Short vertex labels for connectivity check in planar graphs with forbidden parts

Bruno Courcelle

Université Bordeaux 1, LaBRI and *Institut Universitaire de France*

References : Articles with R. Vanicat (2003), with A. Twigg (STACS 2007), with C. Gavoille, M.M. Kanté, A.T. (TGGT, Paris, 2008), and with C.G., M.M.K. (FAW, Changsha, China, 2008)

See : http://www.labri.fr/perso/courcell/ActSci.html

Labelling Schemes for solving First-Order and Monadic Second-Order Queries

Aim : to check properties of vertices and to compute functions like distance from fixed *short* vertex labels (label = bit sequence).

Short = of length O(log(n)) or $O(\log^2(n))$; n = number of vertices

Wanted : for fixed class of graphs *C* and fixed property or function *F* ,

two algorithms :

Algo *A* : defines for G in *C* a label J(x) for each x in V = V(G).

Algo *B* : computes *F*(u,v,X,Y) from J(u), J(v), J(X), J(Y)

for vertices u, v, sets of vertices X,Y in some G in *C*

Idea : all necessary information from G is distributed on vertices;

computing *F* need not process the graph (this has been done by *A*).

Results

Adjacency (implicit representation) with labels of size $O(\log(n))$:

Bounded arboricity (includes planar, bounded tree-width)
Bounded clique-width,
Interval graphs (unbounded clique-width and arboricity)

Distance Static : all $\Theta(n)$

trees, bdd tree-width, clique-width $\Theta(\log^2(n))$
planar between $O(n^{1/3})$ and $O(n^{1/2})$
interval graphs $O(\log(n))$

Distance Dynamic : obstacles (forbidden parts, specified in query)

to be computed : $d(u,v,X,F)$, the distance of u and v in $(G-F)\backslash X$

X : forbidden vertices, F : forbidden edges.

Clique-width $\leq$ k : $O(k^2.\log^2(n))$ (B.C., A. Twigg, stacs07)

Connectivity labelling (dynamic)

Clique-width ≤ k : $O(k^2.\log(n))$ (B.C., A. Twigg, stacs07)

Planar graphs with obstacles :$O(\log(n))$(B.C.,C.Gavoille, M.Kanté,A.T.08)

General logical approach

Monadic second-order properties (optimization or counting functions)

Clique-width ≤ k : $O(f(k).\log(n))$ ($O(f(k).\log^2(n))$) (B.C., R. Vanicat 03)

First-order properties and counting functions :

Classes of graphs "nicely decomposable",
of locally bounded tree-width or clique-width :
$O(\log(n))$ or $O(\log^2(n))$ (B.C.,C.Gavoille,M.Kanté08)

Tools (in red : this talk)

Graph structure properties : unions of forests, (*balanced*) tree-decompositions and clique-width expressions,

Coverings by families of subgraphs of bounded clique-width with limited overlapping.

Straightline planar embeddings (De Fraysseix *et al.*, Schnyder)

Decompositions in 3-connected components of planar graphs.

Monadic second-order formulas on terms translated into finite automata (Doner, Thatcher-Wright).

First-order formulas decomposed into local and "connected" formulas (Gaifman, Frick)

Bounded arboricity

G is the union of k edge-disjoint rooted forests $F_1, \ldots, F_k$

f_i is the (partial) *father* function in forest F_i

We define $J(x) = (x, f_1(x), \ldots, f_k(x))$ of size $\leq (k+1).\ \lceil \log(n) \rceil$

(vertices are numbered from 1, and denoted by *binary bit sequences*)

Adjacency check :

u and v are adjacent if and only if :

$u = f_i(v)$ or $v = f_i(u)$ for some i.

Monadic Second-Order (MS) Logic

= First-order logic on *power-set* structures

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

MS properties : transitive closure, properties of paths, connectivity,
planarity (via Kuratowski, uses connectivity), k-colorability.

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

Non connectivity :

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

2-colorability (i.e. G is bipartite) :

$\exists X \, (\, \forall u,v \, (u \in X \;\&\; edg(u,v) \Rightarrow v \notin X) \;\&\; \forall u,v \, (u \notin X \;\&\; edg(u,v) \Rightarrow v \in X) \,)$

1. Short labels for MS definable queries

(C&V = B.C. & R.Vanicat, Discrete Applied Maths, 2003)

Theorem : 1) Given k and a monadic second-order graph property $P(X_1,\ldots,X_m)$:

For every graph G defined by a clique-width expression of width k,

one can define a label J(x) for each vertex x of G such that,

from the sets of labels $\{J(y) / y \text{ in set of vertices } A_i\}$, $i=1,\ldots,m$,

one can determine if $P(A_1, \ldots,A_m)$ is true.

Size of J(x) : $O(\log(n))$

Preprocessing time : $O(n.\log(n))$

Answer to query : $O(a.\log(n))$ where $a = |A_1 \cup \ldots \cup A_m|$

2) For MS optimization functions (like distance) or counting functions (number of tuples of vertices $b_1,\ldots,b_q$ that satisfy $P(b_1,\ldots,b_q,A_1,\ldots,A_m)$ for given $A_1,\ldots,A_m$), we replace log(n) by $\log^2(n)$

2. Short labels for connectivity check in planar graphs with obstacles.

(B.C., C.Gavoille, M.Kanté, A.Twigg 2008)

Question is : are u and v connected in (G-F) \ X ?

Method :

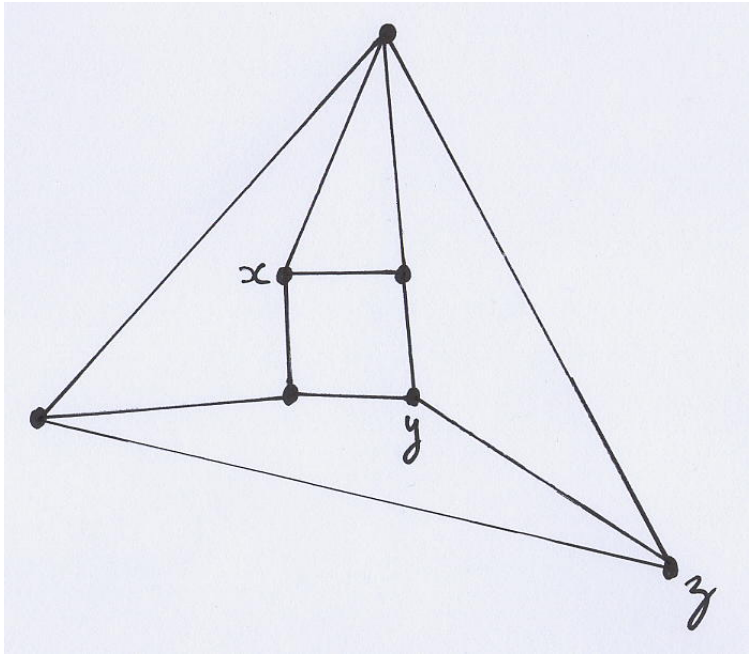
(1) We treat 3-connected planar graphs and their edge subdivisions (new vertices inserted on edges).

(2) Then 2-connected planar graphs decomposed as trees of 3-connected components.

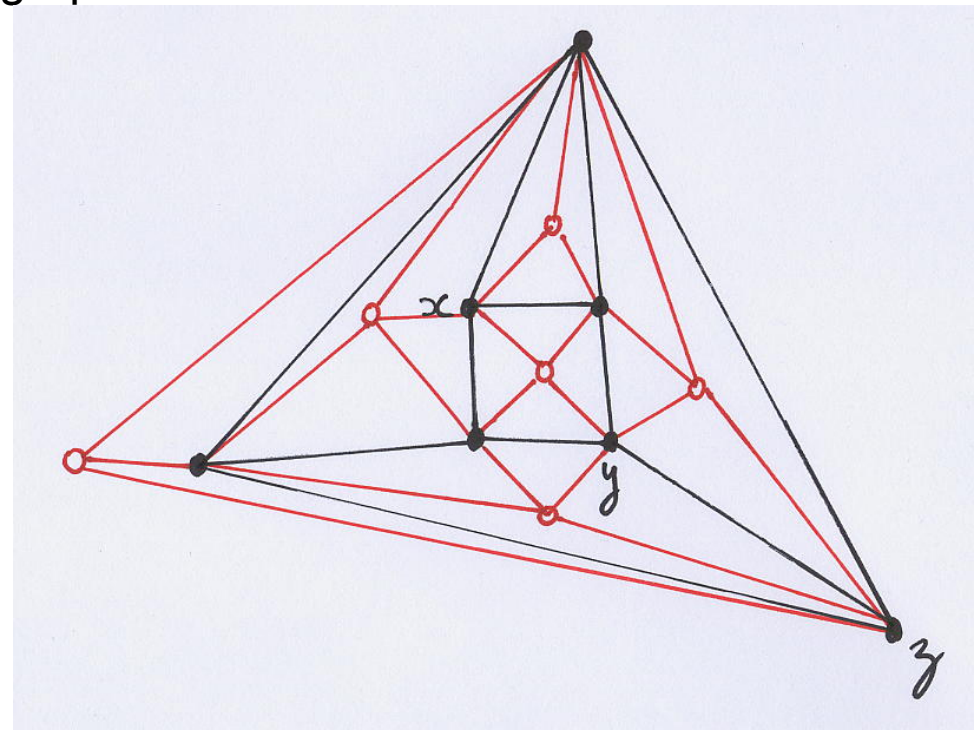
(3) Then connected planar graphs decomposed as trees of 2-connected components.

