in compilation).

4. Practical difficulties and (some) remedies.

1. Parsing : 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 difficult to implement.

   The situation is similar if tree-decompositions and tree-width are used instead of “clique-width” terms.

2. 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 (because $\exists$ introduces *nondeterminism* and each negation needs a *determinization* that can produce 2^n states for an automaton with n states.)

There is no alternative construction giving a fixed bound on nestings of exponentiations (Meyer & Stockmeyer, 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, Cycle

Property	Partition $(X_1,\ldots,X_p)$	edg(X,Y)	NoEdge	Connected, Cycle for degree $\leq p$	Path(X,Y)	Connected, Cycle
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, Cycle (i.e., has cycles).

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.

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 :

$Add_{a,c}$: $q \longrightarrow$ { {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.

Using fly automata works for formulas with no (or few) 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)* = { $v_1,\ldots,v_p$ }

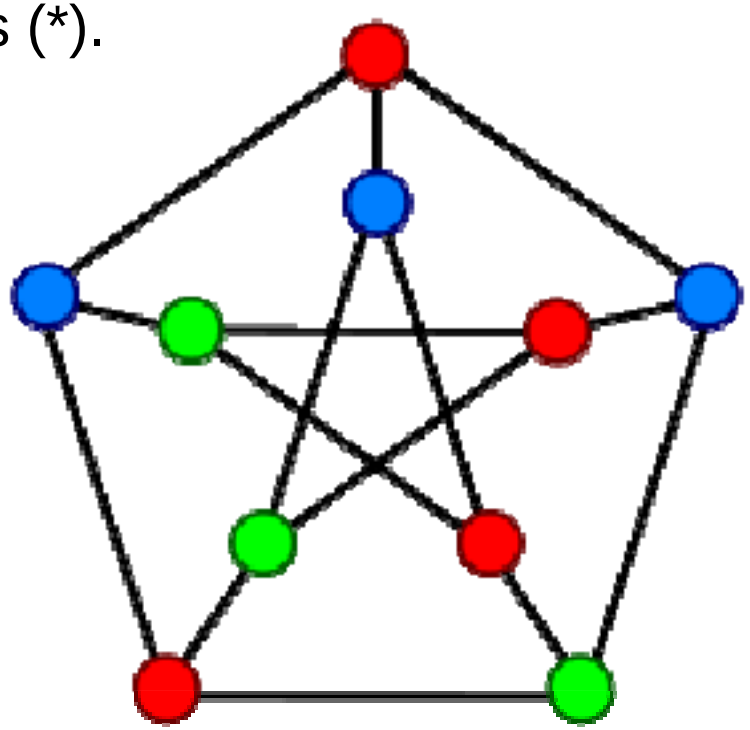
$\exists X_1,\ldots,X_p$ (Disjoint($X_1,\ldots,X_p$) $\wedge$ Conn(X_1) $\wedge$ ... $\wedge$ Conn(X_p) $\wedge$ ...
... $\wedge$ Link(X_i , X_j) $\wedge$ ...)

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 10 minutes.

3-colorability of the Petersen graph (clique-width 7) in 1.1 second,
its 4-acyclic-colorability in 4 minutes (*).

