# Structured graphs and the verification of their monadic second-order properties by means of automata

*Bruno Courcelle* *and* *Irène Durand*

Bordeaux University, LaBRI (CNRS laboratory)

Demonstration of the software TRAG

Written by Irène Durand and Mikhail Raskin

*(outside of this lecture)*

# Summary

First-order model-checking : a review of some algorithmic "meta-theorems".

Monadic second-order model-checking with fly-automata.

FO (*First-order*) and MSO (*monadic second-order*) model-checking:

verification of $G \models \varphi$ for fixed $\varphi$, in terms of the size of G.

This is called "data complexity".

The size of a graph G is the number of vertices.

Graphs are relational structures: the vertices form the domain, binary relations express adjacency or incidence, and unary relations express labellings. The case of graphs capture most difficulties.

Other relational structures can be, to some extent, encoded as labelled graphs.

# First-order model checking.

Typical FO graph properties :

degree $\leq$ d, diameter $\geq$ d (for fixed d).

Connectedness, planarity, k-colorability are not FO. They are MSO (monadic second-order) expressible.

*Time complexity of checking* $G \models \varphi$ for fixed $\varphi$, n = # of vertices.

*For all graphs* : $O(n^s)$, s = the number of quantifiers of $\varphi$.

*Sparse graphs : they have O(n) edges for n vertices.*

Bounded degree : O(n) (Seese, 1996).

Locally bounded tree-width, includes bounded degree, planar,

bounded tree-width : $O(n^{1+\varepsilon})$ (Frick and Grohe, 2004)

Bounded expansion : O(n) (Dvorak et al., 2010)

Nowhere dense : O(n) (Grohe et al., 2014)

Tree-width will be reviewed soon (Trees have tree-width 1).

Locally bounded tree-width means : each ball of radius r has tree-width

$\leq f(r)$ for some function f.

A *ball of radius* r is $N_r(x)$ = induced subgraph of vertices at distance at most r of a vertex x.

Bounded expansion and nowhere dense classes have been defined by Nesetril and Ossona de Mendez in terms of average degree for certain shallow minors.

## The basic tool for FO model-checking

*Gaifman's Theorem:* Every FO sentence (*closed* formula) is equivalent to a Boolean combination of local formulas.

*Local formula* : Conjunction of $\psi[B_i]$, i = 1, …,k, where $B_1, \ldots, B_k$ are pairwise disjoint balls of radius *r* .

For graphs of degree $\leq$ d, there are finitely many balls of radius r, up to isomorphism.

Bounded expansion has a characterization in terms of neighbourhood complexity (Reidl et al. 2016) :

G, a graph, Y a set of vertices and $r \geq 1$.

$\mu_r(Y)$ := the number of sets $N_r(x) \cap Y$, $x \in V$.

A class of graphs $C$ has *bounded expansion* if and only if, for each $r$, there is number $a$ such that $\mu_r(Y) \leq a.|Y|$ for all graphs in $C$, all sets $Y$.

There is a similar characterization for *nowhere dense classes* by Eickmeyer et al. (2017).

It is not surprizing that conditions related to neighbourhoods have consequences for FO verification in view of Gaifman's Theorem.

Proofs are rather technical and involve huge constants.

Bounded expansion and nowhere dense classes do not include all classes of graphs of bounded average degree, (those such that $|E| = O(|V|)$ for all subgraph (V,E) ). Cliques with subdivided edges are so, but are not of bounded expansion.

## *FO model-checking for dense graphs*

Graphs of bounded clique-width : O(n), a special case of the theorem for MSO logic to be discussed below.

Graphs defined by first-order definabble transformations from graphs of bounded degree d : $O(n^s)$ (Gajarsky et al.). The exponent s depends on d and the sentence to check.

*Example*: Take a rectangular grid ; use labels to select 2 rows. Add edges between any two vertices of these 2 rows. They are not sparse and not of bounded clique-width.

*Observations* : (1) The constants “hidden” behind the O-notation are extremely large. The algorithms from the proofs are not practically implementable.

(2) First-order logic is weak for expressing graph properties.

(3) However, it is used in relational databases. The constraints like bounded degree, tree-width etc. are never satisfied by the relational structures representing the content of a database.

Polynomial-time algorithms are obtained from constraints on formulas.

For *conjunctive queries* $\varphi$ of *tree-width* $\leq k$, $S \models \exists \underline{y}\ \varphi$ is decidable in time $f(\varphi).|S|^{c(k)}$. (tree-width of the “graph of the variables” , two variables are adjacent if they are in a same atom).

# Monadic second-order (MSO) model-checking

MSO logic expresses many useful graph properties : planarity, connectedness, cycles, spanning trees, but also NP-complete problems : 3-vertex colorability, Hamiltonicity.

Examples : *3-colorability :*

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

*Non connectedness :*

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

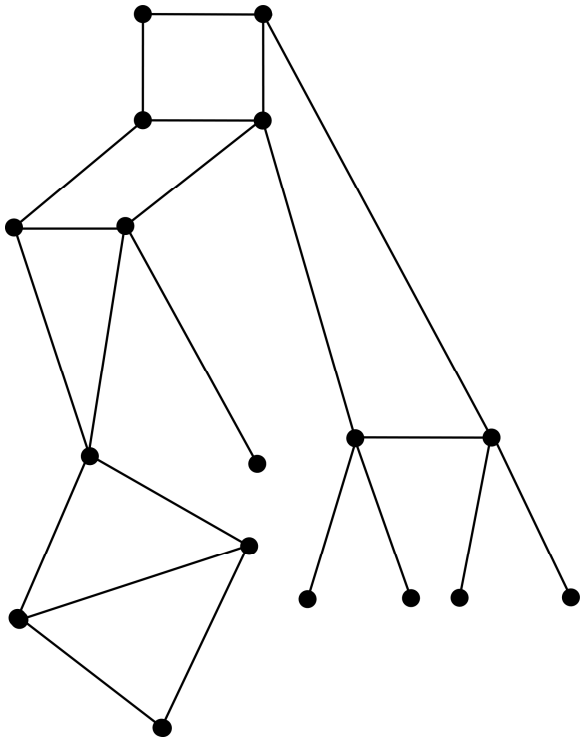
For getting polynomial time algorithms, tree-structurings of graphs, in particular those defined by tree-decompositions, are *needed*.