For case (1) we use a geometric method. For cases (2) and (3) we use a labelling (based on C&V, cf. **1.**) for querying the decomposition trees, combined with labellings of type (1) for the 3-connected components

(1) The case of 3-connected planar graphs

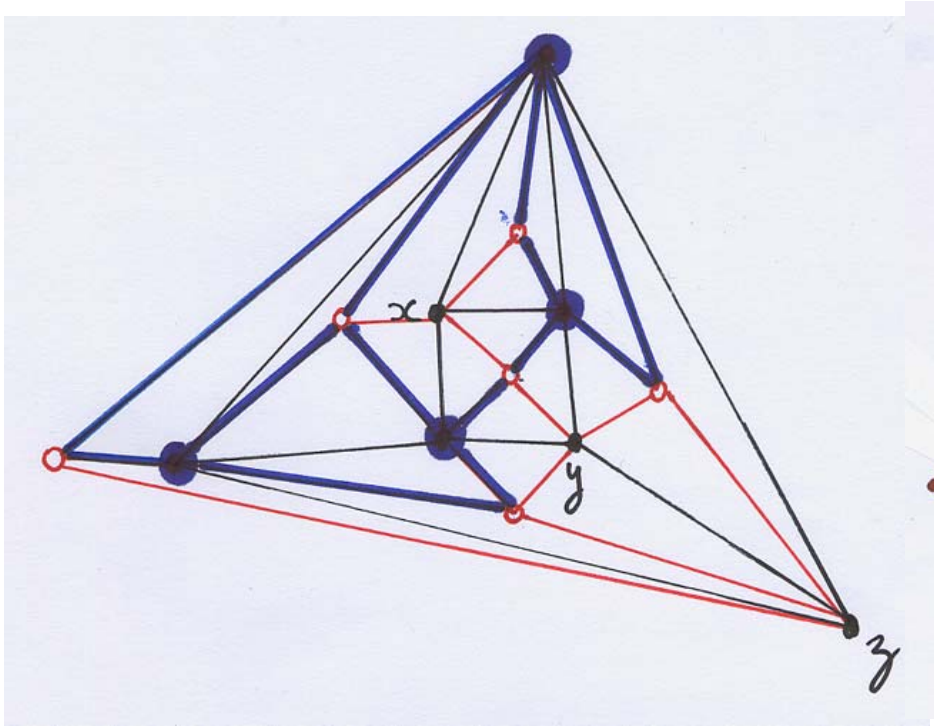


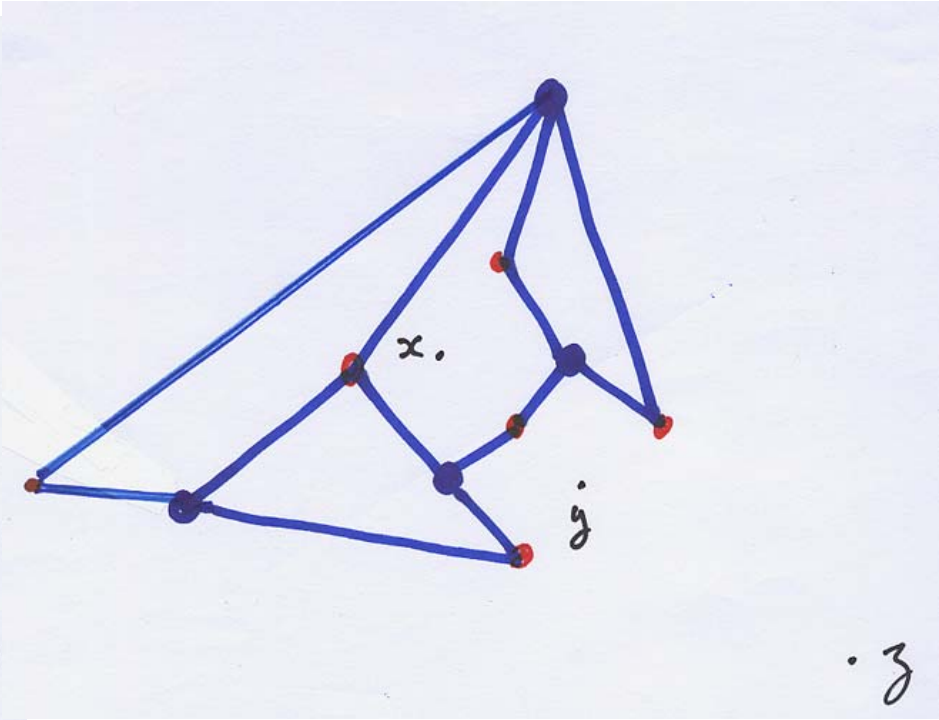
A graph G



Its *augmented* graph G^+

= G with *face to vertex edges*





X = the set of 4 big blue vertices.

Its *barrier* Bar(X) is the set of **thick blue** edges (consists of all u--f--v where u,v in X and f is a face-vertex)

Deleting X separates x and y but not y and z.

The barrier separates *topologically* x and y, but not y and z

To be done :

1) Construct a straightline embedding of G^+ with integer coordinates of maximum value O(n). This is possible since G^+ is simple, because G is 2-connected. (dF, Sch)

2) From labels of vertices in X, we want to determine the coordinates of the end vertices of the *line segments* of the edges of Bar(X).

3) Using a *computational geometry* algorithm, we can test whether two vertices u, v given by their coordinates, are separated in the plane by Bar(X).

For 2) since G^+ is planar, it is the union of 3 forests. One uses an adjacency labelling for G^+, from which, for any two vertices u,v, one can obtain the at most two faces f and h to which they are both adjacent, as values of $g_i(u)$ or $g_i(v)$ for some i = 1,…,30 where $g_1,\ldots,g_{30}$ is a finite list of partial functions. (Which i's give the faces incident with u and v depends on tests of the form "$g_j(u) = g_k(v)$?").

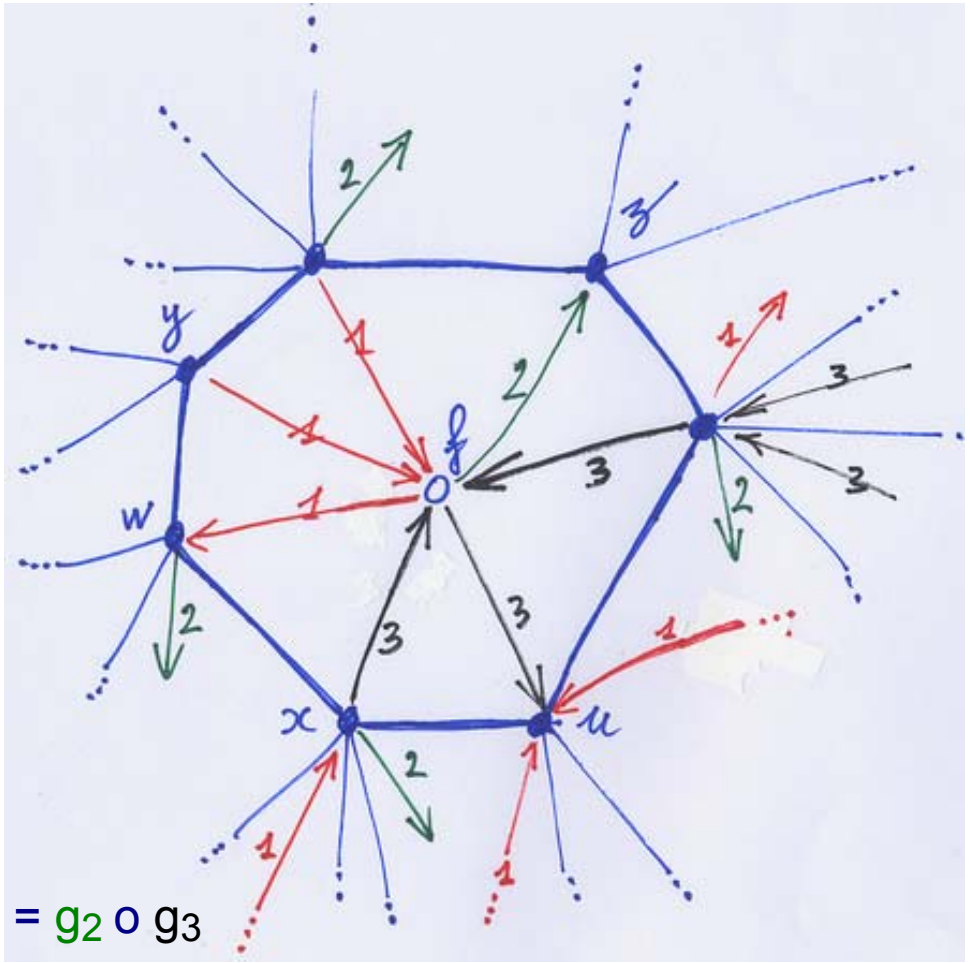
Labels : D(x) = (C(x), $C(g_1(x)),\ldots,C(g_{30}(x))$) where C(u) = integer coordinates of u.

