Monadic second-order model-checking with fly-automata

Bruno Courcelle and Irène Durand

Bordeaux University, LaBRI (CNRS laboratory)

References : B.C, I. D.: 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, and also : Fly-automata for checking MSO2 graph properties, both *Discrete Applied Maths*, on line on ScienceDirect.com, to appear in 2018

Introduction to a demonstration of some features of the TRAG system

by I.Durand, and M.Raskin for the online version

Computation of graph decompositions of two kinds : tree-decompositions and expressions by clique-width terms ;

Checking MSO (*Monadic second-order*) properties, possibly expressed with edge set quantifications ;

Computing MSO-definable values (e.g. # of 3-colorings).

Automatic construction of automata from MSO sentences.

We use *fly-automata* (they *compute* their transitions) that run on clique-width terms describing the input graphs.

Existential properties imply nondeterminism.

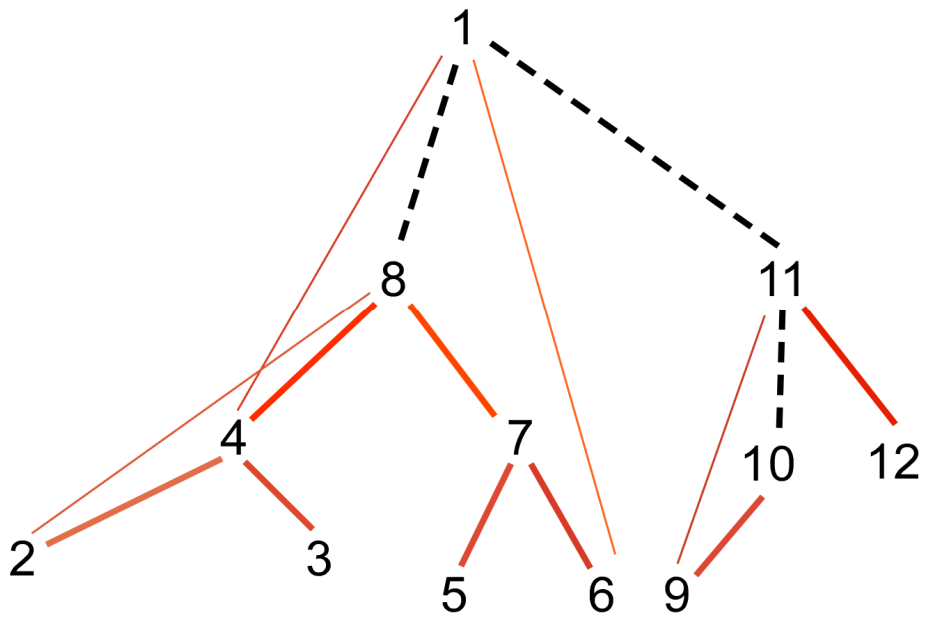
Nondeterministic automata run in several ways :

by simulating a deterministic automaton,

and possibly counting the number of runs,

or by an “enumerating” computation that stops on first success : this is ok for a positive answer.

Tree-decompositions are defined from normal trees.



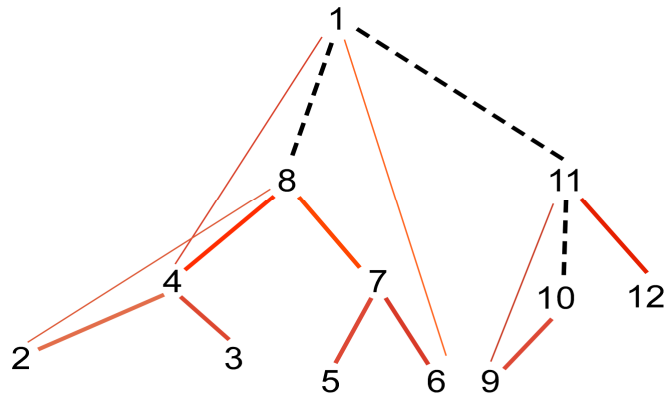
Red edges are in the graph; dotted edges in the tree, not in the graph

The tree-decomposition is (T, f_T) where :

$f_T(u) := \{u\} \cup \{ v >_T u \ / \ v \text{ is adjacent to some } w \leq_T u\}$

Above : $f_T(4) := \{4,8,1\}$: the edge 2-8 "jumps" over 4

Encoding the tree-decomposition by the word **TreeBoxes**



Tree =(1 8)(1 11)(8 4)(8 7)**(4 2)**(11 12)(11 10)(10 9)(4 3)
(7 5)(7 6)
Boxes = 1:NIL **8:(1):NIL** 11:(1):NIL 4:(1 8):(1 8) **7:(1 8):(8)**
10:(11):NIL 12:(11):(11) 2:(4 8):(4 8) 3:(4):(4) 5:(7):(7)
6:(1 7):(1 7) 9:(10 11):(10 11)

In **Tree,(4 2)** means that 2 is a son of 4; T is described top-down.