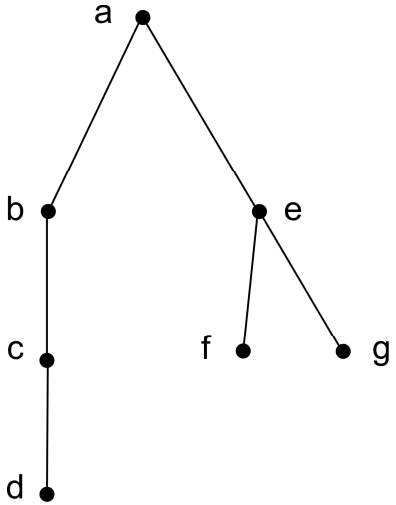
Only two types of decompositions (or equivalent notions) help : Tree-decompositions and descriptions of graphs by clique-width terms. They yield parameters in the sense of Fixed Parameter Tractability.

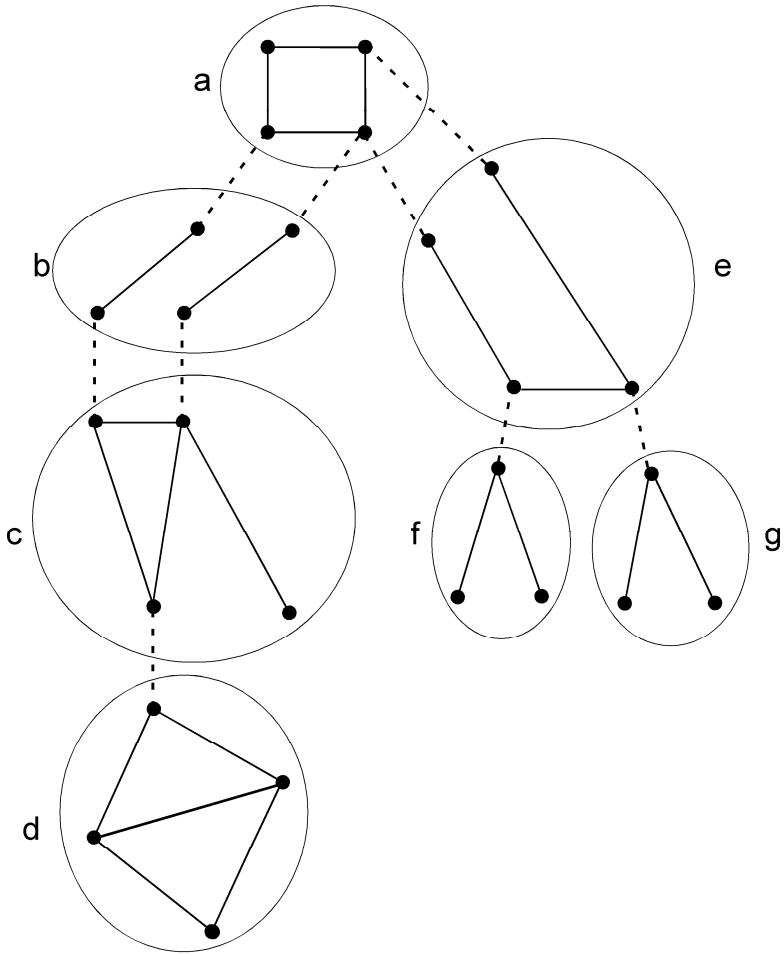
Both types of decomposition are expressed by algebraic terms over graph operations (that compose or transform graphs).

*Next* : a quick review of tree-width, clique-width and mutual relations.

# Tree-width ( twd(G) ) illustrated :






width of decomposition : 3 = 4-1

dotted lines : equal vertices.

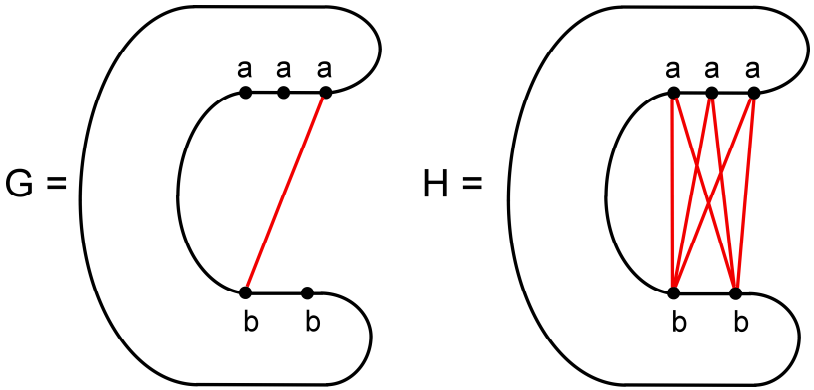
## Clique-width terms construct (labelled) graphs.

Vertices are labelled by *a,b,c, ...* . A vertex labelled by *a* is an *a-vertex*.

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

*Unary operations*: edge addition denoted by $add_{a,b}$

$add_{a,b}$(G) is G augmented with (un)directed edges from (between) every *a*-vertex to (and) every *b*-vertex.