Constructing these functions.

We cover G^+ with 3 forests : red (**1**), green (**2**) and black (**3**) ;

"father functions" are g_1, g_2, g_3 .

That 2 vertices belong to face f is described by 3 cases :



(1) $g_3(x) = g_1(y) = f$

(2) $g_{2,3}(x) = z$ and $g_2(x) = f$ where $g_{2,3} = g_2 \circ g_3$

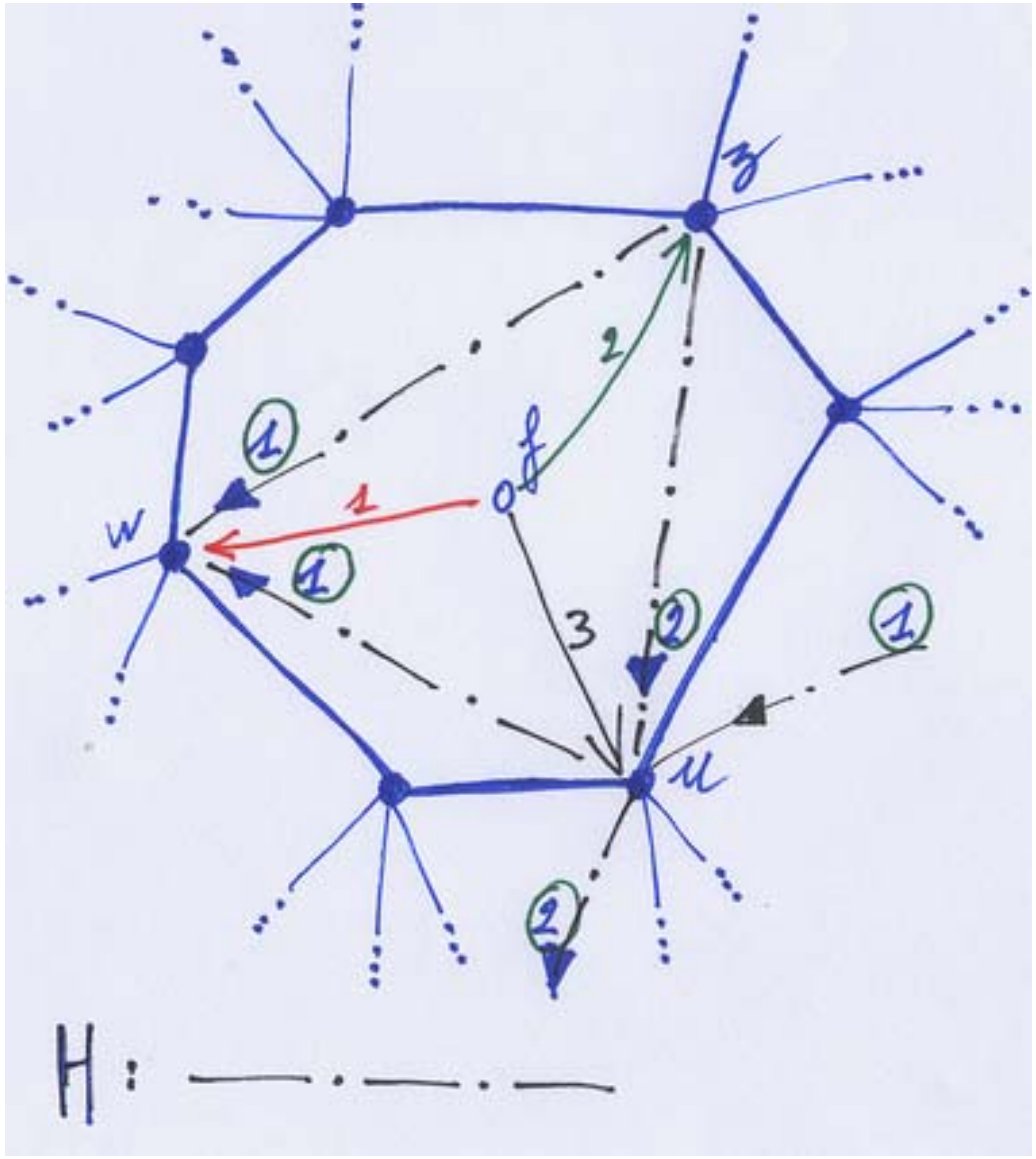
(3) $g_1(f) = w$ and $g_2(f) = z$: more difficult : no function can invert g_1, g_2, g_3 .

We construct another planar graph H by putting a triangle or an edge in certain faces.

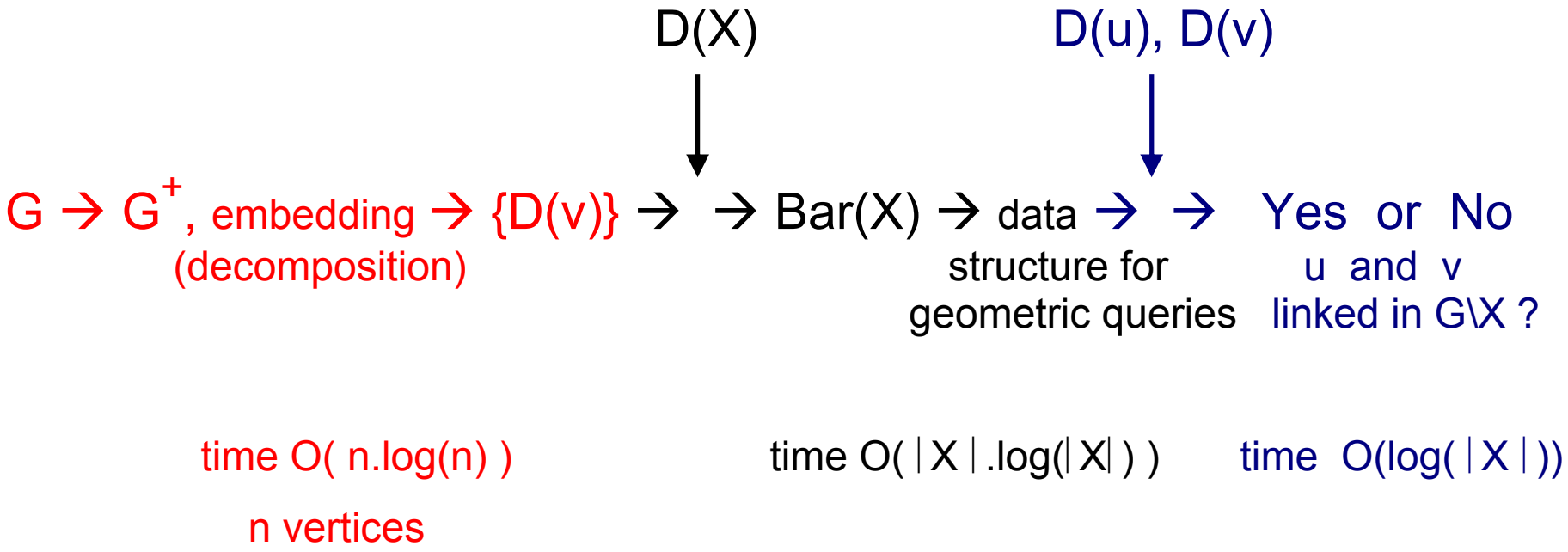
We cover H with 3 forests, giving 3 functions h_1, h_2, h_3 (circled numbers in green).

Then

$h_1(z) = w$ indicates that z and w belong to a same face ; this face is determined by 6 other unary functions.



Overview of the algorithm(s) :



First we process G and define label D(v) for each vertex v.

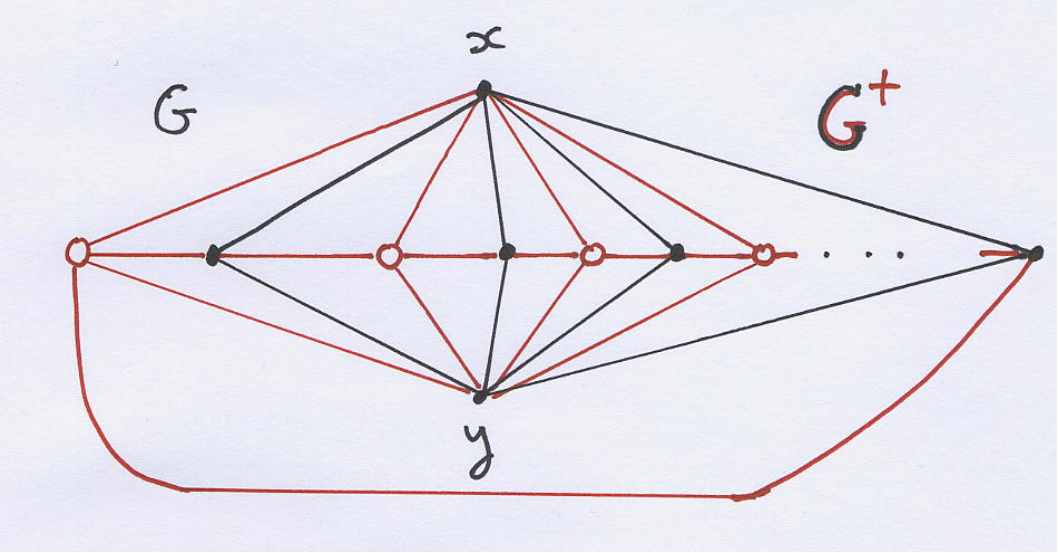
Then we process a set of vertices X given by D(X) = set of D(x), x in X.