In **Boxes, 7:(1 8):(8)** means that the box of 7 is {7,1,8} and 8 (only) is adjacent to 7.

8:(1):NIL :The box of 8 is {8,1} but 8 is not adjacent to 1.

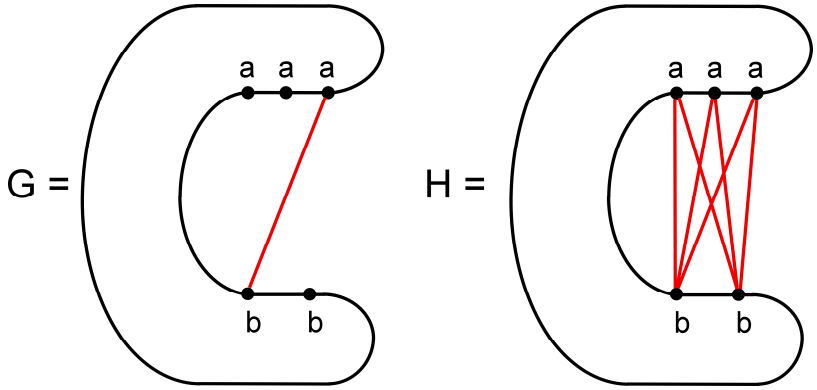
Clique-width : an algebraic construction of graphs

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.



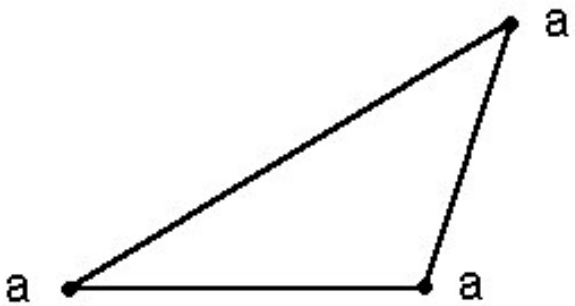
vertex relabellings :

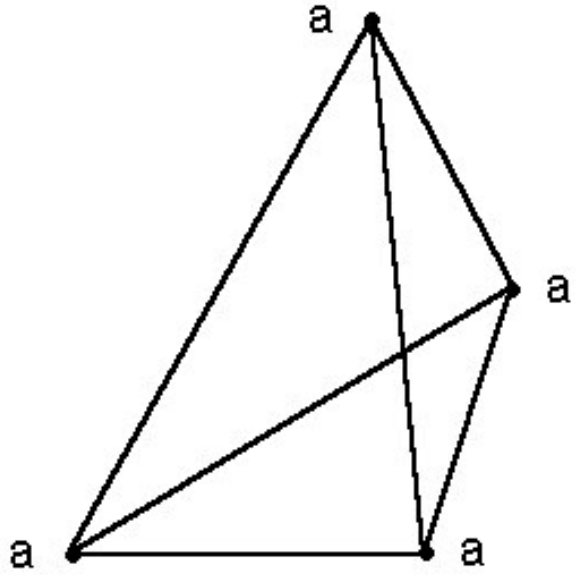
$relab_{a \to b}$(G) is G with every *a*-vertex is made into a *b*-vertex

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 (a-labelled) 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**))

Algorithms for decompositions

Heuristic algorithm for tree-decomposition ;

Translation from tree-decomposition to cwd-term ;

Exact algorithm for cwd-term (for “small” undirected graphs, uses a reduction by Heule and Szeider to a SAT problem) ;

Heuristic algorithms for cwd-terms to define directed or undirected graphs ;

Heuristic algorithms for cwd-terms defining incidence graphs (useful for MSO formulas using edge set quantifications).

End of introduction

Main features of the formal setting

Goal: Fixed-parameter tractable (FPT) graph algorithms for monadic second-order (MSO) expressible problems,

for graphs of bounded tree-width (twd) or clique-width (cwd),

based on automata running on algebraic terms denoting the (decomposed) input graphs.

Can compute values, not only *True / False* answers.

Tools: (1) Fly-automata (FA): they *compute* their transitions, to overcome the well-known "huge size problem",

(2) Tree-decompositions encoded by clique-width terms,

(3) Linear bounds on cwd in terms of twd for sparse graphs.