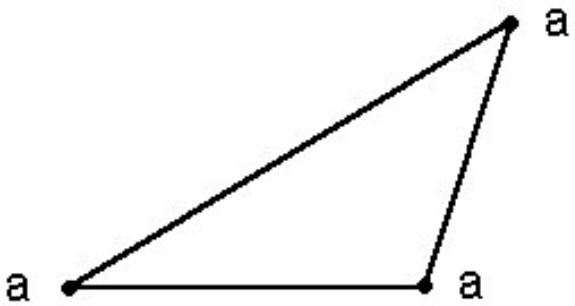

vertex relabellings :

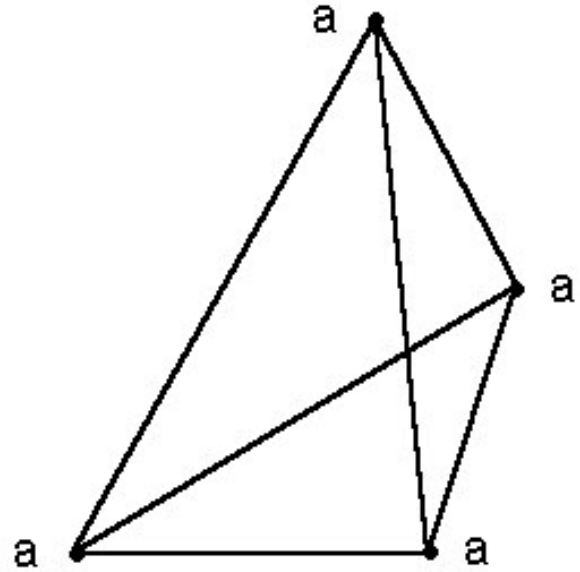
$relab_{a \to b}$(G) is G with every *a*-vertex is made into a *b*-vertex

Basic graphs : **a** denotes a vertex labelled by *a.*

The clique-width of G, denoted by cwd(G), is the smallest k such that G is defined by a term using k labels.

*Example : Cliques* with a-vertices, have clique-width 2 and unbounded tree-width.




$K_n$ is defined by $t_n$ where $t_1$ := **a**

$t_2$ := $relab_b \longrightarrow a$ ( $add_{a,b}$ (**a** $\oplus$ **b**) )

$t_3$ := $relab_b \longrightarrow a$ ( $add_{a,b}$ ($t_2$ $\oplus$ **b**) )

$t_4$ := $relab_b \longrightarrow a$ ( $add_{a,b}$ ($t_3$ $\oplus$ **b**) )

Bounded clique-width: cographs (2), cliques (2), complete bipartite graphs (2), trees (3), any class of bounded tree-width.

Unbounded clique-width: Planar graphs, chordal graphs, bounded degree.

Comparing tree-width and clique-width :

Undirected graphs :

$cwd(G) \leq 3.2^{twd(G)-1}$ (by Corneil & Rotics ; the exponential is not avoidable).

Directed graphs :

$cwd(G) \leq 2^{2.twd(G)+1}$ (Courcelle-Olariu, 2000; book Courcelle-Engelfriet 2012).

Classes for which cwd(G) = O(twd(G)$^{c}$ ) with "good values" of **c** and hidden constants .

| Graph class | cwd(G) where k = twd(G) |
|---|---|
| planar | 6k – 2 ( 32k – 24 if directed) |
| degree ≤ d | k.d + d + 2 |
| incidence graph | k + 3 ( 2k + 4 if directed) |
| p-planar | 12k.p |
| at most **q**. n edges for n vertices | O(k$^{q}$ ) where **q** << k |

Incidence graphs : for $MSO_2$ properties (MSO with edge quantifications), we get linear-time $MSO_2$ model-checking for graphs of bounded twd.

*Remark* : The algorithm that transforms a normalized tree-decomposition T into a clique-width term uses time :

$O(n.k.(\log(k) + m.\log(m)))$ where :

n = number of vertices = number of nodes of tree T,

k = the width of the tree-decomposition,

m = number of labels of the produced clique-width term.

Normalized : The vertices are the nodes of the tree.

## *The basic theorem for MSO model-checking*

*Theorem*: Each MSO property of graphs of cwd or twd $\leq k$ is decidable in time $f(k)$ **x** *# of vertices (hence FPT time, where the parameter is* $(k, \varphi)$*, and* $\varphi$ *expresses the considered property).*

Also for MSO properties expressed with edge set quantifications, ($MSO_2$) but only for graphs of bounded tree-width.

Restriction to graphs (or relational structures) of bounded tree-width is necessary for getting polynomial-time algorithms for $MSO_2$ model-checking.

*Theorem* (Kreutzer and Tazari, 2010): Assume ETH (3-SAT is not solvable in time $2^{o(n)}$). For a graph class closed under subgraphs, if $MSO_2$ model-checking is doable in time $O(n^{f(\varphi)})$, then $twd(G) < \log(n)^{48}$, $n = |V_G|$.

*The basic tool for the* MSO *meta-theorem:*

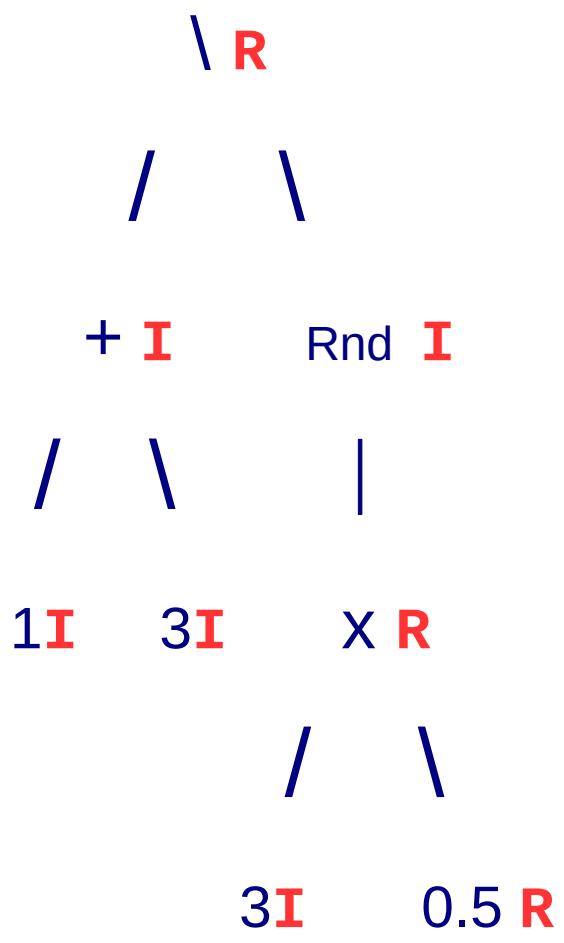
Translation of the MSO sentence $\varphi$ to check into a *finite automaton* *A* that runs on the term that defines the graph G, and accepts the term if and only if $G \models \varphi$.