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



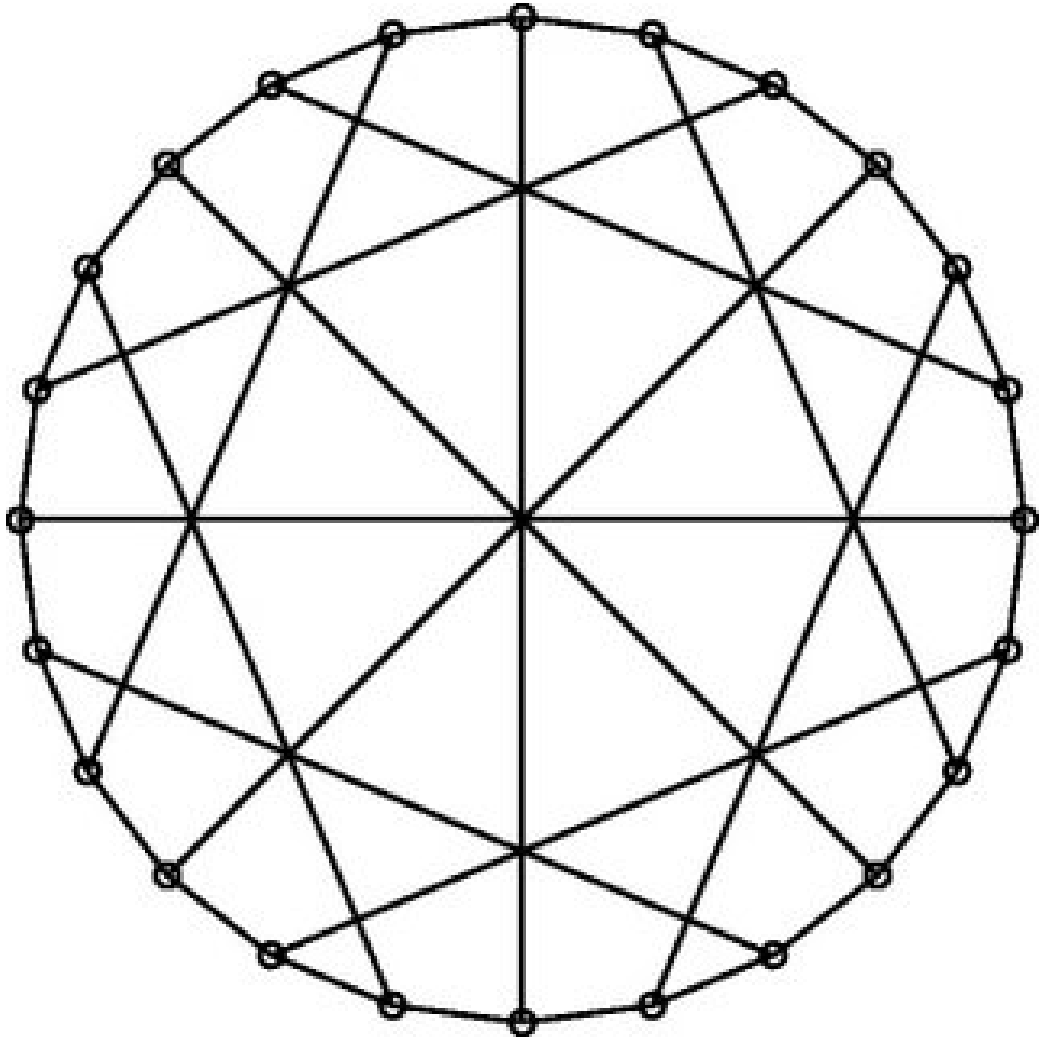
(*) For a term with *annotations*
(a kind of preprocessing).

The McGee graph

24 vertices,

36 edges,

clique-width $\leq$ 10.



3-colorability in 7 minutes,

3-AC-colorability in 21 hours (11 hours with annotated term).

5. 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 with no (or few) 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.

Bonus : Bounded tree-width.

If we replace a graph G by its incidence graph inc(G) (where each edge of G becomes a vertex), then, monadic second-order formulas interpreted over inc(G) can use quantifications on sets of edges. They have more expressive power.

Model-checking with finite automata can be done for graphs of bounded tree-width for such formulas.

Tree-width is well-known. Below we show its algebraic expression by means of appropriate graph operations.