The basic theorem: Each MSO property of graphs of cwd or twd $\leq k$ is decidable in time $f(k)$ **x** #*vertices*.

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

Graphs given with relevant decompositions, of “small width”.

Optimal decompositions are difficult to construct (NP-complete problems). But optimality is not essential.

Computation of graph evaluations (among others)

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.

Informal review of basic notions

1) Graphs G are finite, simple, loop-free, directed or not.

Can be given by the logical structure

(V_G , $edg_G(.,.)$) = (vertices, adjacency relation)

2) Monadic second-order (MSO) formulas can express p-colorability (and variants), transitive closure, properties of paths, connectedness, planarity (via Kuratowski), *etc…*

Examples : *3-colorability :*

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

The graph is not connected :

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

Planarity is MSO-expressible (no minor K_5 or $K_{3,3}$).

Alternative description of graphs :

Inc(G) := (V_G U E_G , $inc_G(.,.)$)

= (vertices *and* *edges*, incidence relation)

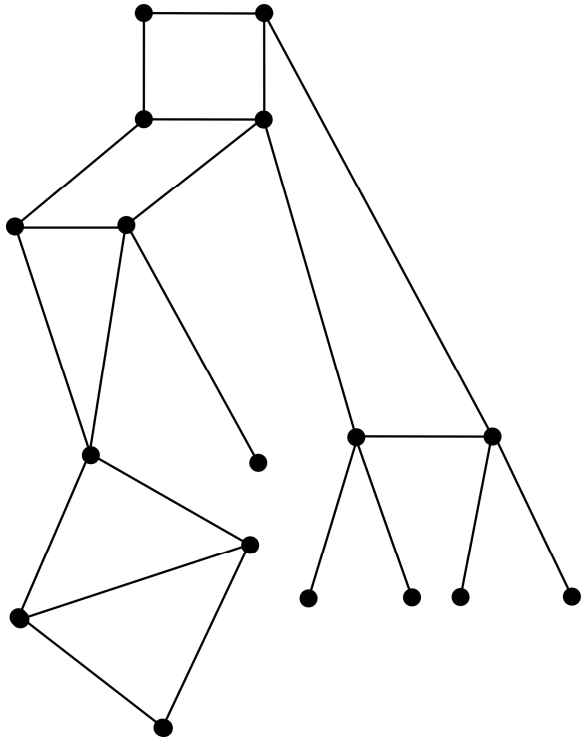
→ the bipartite *incidence graph* of G.

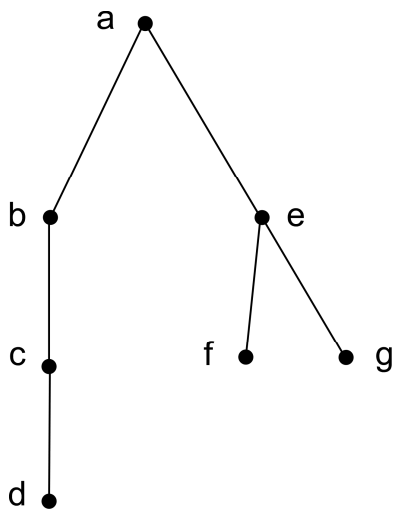
MSO formulas on Inc(G) can use quantifications on sets of edges of the considered graph G.