*Major difficulty* : The "finite" automata are huge (this is not avoidable by Grohe 2004)

*Remedy*: Use of fly-automata : they compute the needed states and transitions. States are easily parsable words and transitions are defined by "small" programs. Only the needed transitions for a given term are computed.

# Finite automata on terms (also called tree-automata)

A small example with 2 states **I**(**Int**) and **R**(**Real**) for type-checking arithmetic expressions. Bottom-up computation using a table saying that:




**I** \ **I** = **R**

**I**+**I** = **I** ; Rnd(**R**)= **I**

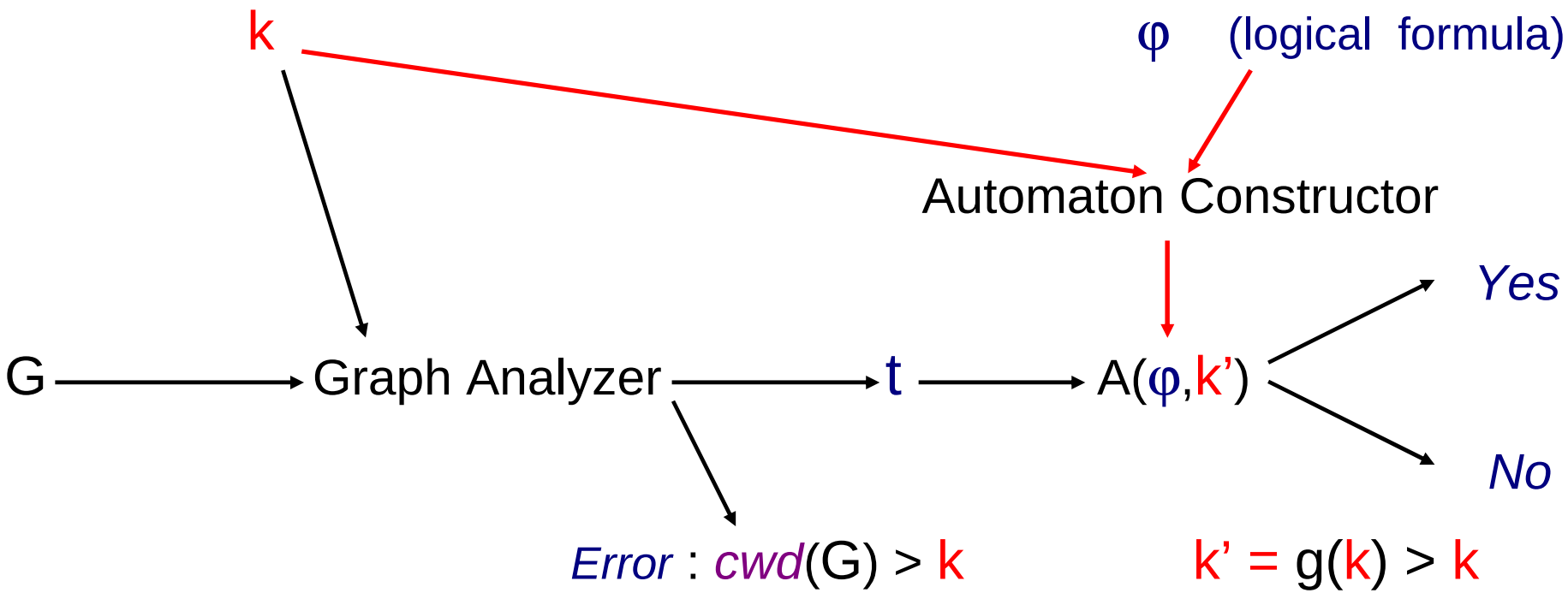
**I**X**R** = **R**

# Fly-automata for the verification of MSO graph properties

*Standard proof of the basic theorem* : For each MSO formula $\varphi$ and integer k, one builds a finite automaton A($\varphi$,k) that takes as input a term denoting a graph G of clique-width $\leq$ k and answers in time f(k).n whether G $\models \varphi$ (where n is the number of vertices).

The construction is by induction on the structure of $\varphi$.

# The MSO meta-theorem through *finite* automata:




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

A(φ,k'): "finite" automaton, running on terms t .

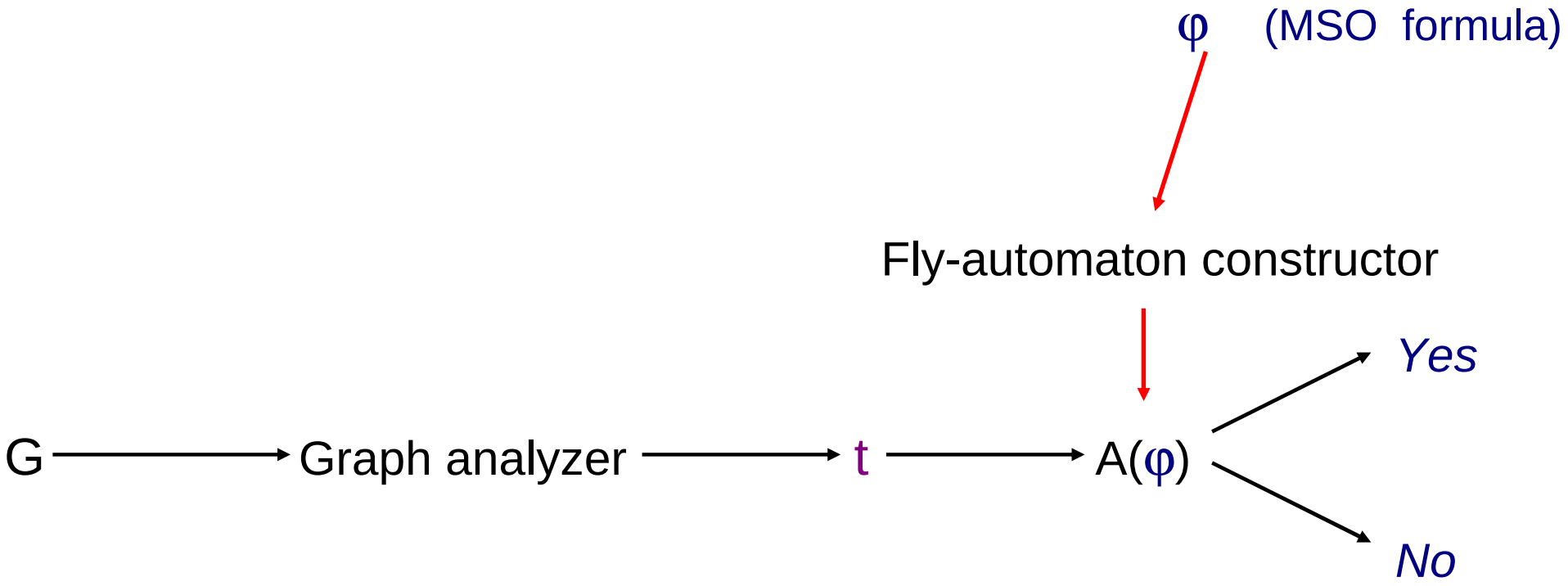
*Difficulty* : The *finite* automaton A(φ,k) is too large to be implemented by a (usual) transition table as soon as k ≥ 2 :

