



Computation and enumeration
by fly-automata

Bruno Courcelle

Irène Durand

Bordeaux University, LaBRI (CNRS laboratory)

Topics

Fixed-parameter tractable (FPT) graph algorithms based on infinite “fly”-automata (FA) running on clique-width terms denoting the input graphs.

Generic constructions of FA : example, the number of accepting runs of a nondeterministic FA, based on attributed FA.

Meta-dynamic-programming: constructions from logical descriptions of problems.

Part 1 : review of definitions and basic facts.

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

A graph G is given by the logical structure

$$(V_G, \text{edg}_G(\dots)) = (\text{vertices, adjacency relation})$$

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

Clique-width (denoted by $cwd(G)$).

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

Binary operation: disjoint union : \oplus

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

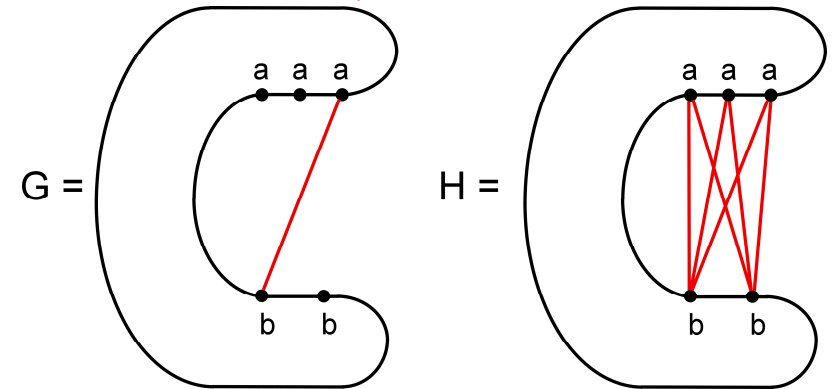
$Add_{a,b}(G)$ is G augmented

with (un)directed edges from (between) every a -vertex to (and) every b -vertex.

Vertex relabellings :

$Relab_a \rightarrow b(G)$ 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 $\text{cwd}(G)$) is the smallest k such that G is defined by a term using k labels.

Each MSO property φ can be checked in polynomial time by a finite automaton $A(\varphi, k)$ taking as inputs terms denoting graphs of clique-width $\leq k$.

Difficulty : The *finite* automaton $A(\varphi, k)$ is much too large as soon as

$k \geq 2$: $2^{(2^{(\dots 2^k \dots)})}$ states (because of quantifier alternations).

To overcome this difficulty, we use *fly-automata* whose states and transitions are *described* and *not tabulated*. Only the transitions necessary for an input term are computed “on the fly”.

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

Part 2 : Fly-automaton (FA)

$A = \langle F, Q, \delta, \text{Out} \rangle$ to compute 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),