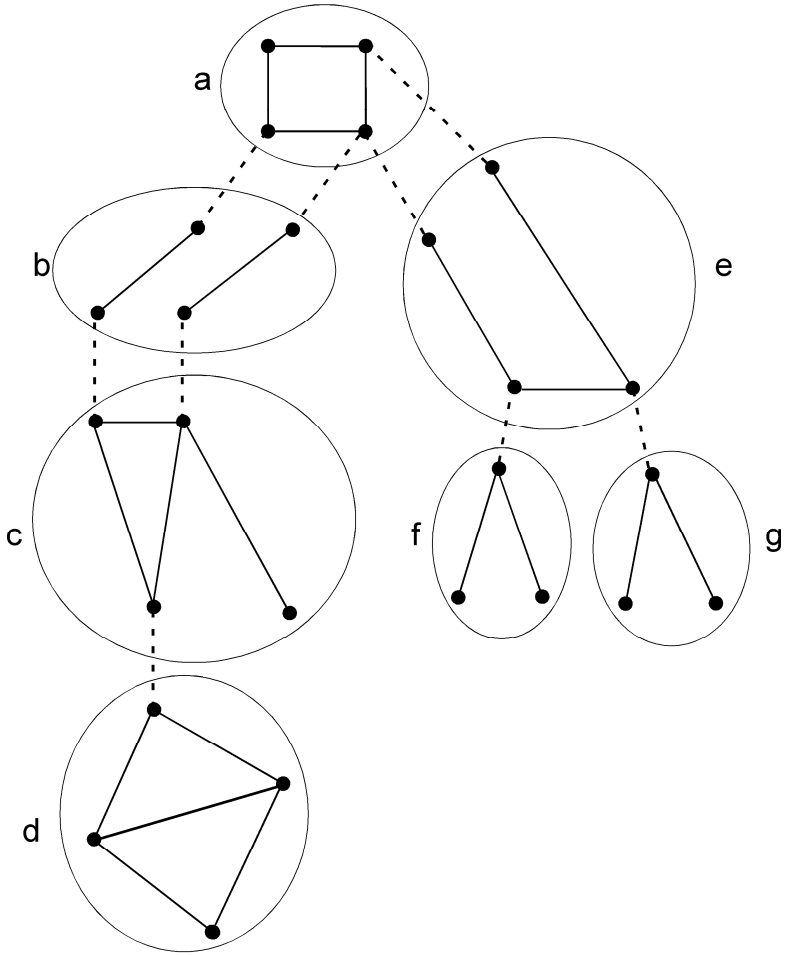
They are called MSO_2 formulas.

Expressing Hamiltonicity of G is possible by an MSO formula on Inc(G) but not on G (hence, MSO_2 but not MSO).

Tree-width (twd(G)) is well-known







width of decomposition : 3 = 4-1

dotted lines : equal vertices

Normal trees : compact data structure and easy logical description

Clique-width has been defined in introduction.

Classes of bounded clique-width:

cographs, cliques, complete bipartite graphs, trees,

any class of bounded tree-width.

Classes of unbounded clique-width:

Planar graphs, chordal graphs.

Comparing tree-width and clique-width (undirected graphs)

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

(by Corneil & Rotics ; the exponential is not avoidable).

For which classes do we have cwd(G) = O(twd(G)$^{\mathbf{c}}$) for fixed **c**, and with "good values" of **c** and of hidden constants ?

Graph class	cwd(G) where k = twd(G)
planar	6k – 2 (32k – 24 if directed)
degree $\leq$ 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$^{\mathbf{q}}$) where q << k

These results hold for directed graphs.

Remark : About incidence graphs of graphs of bounded tree-width and MSO_2 properties (using MSO formula with edge set quantifications).

Example : There exists a set of edges forming a perfect matching, or forming a Hamiltonian path. Not possible without such quantifications.

1) From of a tree-decomposition of G of width k, we construct a clique-width term t for Inc(G) of "small" width k+3 (or 2k+4); no exp. !

2) We translate an MSO_2 formula φ for G into an MSO formula θ for Inc(G).

3) The corresponding automaton A(θ) takes term t as input.

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

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

n = # vertices = # nodes tree T,

k = the width of the tree-decomposition (T, f_T),

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

First conclusions

From a "good" tree-decomposition of a sparse graph (planar, bounded degree, etc…), we can get a "good" clique-width term, of comparable width (we avoid the general exponential jump).

Many algorithms construct "good" (not optimal) tree-decompositions, but not so many construct "provably good" clique-width terms (however, see TRAG).