Then connectivity queries for various u,v, and the fixed set X.

General case : Two difficulties

1st : If G is not 2-connected, G^+ is not simple and has no straightline embedding.

2nd : If G is 2-connected but not 3-connected :



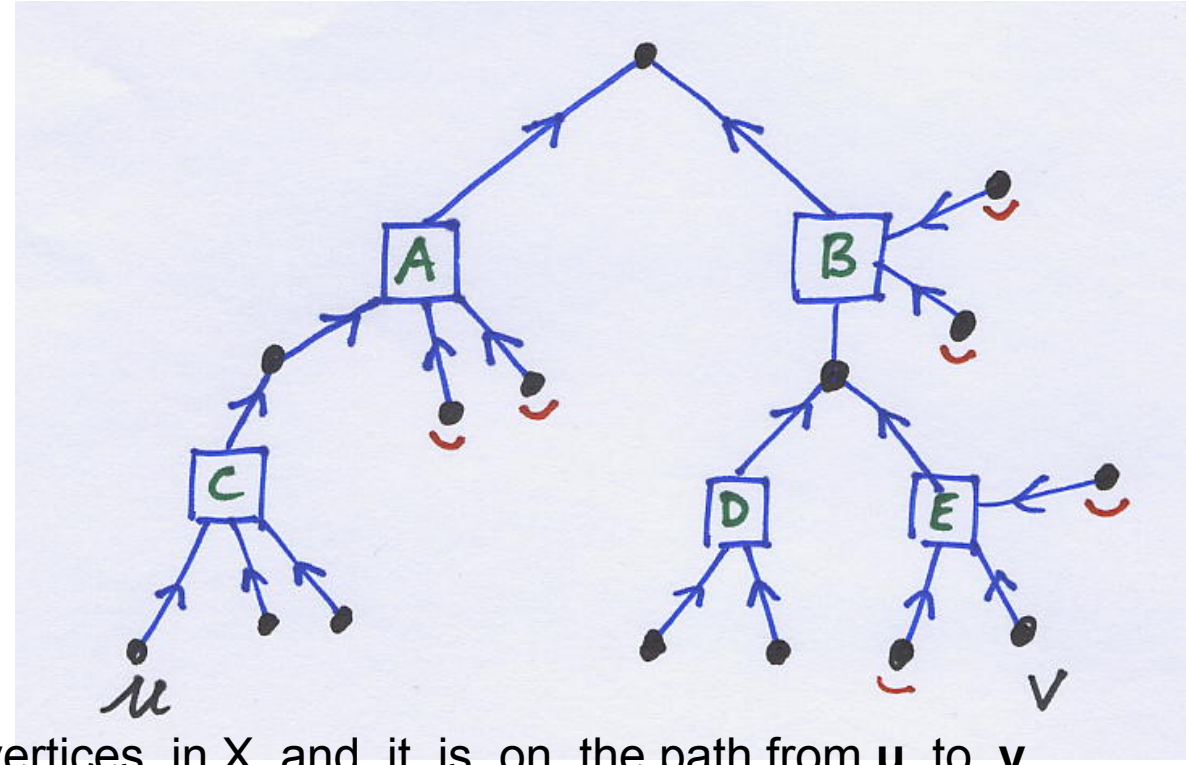
x and **y** are incident with an unbounded number of faces.

One cannot specify all of them with a fixed number of functions ; one cannot find the coordinates of all segments (edges) in Bar({**x,y**}).

Tool 1 : The (auxiliary) tree of 2-connected components.

Its nodes are the vertices of G and nodes representing the 2-connected components (A,B,C,…). Adjacency = membership of a vertex in a 2-connected component.

Fact : X separates **u** and **v** in the graph G **if** **u** and **v** are separated by X in the tree **or if** they are separated in G by X∩B, where B is a problematic biconnected component.



Problematic means : B has ≥ 2 vertices in X and it is on the path from **u** to **v**.

Example : A,B,E are problematic.

By C&V's technique, a *log*-labelling K of this tree can be built from which :

(a) one can detect if the first case holds

(**u**, **v** are separated by a separating vertex of G that is in X)

and

(b) if it does not, one gets the problematic 2-connected components relative to X,**u**,**v** : they *may* separate **u** and **v**. For each of them, the *geometric method* based on labelling D, can be used.

The label of x is defined as (D(x), K(x)) of size O(log(n)). (Omitting details).

Remark : We define D from a straightline embedding of Simple(G^+), the graph obtained from G^+ by fusing parallel edges. We *do not claim* that we can combine "independent" *log*-labellings for the biconnected components into a single *log*-labelling.

Tool 2 : Decomposition of 2-connected planar graphs in “3-blocks”

(3-connected components, cycles and “bonds”, sets of parallel edges).

Method : We replace the barrier by a *reduced barrier* *RBar*(X) :

if **x** and **y** in X are incident with at least 3 faces, $f_1,\ldots,f_k$ we put in *RBar*(X) *only* **x**— f_1 and **y**— f_1.

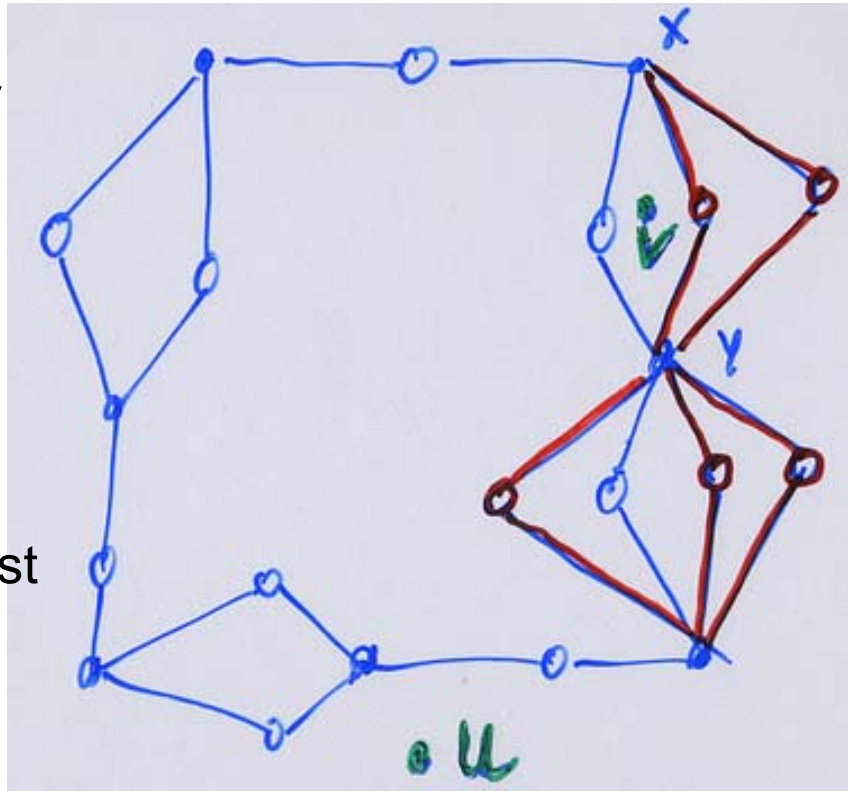
Problem : The reduced barrier *RBar*(X) will *miss some cases* where the given vertices are separated by the “full barrier” *Bar*(X).

These cases will be detected on the *tree* representing the decomposition of the graph in 3-blocks, by means of a labelling of this tree with C&V’s theorem and several additional nice (?) tricks.

Proposition : **u** and **v** are separated by Bar(X) *iff* either they are separated by RBar(X) or P(**u**,**v**,x,y) holds for some x,y in X,

where P(**u**,**v**,x,y) means :

x and y are incident with at least 3 faces and {x,y} separates **u** and **v** in the graph.



Proof: Let **u**,**v** be separated by Bar(X) and *not* by RBar(X) with X minimal for inclusion. On figure, **o** are vertices representing faces. Edges —**o**— are those in Bar(X) not in RBar(X). Hence, one of **u**,**v** (**v** on figure) is inside an open set of $\mathbf{R}^2$ –Bar(X) with frontier vertices x and y, incident with at least 3 faces. These vertices separate **u** and **v** in G.