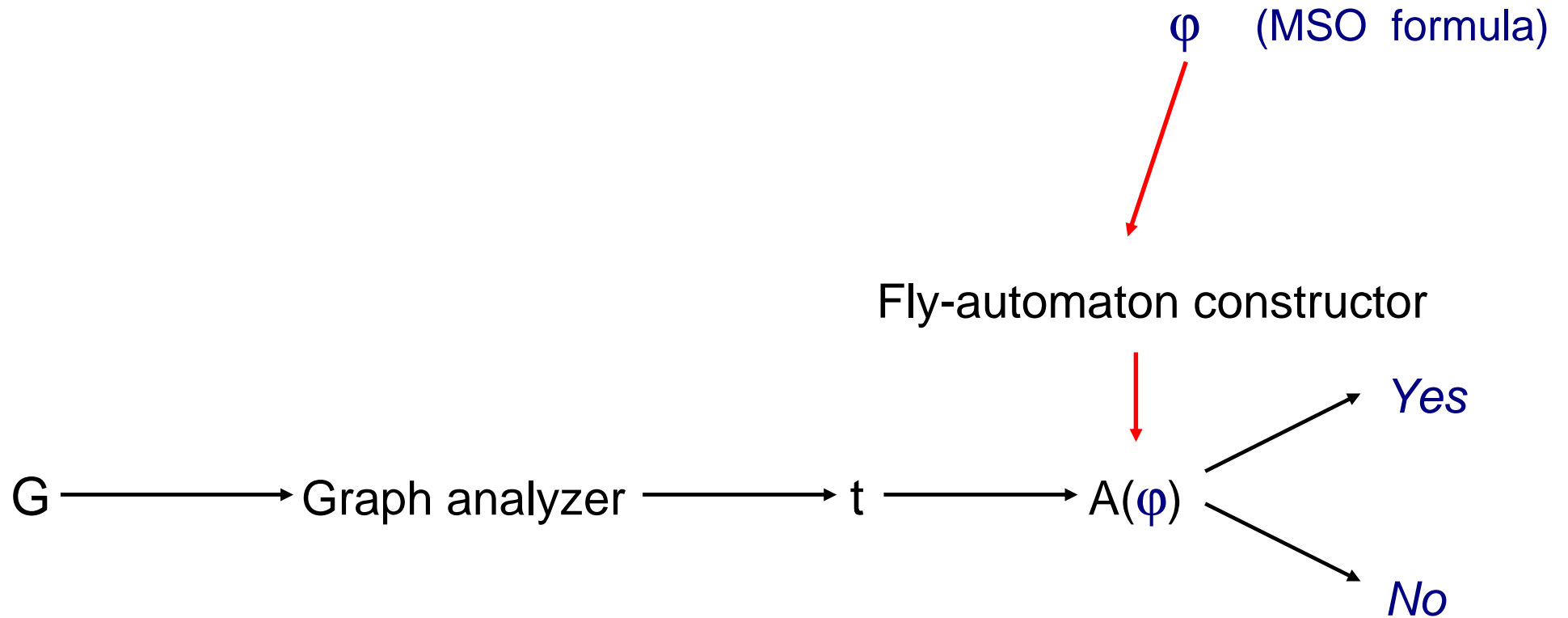
$\text{Out} : Q \rightarrow D$, computable (D is effective, coded by finite words).

δ : computable (bottom-up) transition function

Nondeterministic case : δ is *finitely multi-valued*. Determinization works.

An FA defines a computable function : $T(F) \rightarrow D$, a decidable property if $D = \{\text{True}, \text{False}\}$.

The MSO meta-theorem through *fly-automata*



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

Computation time of a fly-automaton

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 :

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

$f(k).n^c \rightarrow$ it is an FPT-FA,

$a.n^{g(k)} \rightarrow$ it is an XP-FA.

The associated algorithm is polynomial-time, FPT or XP for clique-width as parameter.

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

Part 3 : Computing graph evaluations

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

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

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

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

$\text{MSp}\underline{X}.P(\underline{X})$: **multispectrum** = the corresponding **multiset**.

(for $\underline{X} = X$: the set of pairs (m, i) such that $i > 0$ is the number of sets X of cardinality m that satisfy $P(X)$).

$\text{MinCard } X.P(X)$: minimum cardinality of X satisfying $P(X)$.

SetVal- $\alpha(\underline{X})/P(\underline{X})$: the set of values of $\alpha(\underline{X})$

for the tuples \underline{X} that satisfy $P(\underline{X})$.

Sat $\underline{X}.P(\underline{X})$: the set of all tuples \underline{X} that satisfy $P(\underline{X})$.

Inductive construction for $\exists \underline{X}.P(\underline{X})$ based
on an MSO formula $\varphi(\underline{X})$

Atomic formulas : direct constructions

$\neg P$ (negation) : FA are run deterministically, it suffices to exchange
accepting/non-accepting states.