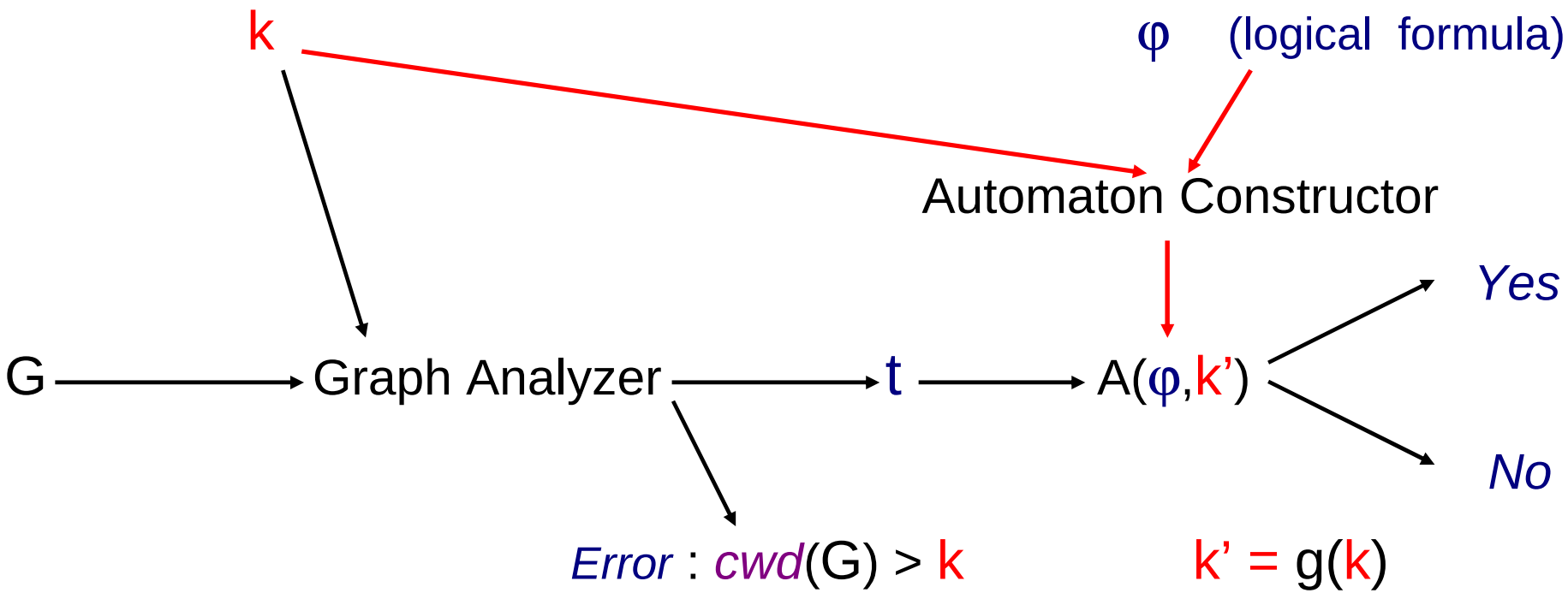
Clique-width terms yield easier constructions of fly-automata than tree-decompositions.

Fly-automata for the verification of MSO graph properties

Standard proof of the basic theorem : For each MSO formula φ and integer k, one builds a finite automaton A(φ,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 φ.

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 .

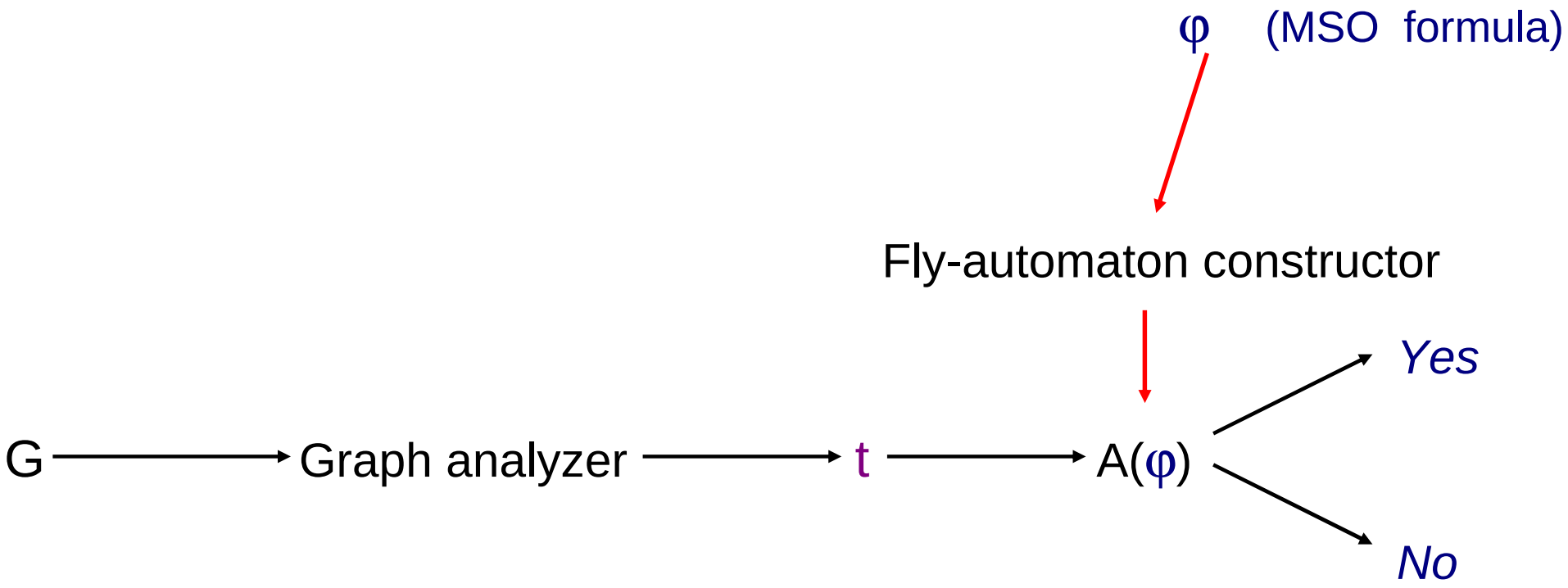
Difficulty : The *finite* automaton A(φ,k) is too large to be implemented by a (usual) transition table as soon as k ≥ 2 :