Next goal : a *log*-labelling for checking P(**u**,**v**,x,y) , built on the tree T of 3-connected components.

Bipolar graphs : Directed, acyclic ; every vertex is on a path from the *South Pole* s(G) to the *North Pole* n(G).

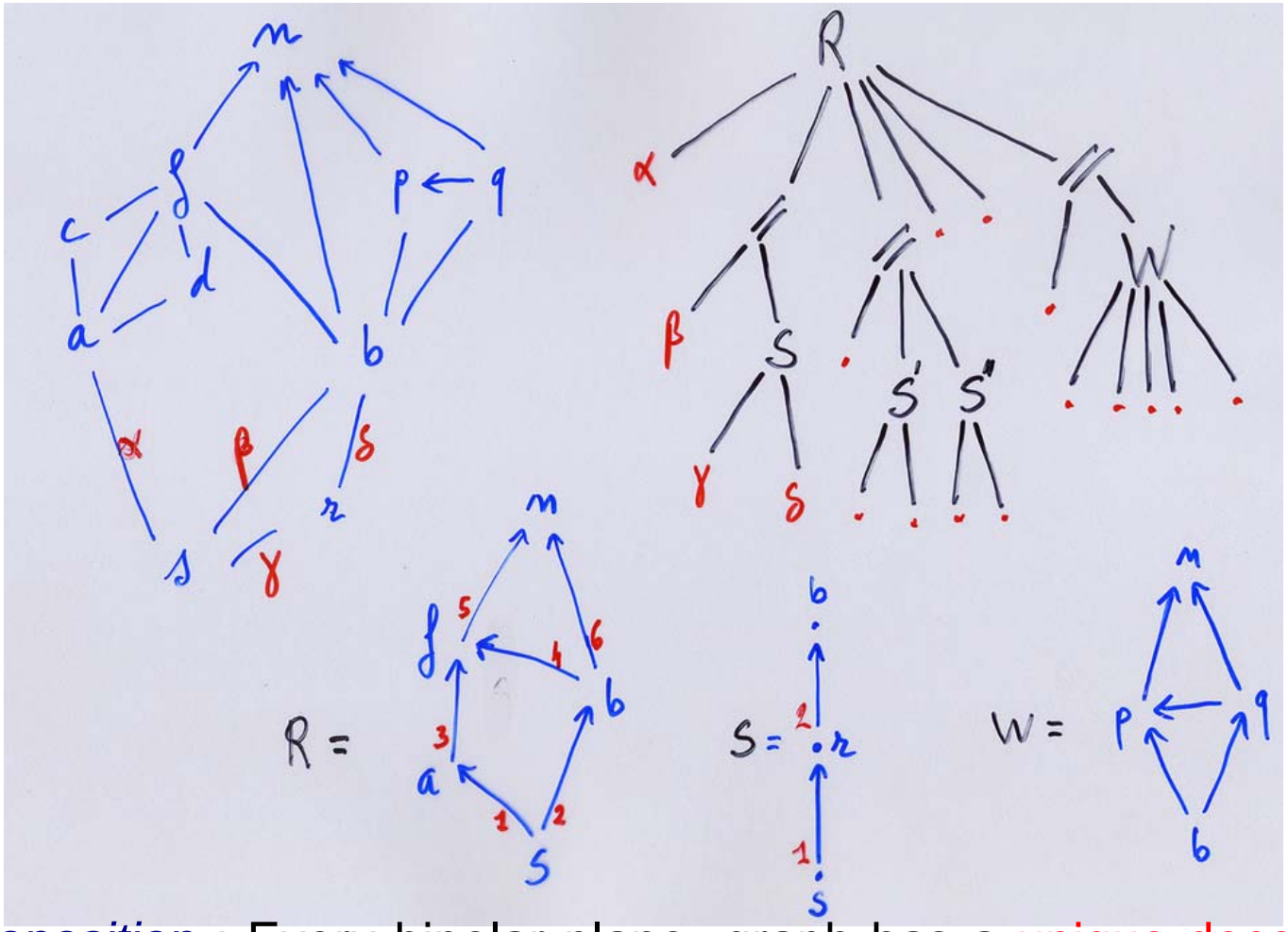
Every 2-connected has a bipolar orientation with adjacent poles.

Operations on bipolar *plane* graphs :

Parallel composition : G // H : glues two (disjoint) *plane* graphs G and H by their poles (it is *not* commutative).

Substitution : $R(G_1,\ldots,G_m)$: substitution of $G_1,\ldots,G_m$ to the directed edges of a simple bipolar *plane* graph R.

G



Tree T

Proposition : Every bipolar plane graph has a unique decomposition in terms of parallel composition, substitutions and edges e = s ⟶ n.

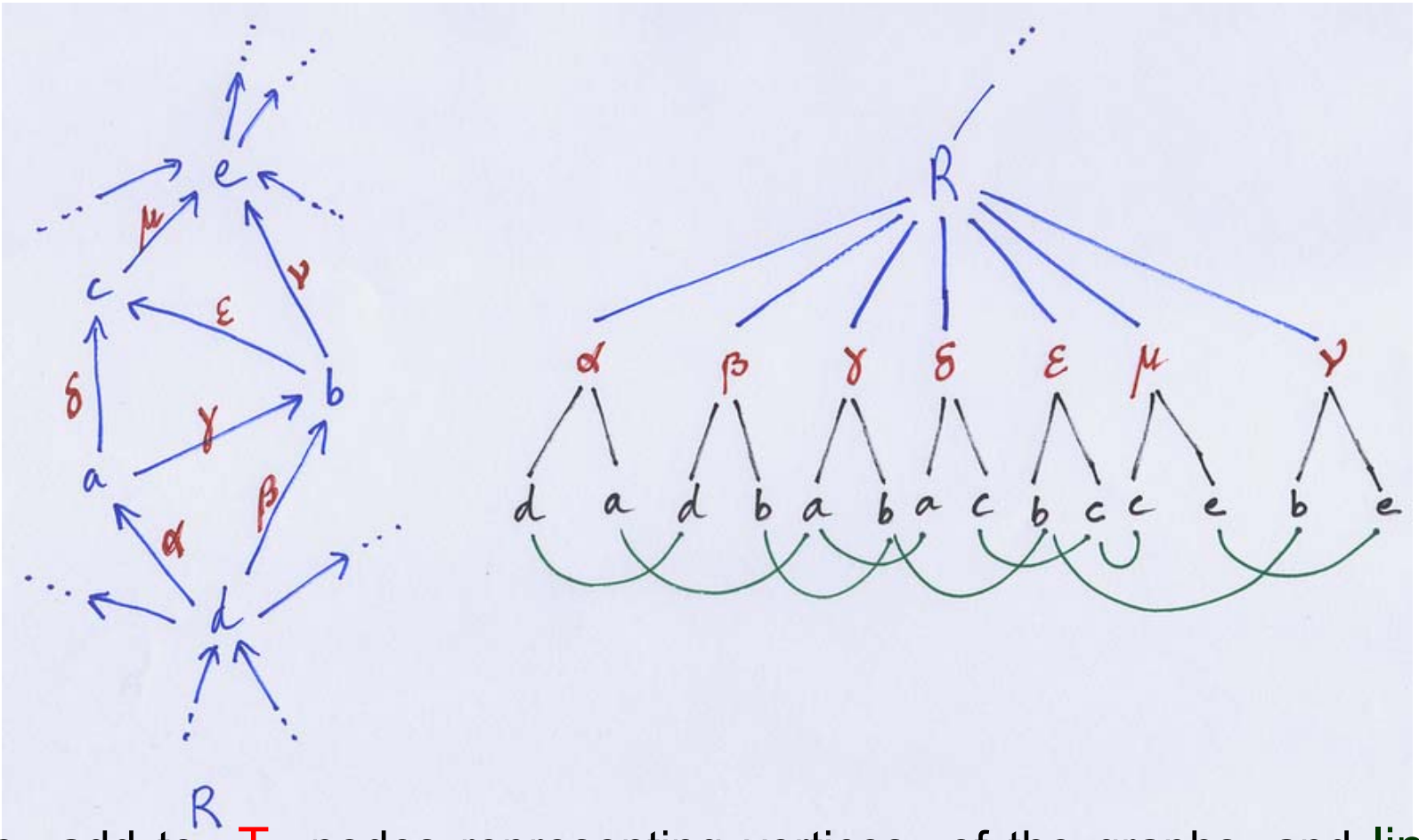
Definition : Two vertices that are the poles of a subgraph defined by subterm with root label // form a *polar pair*.

Example above : (s,b), (b,n), (a,f)

Proposition: Let G be plane bipolar, G = e // H. If two vertices of G belong to more than 2 faces, they form a *polar pair*.