it may have 2^(2^(…2^k)..)) states, because of quantifier alternations.

To overcome this difficulty, we use fly-automata whose states and transitions are described and not tabulated. Only the, say 100, transitions necessary for an input term of size 100 are computed "on the fly".

Sets of states can be infinite and fly-automata can compute values, for example, the number of p-colorings of a graph.

# The MSO meta-theorem through *fly-automata*



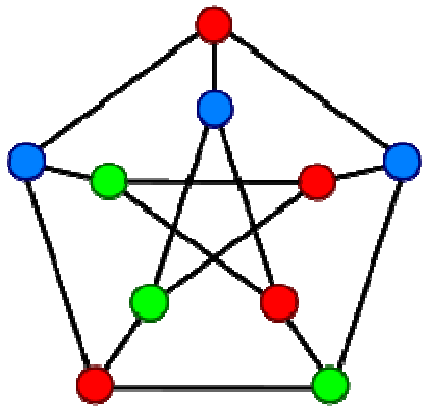

A(φ): *a single infinite fly-automaton*. The time taken by A(φ) is f(k).n where k depends on G via the operations occurring in t and bounds the tree-width or clique-width of G. No uniform linear bound on computation time.

# Computations using fly-automata (by Irène Durand)

Number of 3-colorings of the 3 x 100 rectangular grid (of clique-width 5) in a few seconds. For 4 x 150 : one minute for 3-colorability. A few seconds for 2-colorability.

4-acyclic-colorability of the Petersen graph (clique-width 5) in 2 minutes.

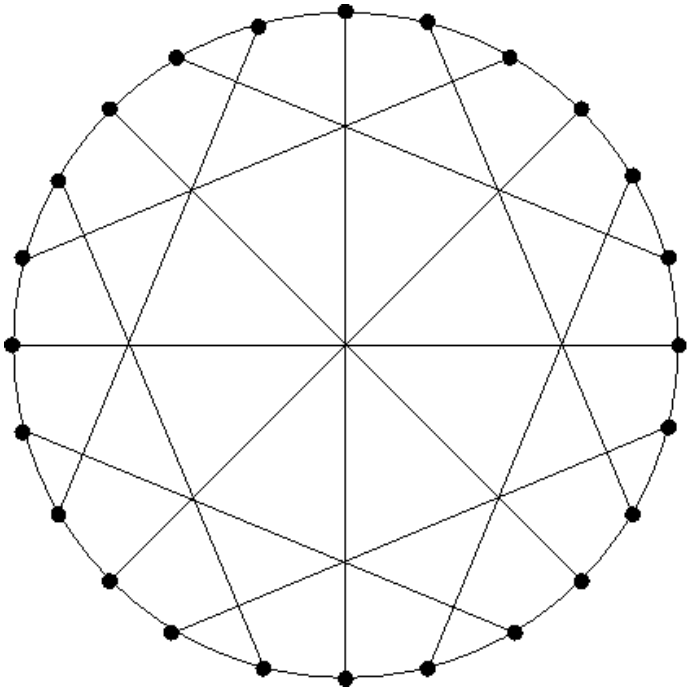
(3-colorable but not acyclically; **red** and **green** vertices induce a cycle).



The McGee graph

is defined by a clique-width term

of size 99 and depth 76.



This graph is 3-acyclically colorable.

Checked in 40 minutes.

Even in 2 seconds by enumerating the accepting

runs, and stopping as soon as a successful one is found.

# Fly-automaton (FA)

Definition : $A = < F, Q, \delta,$ Out $>$ (FA computing a function).

F : finite or countable (effective) set of operations,

Q : finite or countable (effective) set of states (integers, pairs of integers, *etc. :* states are encoded by finite words),

Out : Q → *D* , computable (*D* is an effective set, coded by finite words).

$\delta$ : computable (bottom-up) transition function

Nondeterministic case : $\delta$ is *finitely multi-valued* which ensures that determinization works

A fly-automaton defines a computable function : T(F) → *D* ,

hence, a decidable property if *D* = {*True, False*}.

*Extension :* computation of graph evaluations.

P($\underline{X}$) is a property of tuples $\underline{X}$ of sets of vertices (usually MSO expressible).

$\exists\, \underline{X}$.P($\underline{X}$) : the basic, "Boolean evaluation".

$\#\, \underline{X}$.P($\underline{X}$) : number of satisfying tuples $\underline{X}$.

**Sp** $\underline{X}$.P($\underline{X}$) : spectrum = the set of tuples of cardinalities of the components of the tuples $\underline{X}$ that satisfy P($\underline{X}$).

MinCard X.P(X) : minimum cardinality of X satisfying P(X).

Other optimal values can be computed.

# Computation time of a fly-automaton (FA)

F = all clique-width operations, $F_k$ : those using k labels.