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 (say 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*



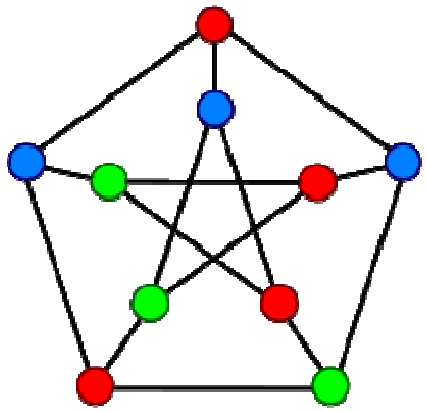
A(φ): *a single infinite fly-automaton*. The time taken by A(φ) is f(k).n where k depends on the operations occurring in t and bounds the tree-width or clique-width of G.

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, takes one minute for 3-colorability. A few seconds for 2-colorability.

4-acyclic-colorability of the Petersen graph (clique-width 5) in 1.5 minutes.

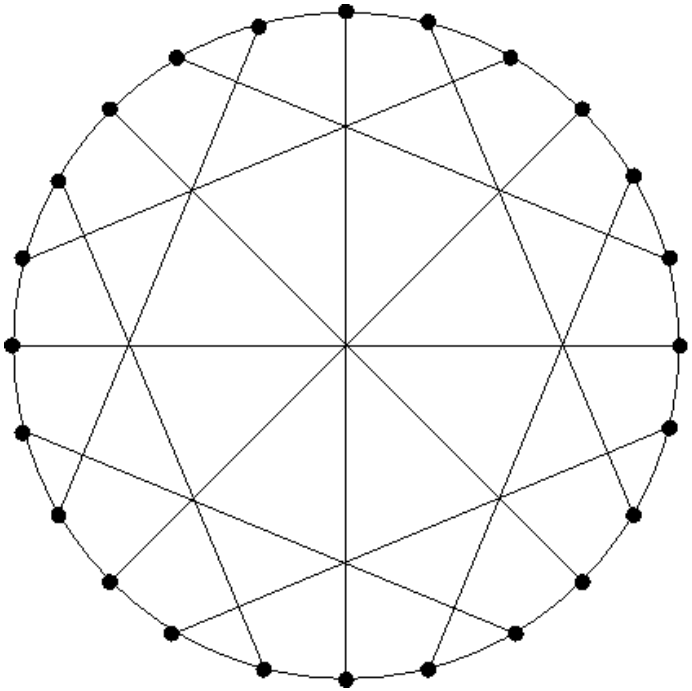
(3-colorable but not acyclically; **red** and **green** vertices induce a cycle).



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 that computes 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).

δ : computable (bottom-up) transition function

Nondeterministic case : δ 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*}.

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(t) 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 max. 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 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 may 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 replacing the two Boolean values).

Combinations and transformations of fly-automata.

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).

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)

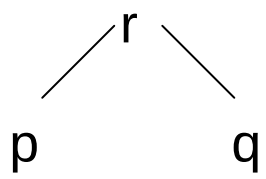
Error : all other cases

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 $\#\, \underline{X}.P(\underline{X})$

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

Atomic formulas (for example $X \subseteq Y$, 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.

How to handle free variables for queries $\varphi(\underline{X})$ and for $\exists \underline{X}.\varphi(\underline{X})$?