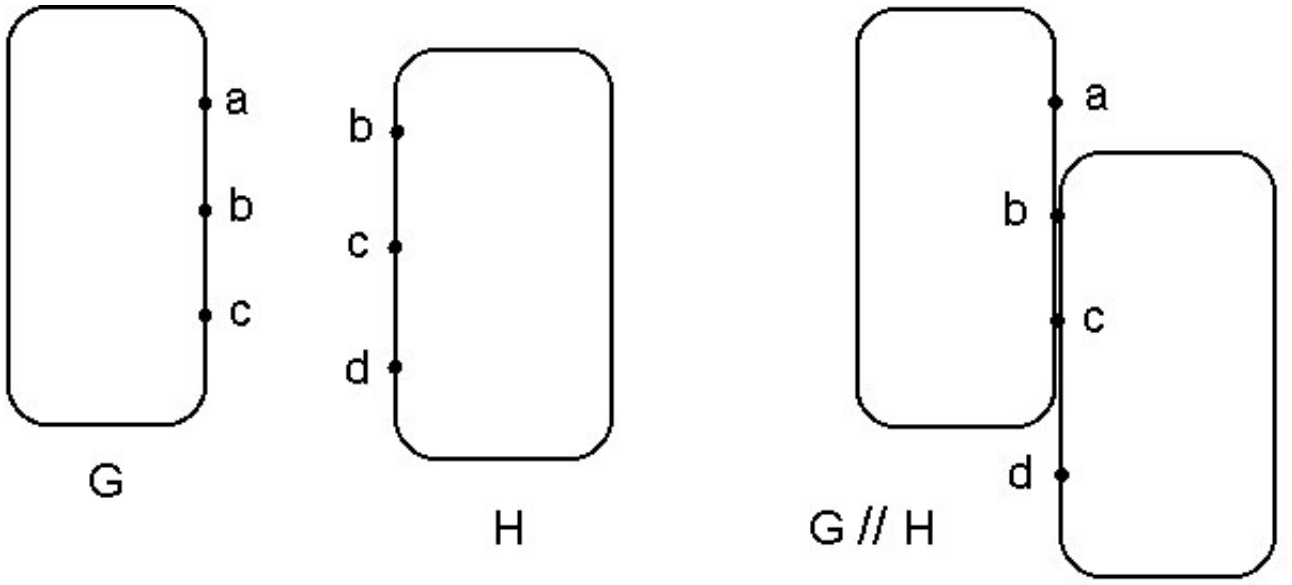
Graph operations that characterize tree-width

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)

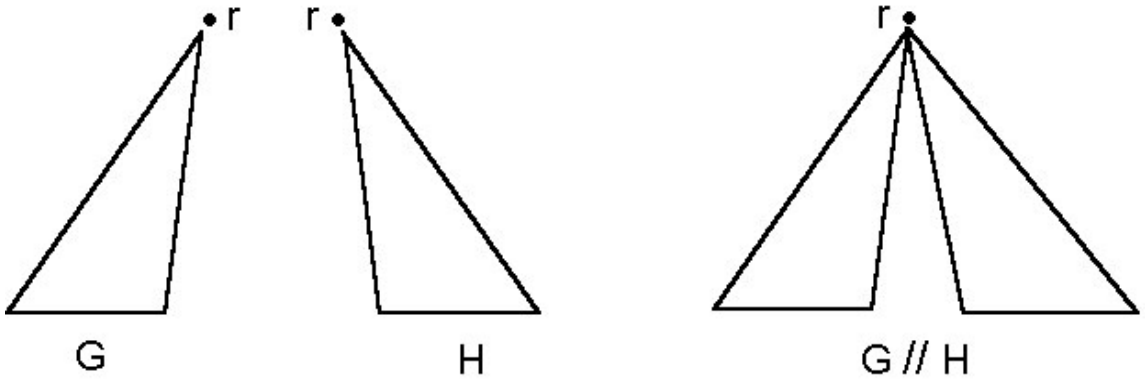
Nullary operations denote *basic graphs* : 1-edge graphs, isolated vertices.

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

Example : Trees

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

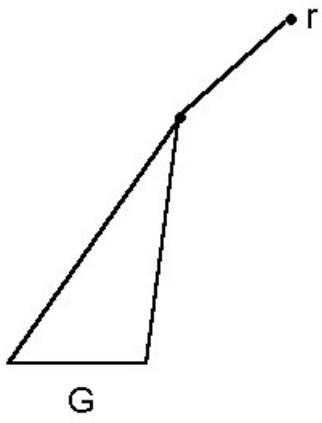
Fusion of two trees at their roots :