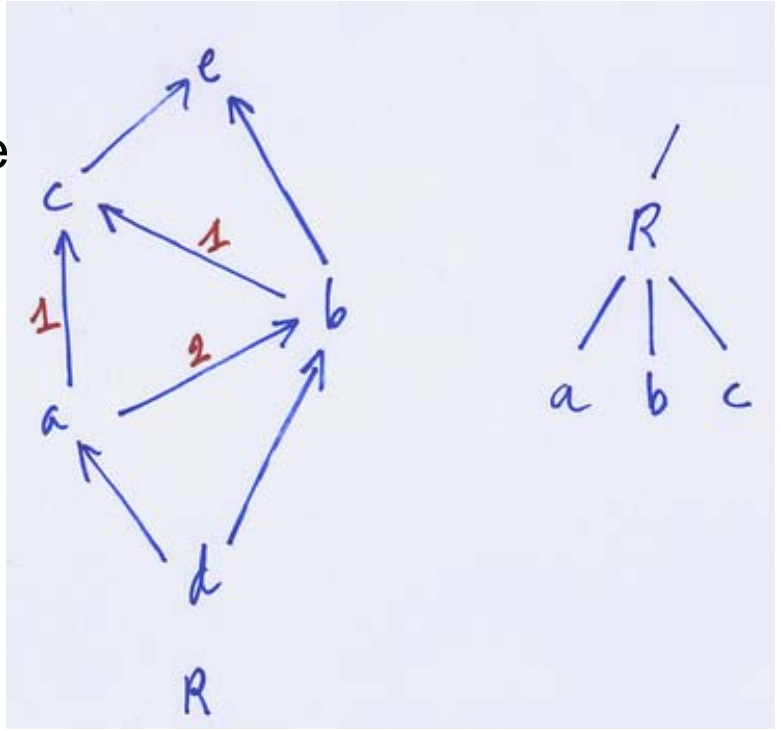
Fact : If **u** is a vertex of an edge *below* a //-node w with polar pair (x,y), and **v** is a vertex of an edge *not below* w, then {x,y} separates **u** and **v** .

We want a labelling that checks this fact, called property P(**u**,**v**,x,y), based on the decomposition tree T.



We add to T nodes representing vertices of the graphs, and links between them representing identical vertices. *Problem*: this graph has unbounded clique-width, hence C&V's result does not work.

Trick : We keep in T* only one node for each vertex ; edges are defined by unary functions :



$g_s(a) = g_s(b) = d$, (towards South)

$g_n(c) = g_n(b) = e$, (towards North)

$g_1(a) = g_1(b) = c$, $g_2(a) = b$

8 functions suffice for all graphs R (because of planarity).

Fact : P(**u**,**v**,x,y) holds ***iff***

T+green edges $\models \varphi(\mathbf{u},\mathbf{v},x,y)$ ***iff***

$T^* \models \psi(\mathbf{u},\mathbf{v},x,g_s(x), g_n(x), g_1(x),\ldots,g_6(x),y, g_s(y), g_n(y), g_1(y),\ldots,g_6(y))$

where $\psi(\mathbf{u},\mathbf{v},x,x_n,\ldots,x_6,y, y_n,\ldots,x_6)$ is a monadic second-order formula.

For this ψ , and for each T^* we construct (with C&V's result) a labelling L ; for every vertex x of G we define :

$$K(x) = (L(x), L(g_s(x)),L(g_n(x)), L(g_1(x)),\ldots,L(g_6(x)).$$

Hence given K(**u**), K(**v**), K(x), K(y) we can check P(**u**,**v**,x,y) on T^* because we get from K(x) and K(y) the necessary informations :

$L(\mathbf{u})$, $L(\mathbf{v})$, $L(g_s(x)),L(g_n(x)), L(g_1(x)),\ldots,L(g_6(x))$

and $L(g_s(y)),L(g_n(y)), L(g_1(y)),\ldots,L(g_6(y))$,

The final labelling of each vertex x of G combines :

the above K(x),

the label for querying the tree of biconnected components,

the integer coordinates of x and of 30 vertices of G^+

(at distance at most 2 of x).

Summary (omitting some details) :

Labelling algorithm A

Input : a simple planar connected graph G

1. Choose a root vertex and build the tree **B** of 2-connected components C.
2. In each component C, let n(C) be maximal in that tree, let s(C) be adjacent to n(C) in C and make C bipolar with poles s(C) and n(C).
3. Decompose C and determine the 8 functions g_s, g_n, g_1,… describing the internal structure of the blocks R.
4. Build the corresponding trees T*(C) and combine them with tree **B** into a single tree **B**T*(G).

5. Build for this tree a *log*-labelling for checking the separation properties in the trees **B** and $T^*(C)$.
6. Define the graph G^- equal to G^+ with multiple edges fused.

   Ask Schnyder to embed it in the plane with straightline segments and small integer coordinates.
7. Compute the 30 unary functions that will help to construct the reduced barriers RBar(X).
8. Set vertex label J(x) for each x, by using steps 5-7.

Decoding algorithm B

Input : J(**u**), J(**v**), J(X).

a. Check for each x and each pair (x,y) such that $x,y \in X$ if **u** and **v** are separated by {x} or by {x,y} (we use here the tree **B**$T^*(G)$). If one is found, then stop and report that X separates **u** and **v**.

b. If no such separation is found, determine the sets $X \cap Vertex(C)$ for all *problematic* 2-connected components C, and their vertices y_C, z_C that are separating in G and are on the path in **B** between **u** and **v**.

c. For each such C build RBar($X \cap Vertex(C)$) from the straightline embedding of G^- and check whether Conn(y_C, z_C, $X \cap Vertex(C)$).

d. If all answers are positive, then report that Conn(**u**,**v**,X) holds. Otherwise that it fails.

Computation times :

n = number of vertices.

Algorithm A : O(n) for graph decompositions ;

O(n.log(n)) for computing the labels of tree $BT^*(G)$.

O(log(n)) for defining each label J(x).

Total : O(n.log(n)).

Algorithm B : N = number of vertices in X (not = 0)

O(N.log(n)) for checking separations ;

O(N.log(N).log(n)) if checking topological separations

(reduced barriers in the plane) is needed .

Total : O(N.log(N).log(n)).

Extension 1 : deleting edges and adding new links

X : N deleted vertices,

F : deleted edges, handled as degree 2 vertices in a subdivided graph

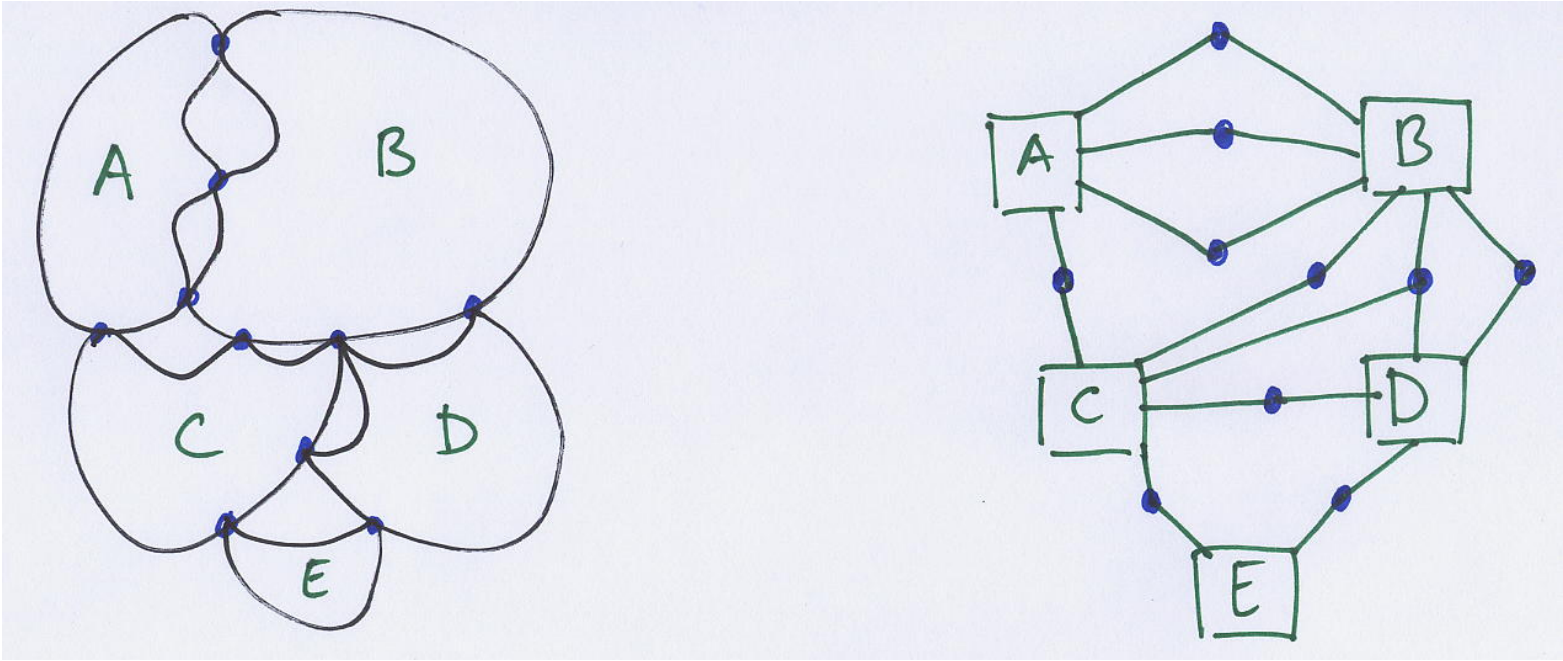
H : h new links between pairs of vertices (repairing the network).