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*);

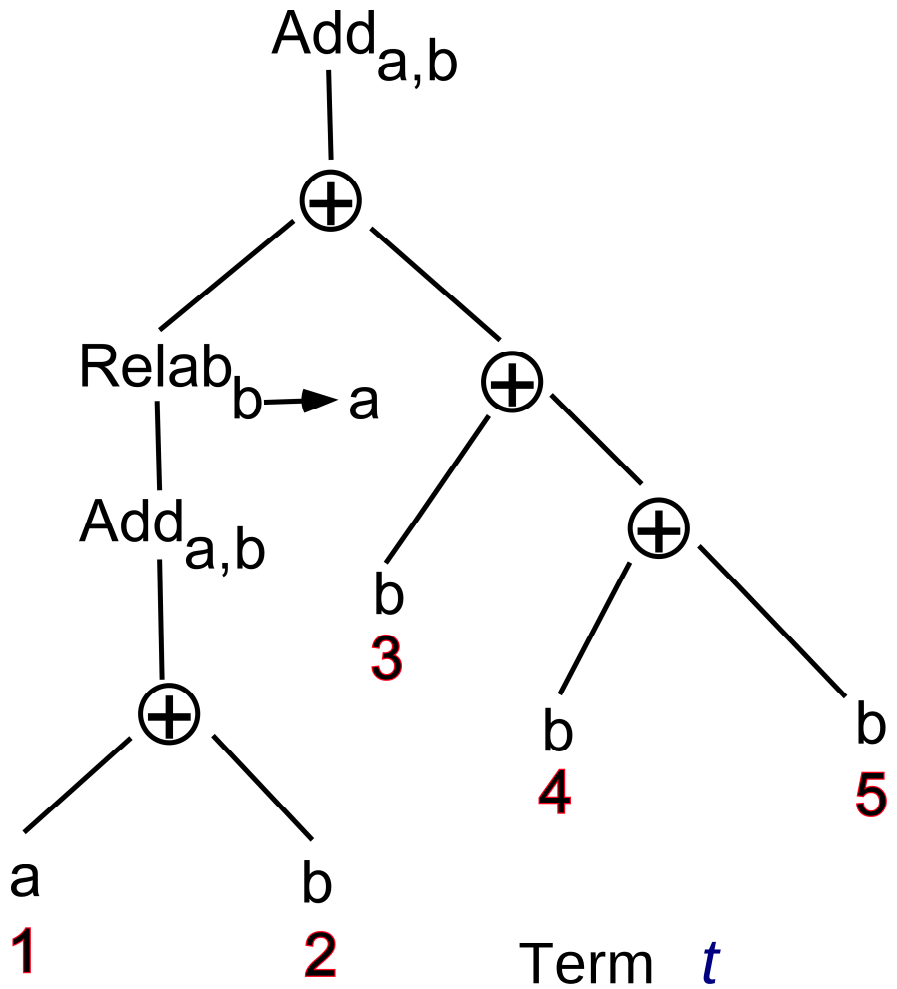
2) a term s in $\mathbf{T}(F^{(p)})$ encodes a term t in $\mathbf{T}(F)$ and an assignment of sets $V_1,\ldots,V_p$ to the set variables $X_1,\ldots,X_p$:

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

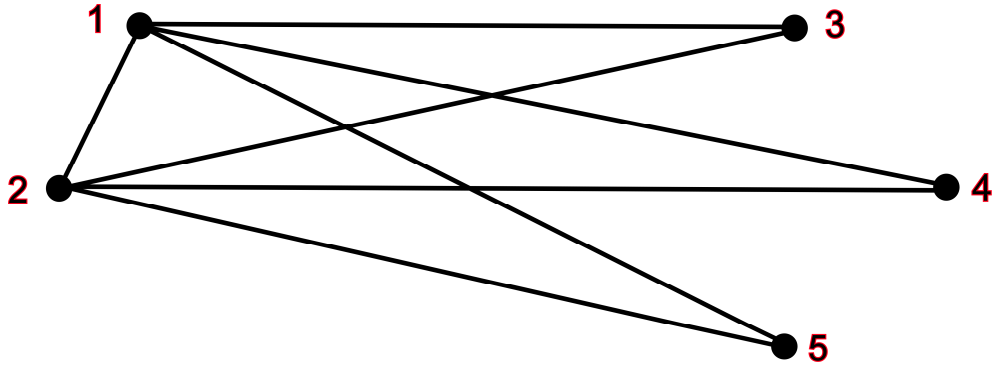
$w_i = 1$ if and only if $u \in V_i$.