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

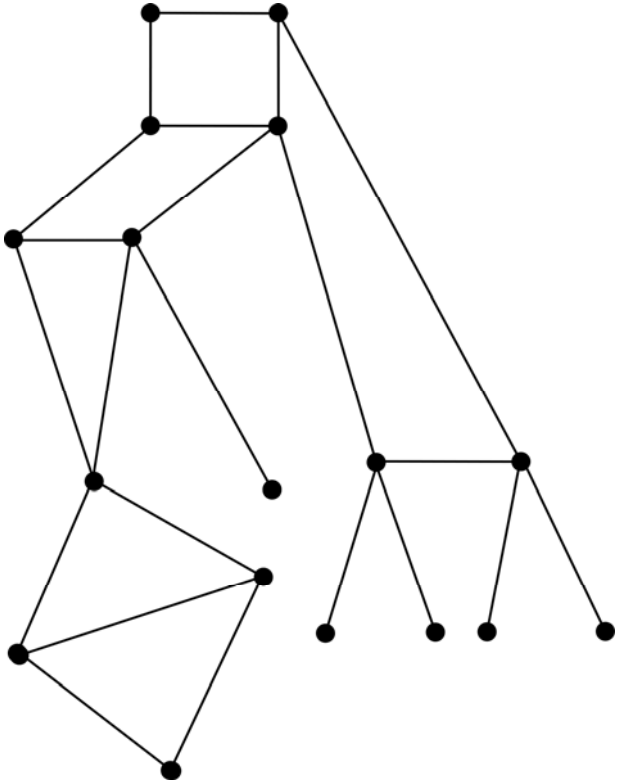
e = ***r*** ●——————● ***n***

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

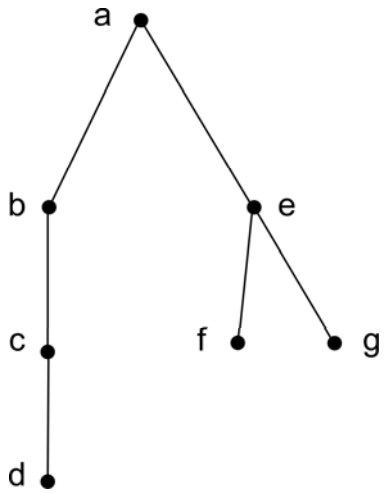


Trees are defined by : T = T // T ∪ extension(T) ∪ ***r***

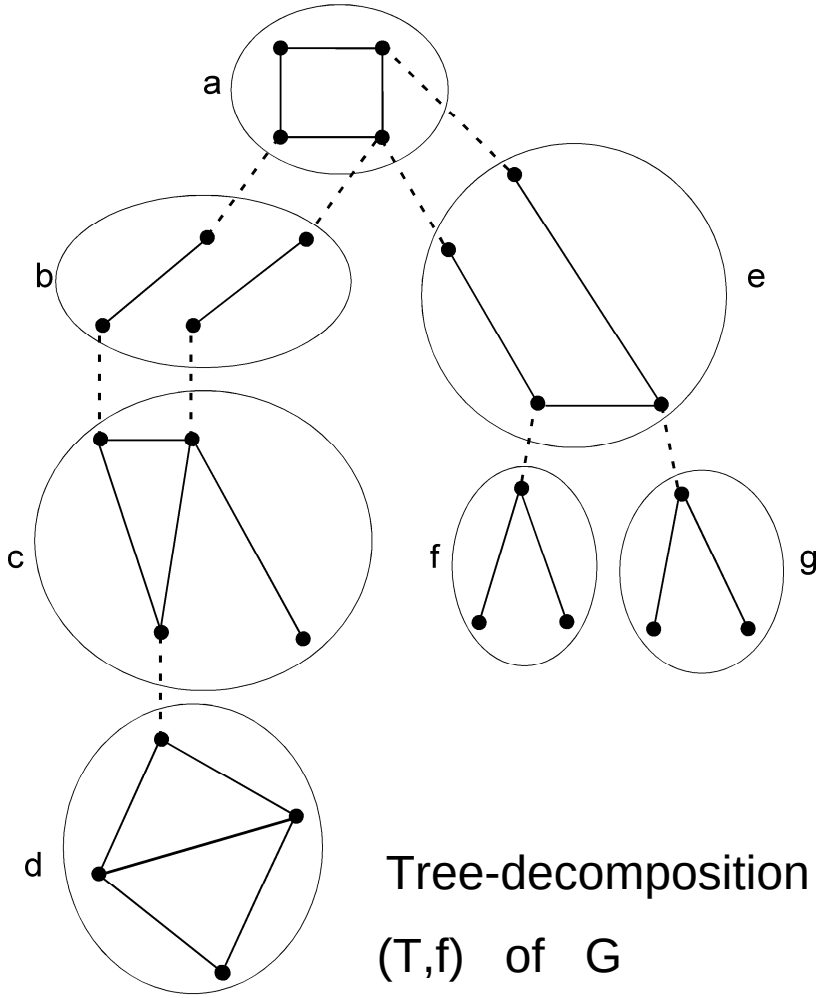
Relation to tree-decompositions and tree-width