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

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

Terms are equipped with **Booleans** that encode assignments of vertex sets V_1, \dots, V_n to the free set variables X_1, \dots, X_n of MSO formulas (*formulas are written without first-order variables*):

1) we replace in F each **a** by the nullary symbol

$(\mathbf{a}, (w_1, \dots, w_n))$, $w_i \in \{0, 1\}$: we get $F^{(n)}$ (*only nullary symbols are modified*);

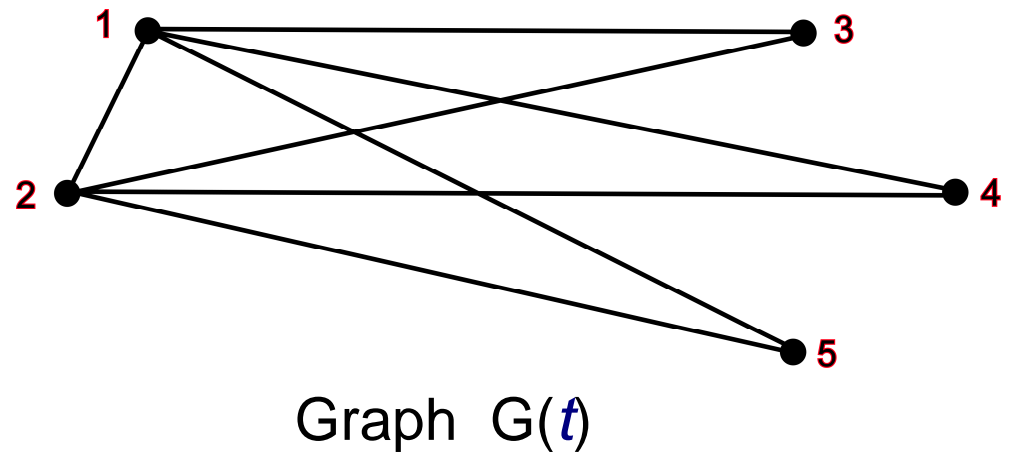
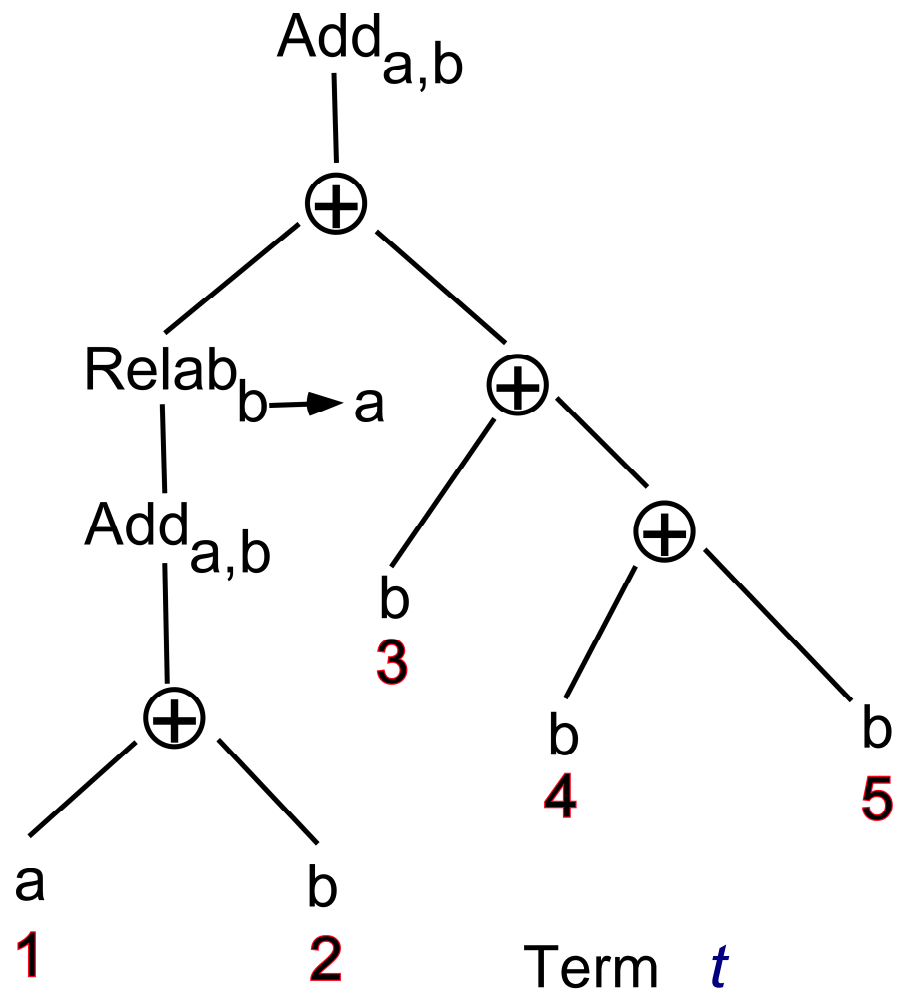
2) a term **s** in $\mathbf{T}(F^{(n)})$ encodes a term **t** in $\mathbf{T}(F)$ and an assignment of sets V_1, \dots, V_n to the set variables X_1, \dots, X_n :