Query : Are **u** and **v** connected by a path in $((G - F) \setminus X) + H$?

The data structure is built for X and F

Query takes time $O(N.h^2)$

Extension 2 : Graph covers with limited overlaps : Combinations of labelling schemes



Connected blocks with *log*-labelling

Skeleton graph

bipartite, degree $\leq$ d

log-labelling

The two labelling schemes can be combined into a single *log*-labelling

Open questions :

Graphs on nonplanar surfaces

How hard is it to update the labels if one adds vertices and edges ? (deletions are handled by set arguments included in queries).

Reachability in *directed* planar graphs with obstacles.

log^2-labelling for *distance* in *directed* planar graphs with obstacles.

(The geometric method does not extend obviously to distances).

Which properties (generalizing adjacency, “same face”, etc…) can be represented by k unary functions ?

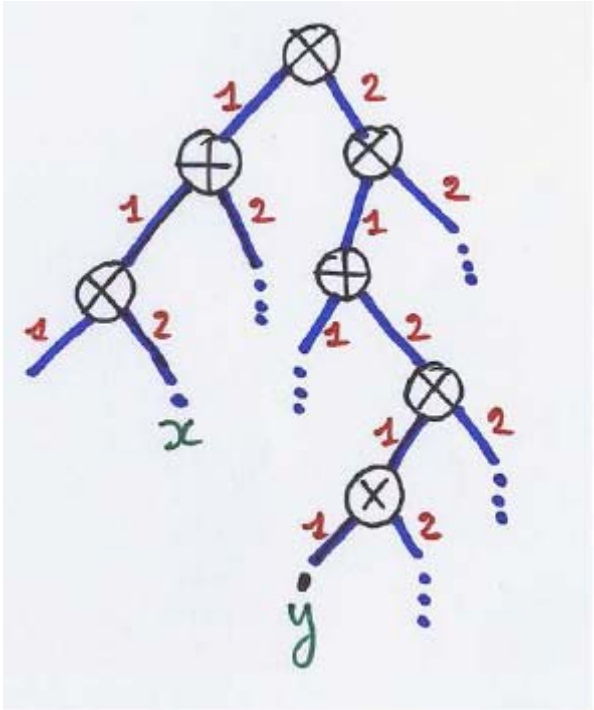
Appendix : Definitions and previous results

Cographs

Built from vertices with two binary operations :

$\oplus$: disjoint union

$\otimes$: complete join = disjoint union plus all edges between the two arguments



Adjacency labels are words (branches)

J(x) = $\otimes 1 \oplus 1 \otimes 2$

J(y) = $\otimes 2 \otimes 1 \oplus 2 \otimes 1 \otimes 1$

The least common ancestor of x and y is labelled by $\otimes$, hence they are adjacent. The size of J(u) is not O(log(n)). By using other graph operations one makes terms balanced i.e., of height O(log(n)).

Graph operations defining Clique-width

Clique-width has no combinatorial characterization but is defined in terms of few very simple graph operations (giving easy inductive proofs).

Equivalent notion: rank-width (Oum and Seymour) with better structural and algorithmic properties.

Graphs are simple, directed or not.

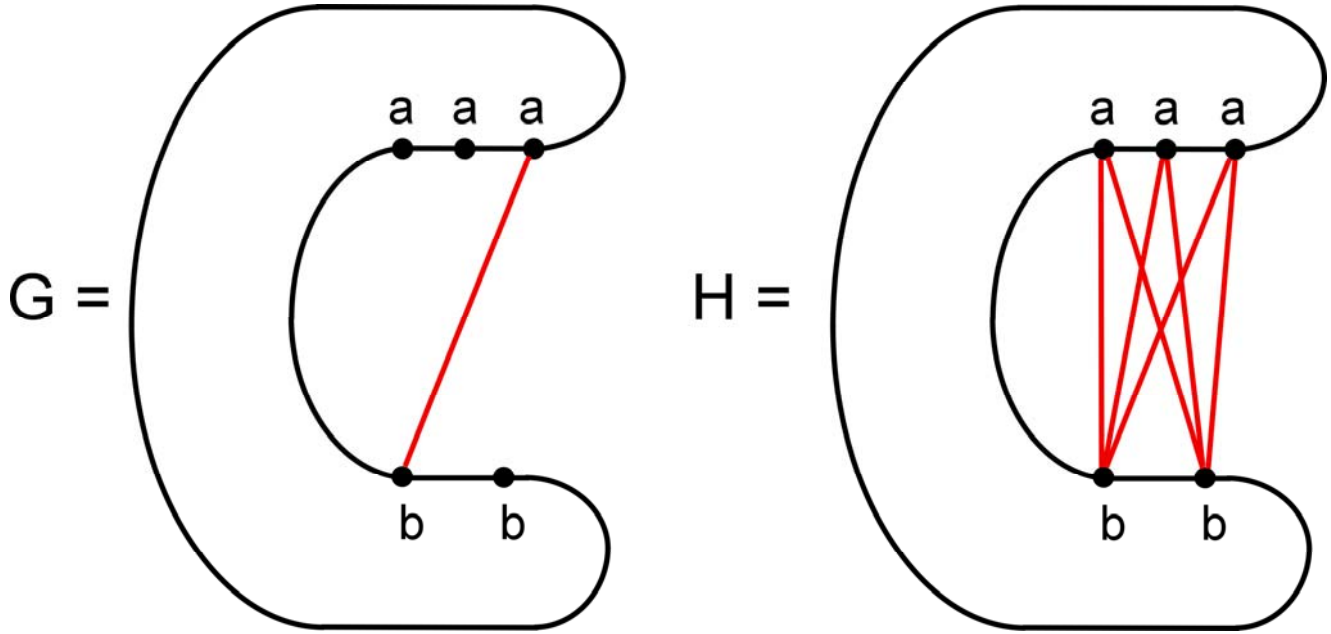
k labels : *a* , *b* , *c*, ..., *h*. Each vertex has one and only one label ;

a label *p* may label several vertices, called the *p-ports*.

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

Unary operations: Edge addition denoted by $Add\text{-}edg_{a,b}$

$Add\text{-}edg_{a,b}$(G) is G augmented with (un)directed edges from every *a*-port to every *b*-port.



H = $Add\text{-}edg_{a,b}$(G) ; only 5 new edges added

The number of added edges depends on the argument graph.

Vertex relabellings :

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

Basic graphs are 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 $\oplus$, $Add\text{-}edg_{a,b}$ and $Relab_{a \longrightarrow b}$ with k labels. Its clique-width *cwd*(G) is the smallest such k.

Proposition : (1) If a set of simple graphs has bounded tree-width, it has bounded clique-width, but not *vice-versa*.

(2) Unlike tree-width, clique-width is sensible to edge directions: Cliques have clique-width 2, tournaments have unbounded clique-width.

Classes of unbounded tree-width and bounded clique-width.

Cliques (2),

Complete bipartite graphs (2),

Distance hereditary graphs (3),

Graphs without P_5 and $\mathbf{1} \otimes P_4$ (5), or $\mathbf{1} \oplus P_4$ and $\mathbf{1} \otimes P_4$ (16) as induced subgraphs. (many similar results for exclusion of induced subgraphs with 4 and 5 vertices).

Classes of unbounded clique-width :

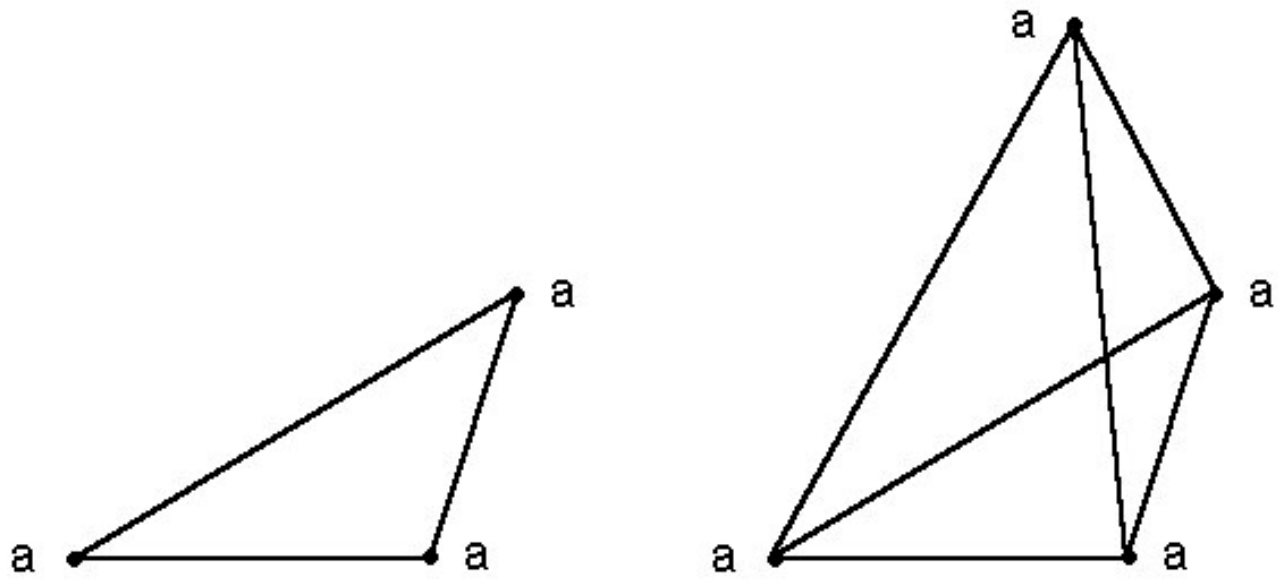
Planar graphs of degree 3,

Tournaments,

Interval graphs,

Graphs without induced P_5.