Graph G



Tree T



Tree-decomposition (T,f) of G

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

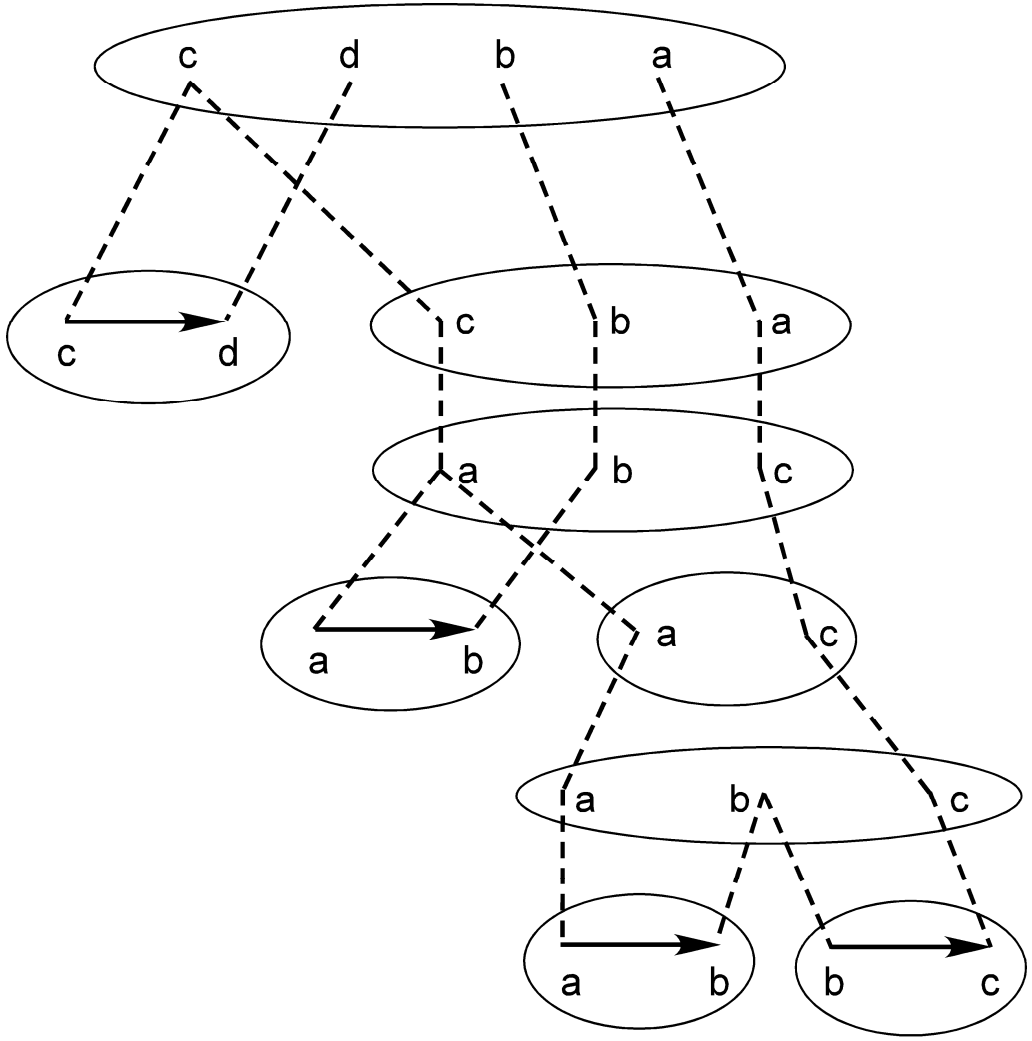
Width = max. size of a box -1. Tree-width = min. width of a tree-dec.

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.