On term $t \in T(F_k)$ defining G with n vertices, if a fly-automaton takes time bounded by :

$(k + n)^c$ → it is a P-FA (a polynomial-time FA),

$f(k). n^c$ → it is an FPT-FA,

$a. n^{g(k)}$ → it is an XP-FA.

The associated algorithm is polynomial-time, FPT or XP for clique-width as parameter. (The important notion is the maximum size of a state.)

All dynamic programming algorithms based on clique-width terms can be described by FA.

We obtain FPT algorithms parameterized by clique-width for the following problems and computations :

Number of p-colorings,

Minimum Cardinality of a color class X in a coloring with color classes X, $X_1$, … ,$X_p$ (for fixed p)

Equitable p-coloring (the sizes of two color classes differ by at most 1); this problem is W[1] (not FPT) for p+tree-width as parameter.

Fly-automata can be constructed :

- either "directly", from our understanding of the considered graph properties,

- or "automatically" from a logical description,

- or by combining previously constructed automata.

## Direct constructions

*Example 1* : Checking that a "guessed" p-coloring is good: a state is a *set* of pairs (a, j) where a is a label and j a color (among 1, …, p) or Error.

Checking the existence of a good p-coloring : a *set of such states*, in practice is not of maximal (exponential) size.

## *Example 2* : Connectedness.

The state at node u of term t is the *set of types (sets of labels)* of the connected components of the graph G(t/u). For k labels (k = bound on clique-width), the set of states has size $\leq$ 2 ^ (2 ^ k).

Proved lower bound : 2 ^ (2 ^ k/2).

→ Impossible to "compile" the automaton (*i.e.,* to list its transitions) .

*Example of a state* : q = { {a}, {a,b}, {b,c,d}, {b,d,f } }, (a,b,c,d,f : labels).

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.

Note *: Also state (p,p) if G(t/u) has $\geq$ 2 connected components, all of type p.*

Fly-automata can have *infinitely* many states and produce *outputs* : numbers, finite sets of tuples of numbers, etc.

*Example continued* : For computing the number of connected components, we use states such as :

q = { ({a}, 4 ), ({a,b}, 2), ( {b,c,d},2), ( {b,d,f },3) },

where 4, 2, 2, 3 are the numbers of connected components of respective types {a}, {a,b}, {b,c,d}, {b,d,f }.

This "counting construction" extends in a uniform way to any FA (the formal setting is based on semi-rings in place of the two Boolean values).

## Inductive construction for $\exists \underline{X}.\ \varphi(\underline{X})$ with $\varphi(\underline{X})$ MSO formula.

## Combinations and transformations of FA's.

*Product of* A and B : states are pairs of a state of A and one of B.

*Determinization of* A : states of Det(A) are finite sets of states of A because the transition is *finitely* multi-valued. At each position in the term, Det(A) gives the finitely many states that can in some computation (the automaton A can be infinite).

*Counting determinization of* A, yielding Cdet(A) : a state of CDet(A) is a finite multi-set of states of A (giving the *number of runs* that can yield a state of A, not only the existence).

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

1) we replace in F each **a** by the nullary symbol $(\mathbf{a}, (w_1,\ldots,w_p))$, $w_i \in \{0,1\}$ : we get $F^{(p)}$ (*only nullary symbols are modified*);

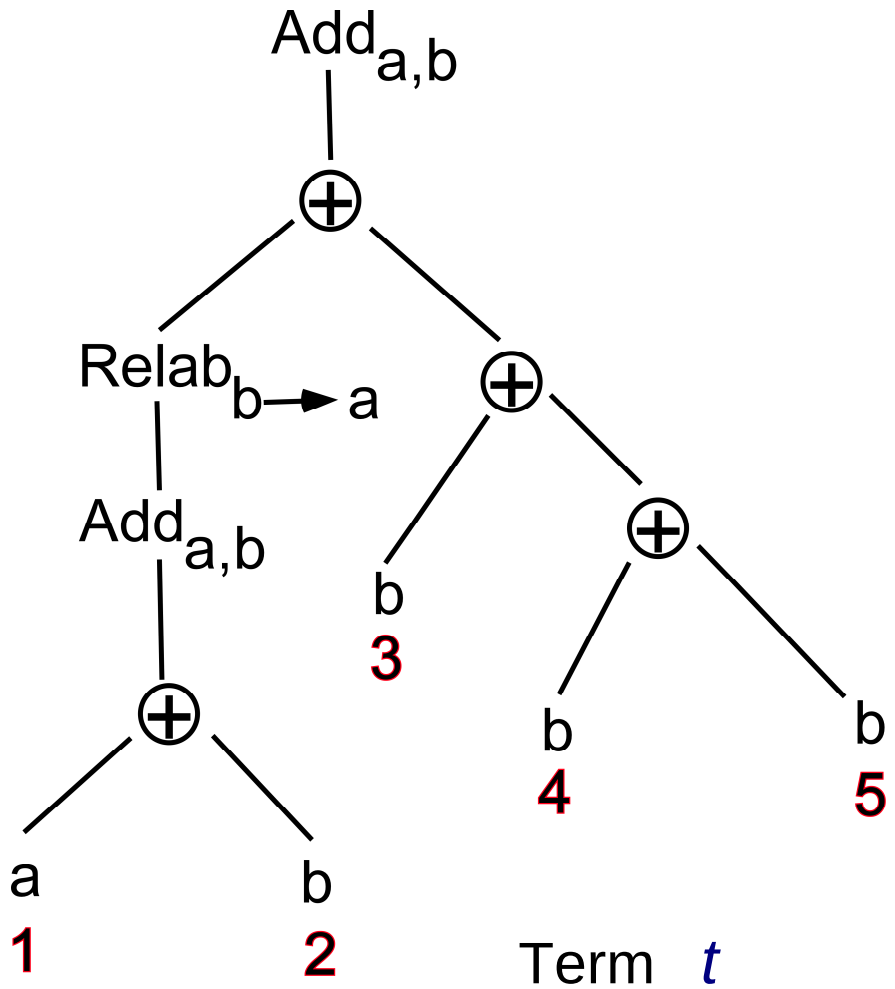
2) a term *s* in $\mathbf{T}(F^{(p)})$ encodes a term *t* in $\mathbf{T}(F)$ and an assignment of sets $V_1,\ldots,V_p$ to the set variables $X_1,\ldots,X_p$ :