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

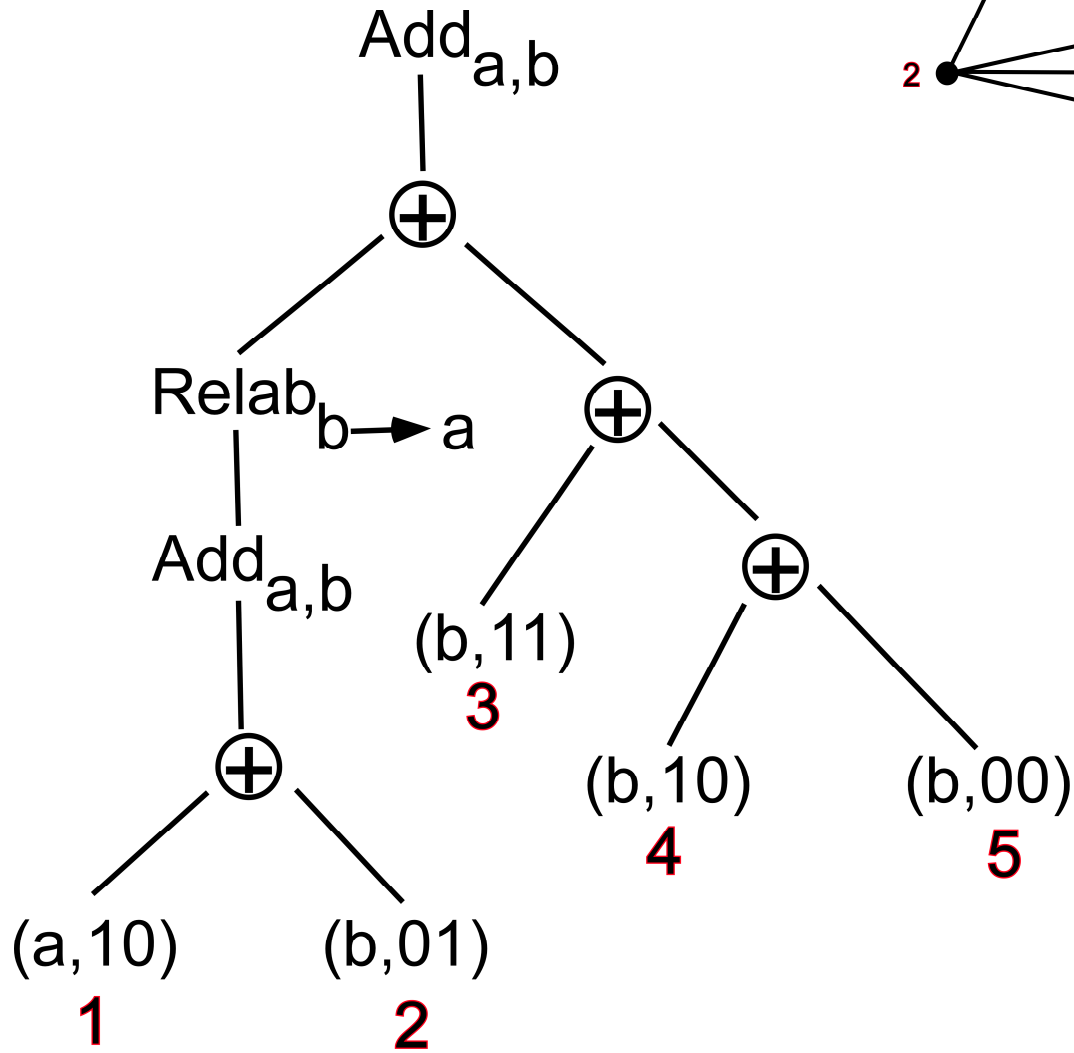
$w_i = 1$ if and only if $\mathbf{u} \in V_i$.