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

Example

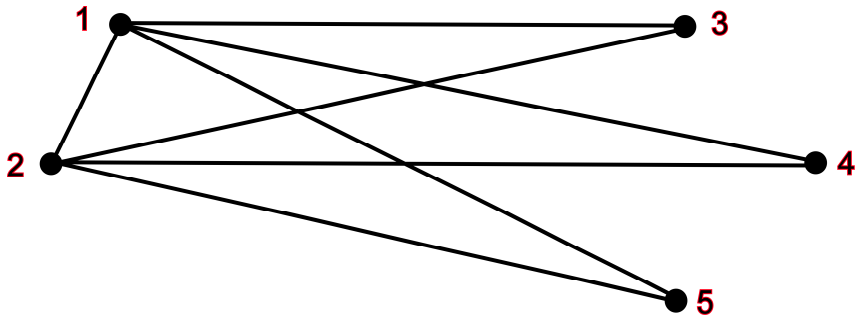


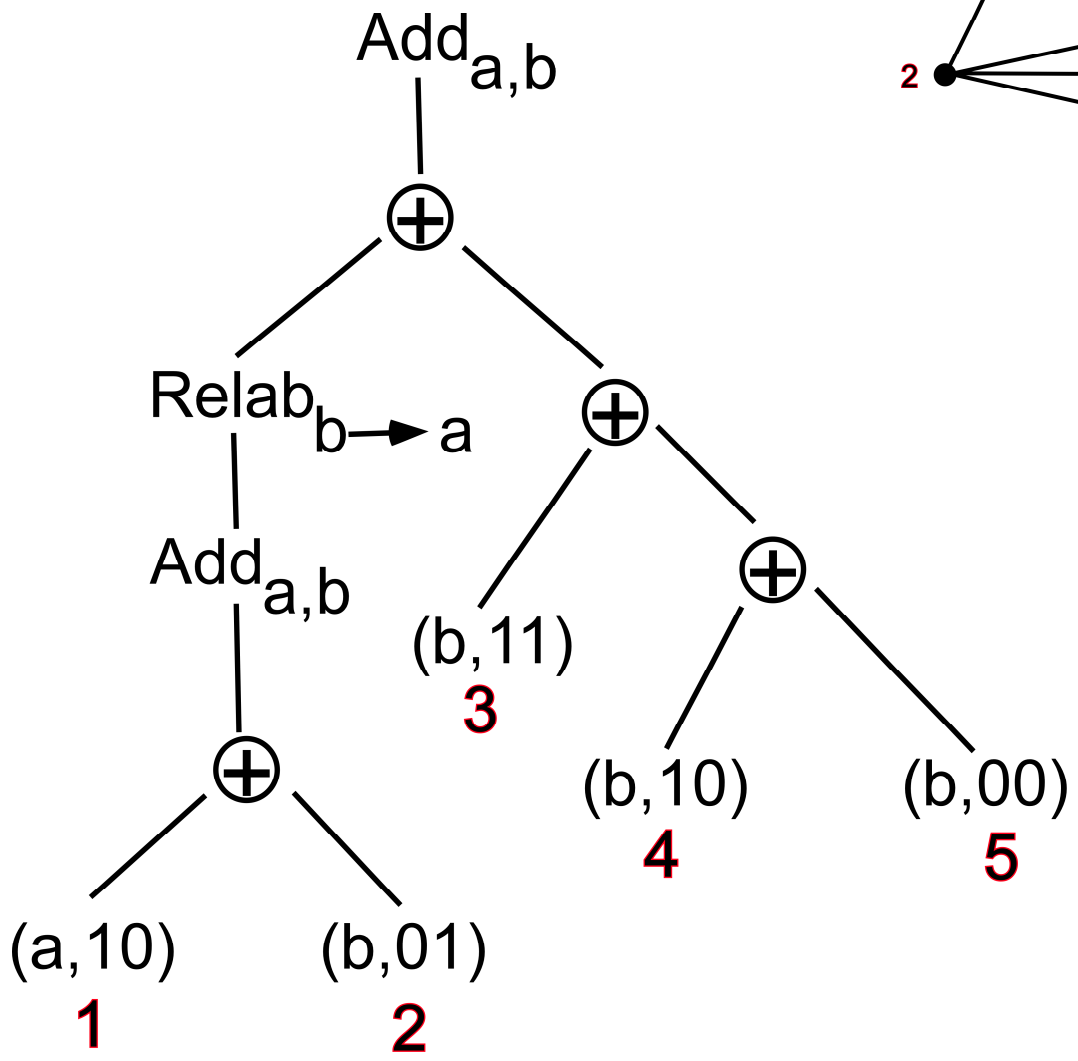
Term *t*



Graph G(*t*)

Example (continued)





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

Term $t * (V_1, V_2)$

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

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

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.

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 φ for G into an MSO formula θ for Inc(G).

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

Fact : The automaton **A**(θ) 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 of width O(k) or $O(k^q)$ in "good cases" and exponential in bad ones	cwd term for Inc(G) of width O(k); more complicated automaton in some cases, 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.