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

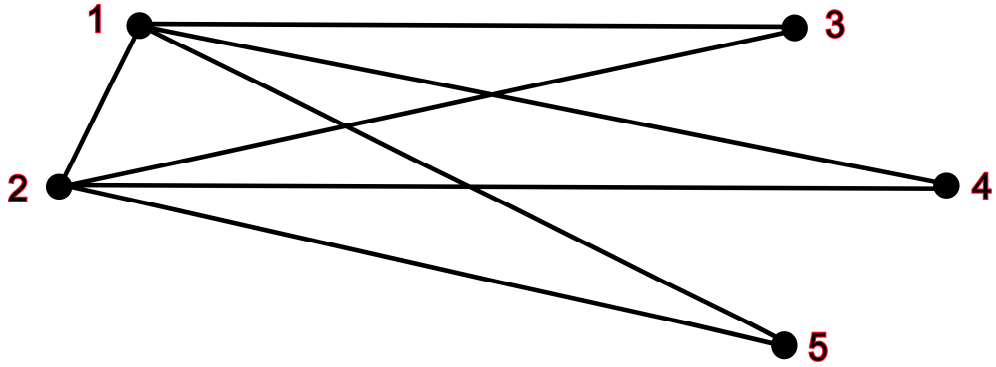
$w_i = 1$ if and only if $u \in V_i$ .

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

## *Example*

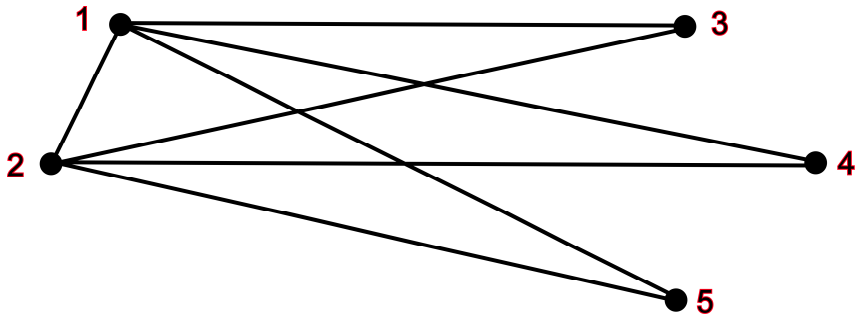


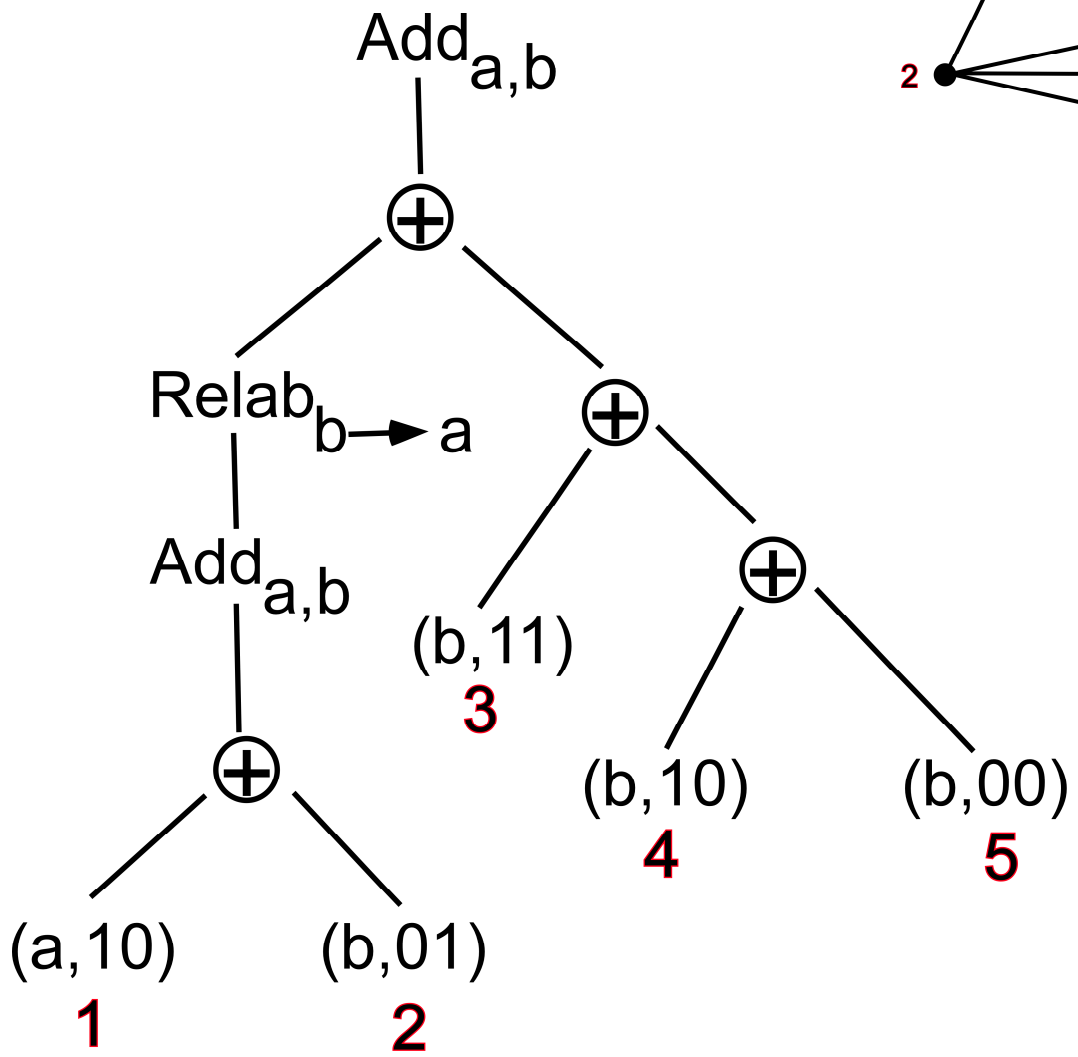

Term *t*




Graph G(*t*)

## *Example (continued)*




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

Term $t * (V_1, V_2)$

By an induction on $\varphi$, we construct, for each $\varphi(\underline{X})$, $\underline{X}=(X_1,\ldots,X_p)$, a fly-automaton $A(\varphi(\underline{X}))$ that recognizes :

$$L(\varphi(\underline{X})) : = \{ t * (V_1,\ldots,V_p) \in \mathbf{T}(F^{(p)}) \ / \ ( G( t ), V_1,\ldots,V_p ) \models \varphi \}$$

Atomic formulas ($X \subseteq Y$, edg(X,Y) ) : direct constructions

$\neg P$ (negation) : as FA are run deterministically (by computing at each position the finite set of reachable states), it suffices to exchange accepting and non-accepting states.