Example : Cliques have clique-width 2.



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

Another example : Cographs are generated by $\oplus$ and $\otimes$ defined by :

$G \otimes H = Relab_{b \to a}(Add\text{-}edg_{a,b} (G \oplus Relab_{a \to b}(H))$

$= G \oplus H$ with "all edges" between G and H.

Proposition : (a) Deciding "Clique-width $\leq$ 3" is a polynomial problem. (Habib et al.)

(b) The complexity (polynomial or NP-complete) of "Clique-width = 4" is unknown.

(c) It is NP-complete to decide for given k and G if cwd(G) $\leq$ k. (Fellows et al.)

(d) There exists a cubic approximation algorithm that for given k and G answers (correctly) :

either that cwd(G) >k,

or produces a clique-width term using 2^{2k+1} labels. (Hlineny and Oum 2007)

This yields Fixed Parameter Cubic algorithms for many hard problems (MS property, (ex. 3-colorability), MS optimization function, (ex. distance), MS counting function, (ex. # of paths).

A log-labelling for MSO properties on graphs of bounded clique-width (C&V)

Basic result : the case of terms (instead of graphs).

We consider terms t in T(F,C) (F : binary operations, C : constants),

Every MS formula $\varphi(X_1, \ldots, X_m)$ with free variables denoting sets of occurrences of constants in t can be translated into a deterministic finite automaton *A* for the signature $F \cup Cx\{0,1\}^m$ such that *A* accepts a term **t** in $T(F, Cx\{0,1\}^m)$ iff

$$t \models \varphi(A_1, \ldots, A_m)$$

where **t** is t with the $\{0,1\}^m$ labels attached to occurrences u of constants and

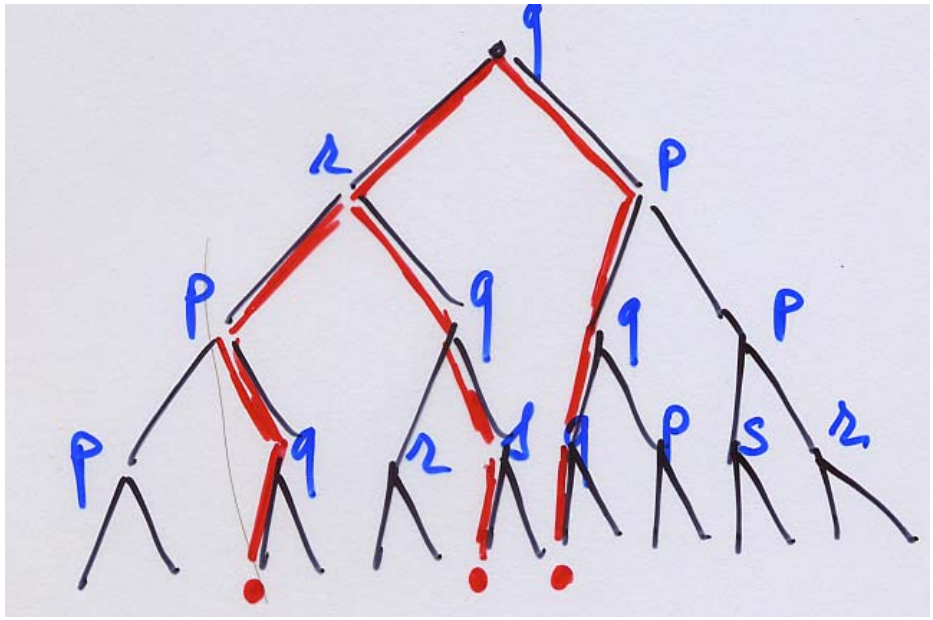
A_i is the set of occurrences u of some (c,w) in $Cx\{0,1\}^m$ such that w[i] = 1.

(*Intuition* : u occurrence of (c,0,1) means that u is in X_2 and not in X_1.
The set of terms accepted by *A* encodes terms and the m-tuples of sets of occurrences of constants (leaves of t as a tree) that satisfy φ in these terms.)

The method for $\varphi(X,Y)$ (m=2).

Assuming *A* constructed and t in T(F,C) be given :

We run *A* on t with each c replaced by (c,0,0) (i.e., for empty sets X,Y), we mark each node with the corresponding states : p,q,r,s, …



If X∪Y is not empty, we modify accordingly some leaves. The new run will only modify the states on the branches from these leaves to the root.

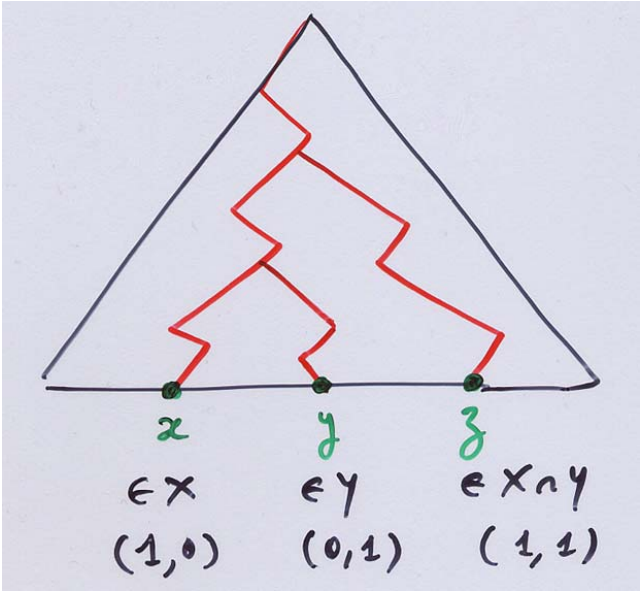
These new states can be obtained from :

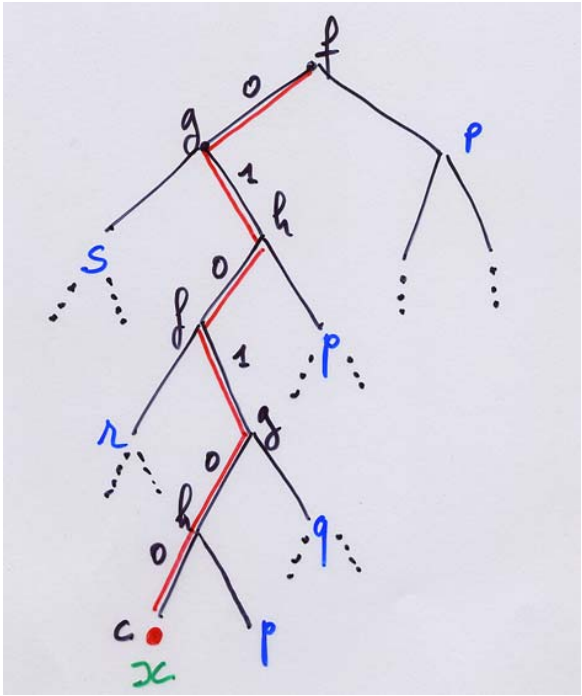
the states on these branches and those at distance 1 of these branches (because the new run is the same on the corresponding subterms).

This information for a branch from **x** can be stored in a word J(**x**) like :

[f,0,p] [g,1,s] [h,0,p] [f,1,r] [g,0,q] [h,0,p]c

of size proportional to the length of the branch (= O(log(n)) for a balanced term with n leaves).





The case of graphs :

For graph G defined as val(t) for a term t in T(F,C) (F, C operations defining clique-width), then V(G) = the set of occurrences of constants in t.

Every MS formula $\varphi(X_1, \ldots, X_m)$ can be translated into an equivalent MS formula $\psi(X_1, \ldots, X_m)$ on the term :

$$G \models \varphi(A_1, \ldots, A_m) \quad \text{iff} \quad t \models \psi(A_1, \ldots, A_m).$$

We apply to ψ the previous construction.

Counting functions : at each node w of t, we store numerical information : for each state p, the number of assignments of 0,1's to the leaves below w that yield state p at this node. This table has size : #-of-states. $\lceil \log(n) \rceil$

Optimization : similar method, the table indicates the maximum value reachable with state p for some assignments of 0,1's.