$P \wedge Q$, $P \vee Q$ : products of automata.

*Quantifications:* Formulas are written without $\forall$

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

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

where $pr_{p+1}$ is the *projection* that eliminates the last Boolean

→ a *non-deterministic* FA denoted by $pr_{p+1}( A ( \varphi(X_1, ..., X_{p+1}) )$, to be run deterministically.

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

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)

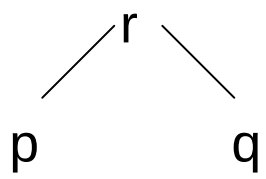
Error : all other cases

*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 $\overrightarrow{add}_{a,b}$




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 , 0, Ok, c(i), cd or dc where c ≠ a then r : = p

## *Examples* : p-acyclic colorability

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

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

$$\exists X_1,\ldots,X_p\ (\text{Disjoint}(X_1,\ldots,X_p)\ \wedge\ \text{Conn}(X_1)\ \wedge\ldots\wedge\ \text{Conn}(X_p)\ \wedge\ldots$$
$$\ldots\ \wedge\ \text{Link}(X_i\ ,\ X_j)\ \wedge\ldots\ )$$

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

Combinations of existing FA reflect the structure of MSO sentences

# Enumeration and efficient recognition

Recognition with Det(A) reports the answer when all states at the root have been determined.

An *enumerating computation* can list one by one:

the states reached at the root by the different runs of A,

the tuples $\underline{X}$ that satisfy an MSO property $P(\underline{X})$.

Recognition by enumeration of root states stops as soon as

an accepting state is found. This is appropriate if $\exists\, \underline{X}.P(\underline{X})$ holds. Not for counting accepting runs or $\#\, \underline{X}.P(\underline{X})$

*Technical remark* : If a graph is denoted by a clique-width term $t$, each of its vertices is represented in $t$ at a single position (an occurrence of a nullary symbol).

If the operation // is also used ( G // H is obtained from disjoint G and H by fusing some vertices of G to some vertices of H, in a precise way fixed by labels), then a vertex of G//H is represented by several positions of the term. The automaton that checks a property $\varphi(X_1, ..., X_p)$ of G denoted by a term $t$ must also check that the Booleans that specify $(X_1, ..., X_p)$ agree on all positions of $t$ that specify a same vertex of G.

We have no such difficulty if we use disjoint union instead of //. Hence, for representing tree-decompositions, clique-width terms are more convenient if one uses automata constructed from logical formulas.

# Application to $MSO_2$ properties of graphs of bounded tree-width *via* incidence graphs.

1) *Recall :* From of a tree-decomposition of G of width k, we construct a term t for Inc(G) of "small" clique-width k+3 (or 2k+4).

2) *Recall :* We translate an $MSO_2$ formula $\varphi$ for G into an MSO formula $\theta$ for Inc(G).

3) The corresponding automaton **A**($\theta$) takes term t as input. *But* an atomic formula edg(X,Y) of $\varphi$ is translated into $\exists U.\ inc(X,U) \wedge inc(U,Y)$ in $\theta$ which adds one level of quantification.

*Fact* : The automaton **A**($\theta$) remains manageable.

For certain graph properties P, for example "connectedness", "contains a directed cycle" or "outdegree < p", we have :

$$P(G) \Leftrightarrow P(Inc(G)).$$

The automaton for graphs G defined by clique-width terms can be used "directly" for the clique-width terms that define the incidence graphs Inc(G).

## Summary : Checking properties of G of tree-width < k

| MSO property | $MSO_2$ property |
|---|---|
| cwd term for G<br>of width O(k) or $O(k^q)$<br>in "good cases" and<br>exponential in bad ones | cwd term for Inc(G)<br>of width O(k);<br>more complicated<br>automaton in some cases,<br>because of edg(X,Y) |

# General conclusion

1) By uniform constructions, we get dynamic programming algorithms based on fly-automata, that can be quickly constructed from logical descriptions → *flexibility*.

A "small" modification of the input formula is reflected easily in the automaton.

2) It is hard to obtain tight upper-bounds to time computations.

3) The algorithms obtained from FA are not better than the specific ones that have been developed. There obtained in uniform ways, rather quickly, as combinations of existing "basic" automata.

4) Even for graphs given by tree-decompositions, clique-width terms are appropriate because of two facts:

(a) fly-automata are simpler to construct and

(b) it is practically possible to translate tree-decompositions of "certain" sparse graphs into clique-width terms.

5) Fly-automata are implemented. Tests have been made mainly for colorability and Hamiltonicity problems.

6) Our constructions and their TRAG implementation concern directed graphs as well as undirected ones.

*References for monadic second-order model checking* :

B.C, Irène Durand : Automata for the verification of monadic second-order graph properties, *J. Applied Logic* **10** (2012) 368-409

and also : Computation by fly-automata beyond monadic second-order logic, Theoretical Computer Science, **619** (2016) 32-67,

B.C.: From tree-decompositions to clique-width terms, *Discrete Applied Mathematics*, **248** (2018) 125-144

and also : Fly-automata for checking MSO2 graph properties, *Discrete Applied Mathematics*, **245** (2018) 236-252.

B.C. , Joost Engelfriet : Graph structure theory and MSO logic, Cambridge University Press, 2012.