3) **s** is denoted by $\mathbf{t}^*(V_1, \dots, V_n)$

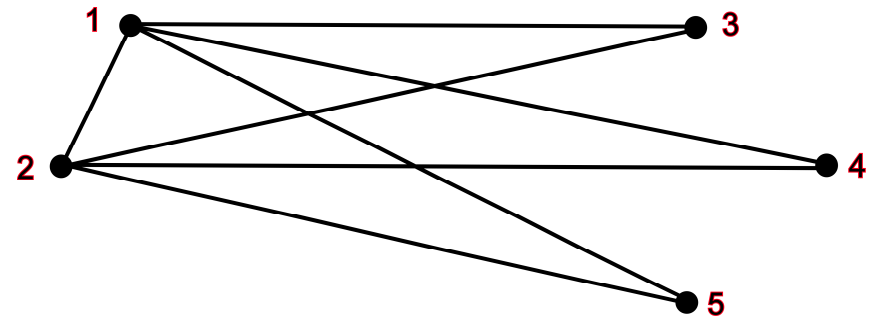
Example



Example (continued)



Term $t^*(V_1, V_2)$



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

By an induction on φ , we construct for each $\varphi(X_1, \dots, X_n)$ an FA $A(\varphi(X_1, \dots, X_n))$ that recognizes:

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

Quantifications: Formulas are written without \forall

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

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

where pr is the *projection* that eliminates the last Boolean;

\rightarrow a *non-deterministic* automaton $B = \text{pr}(A(\varphi(X_1, \dots, X_{n+1})))$.

Determinized runs of **B** by deterministic FA **C**

For $\exists \underline{X}.P(\underline{X})$: the state of **C** at position u is

{ state q of **B** / some run reaches q at position u }

For $\# \underline{X}.P(\underline{X})$: the state of **C** at position u is

{ (q,m) / m = the number of runs that reach q at u }

equivalently, the corresponding multiset of states q , cf. $\exists \underline{X}.P(\underline{X})$

For $\text{Sp} \underline{X}.P(\underline{X})$: the state of **C** at position u is

{ (q,S) / S = the set of tuples of cardinalities of

the “components of \underline{X} below u ” that yield q at u }.

For $\text{MSp} \underline{X}.P(\underline{X})$: S is the corresponding multiset.

For $\text{MinCard } \underline{X}.P(\underline{X})$: the state of \mathbf{C} at position u is

$\{ (q, s) / s = \text{the minimum cardinality of "X below u" that yields } q \text{ at } u \}$.

For $\text{SetVal-}\alpha(\underline{X}) / P(\underline{X})$ where $\alpha(\underline{X})$ is a linear combination of the cardinalities of the components of \underline{X} that satisfy P , we use a variant of $\text{Sp}\underline{X}.P(\underline{X})$.

For $\text{Sat}\underline{X}.P(\underline{X})$: the set of all tuples \underline{X} that satisfy $P(\underline{X})$, the state of \mathbf{C} at position u is $\{ (q, S) / S = \text{the set of all tuples below } u \text{ that yield } q \}$

A common presentation for all these cases:

We call the component s in state (q,s) is an **attribute** of q .

An attribute s of q at u collects certain information about all the runs that yield q at u . Computations of attribute correspond to **variants of the basic determinization**: they use, according to the cases :

Set union (for basic determinization)

Union of multisets, (for counting runs)

Selection of minimal number or minimal set (e.g. for inclusion),

$A + B$ where A and B are sets of numbers,

etc...

Distributive algebras offer a formal setting (see article to appear).

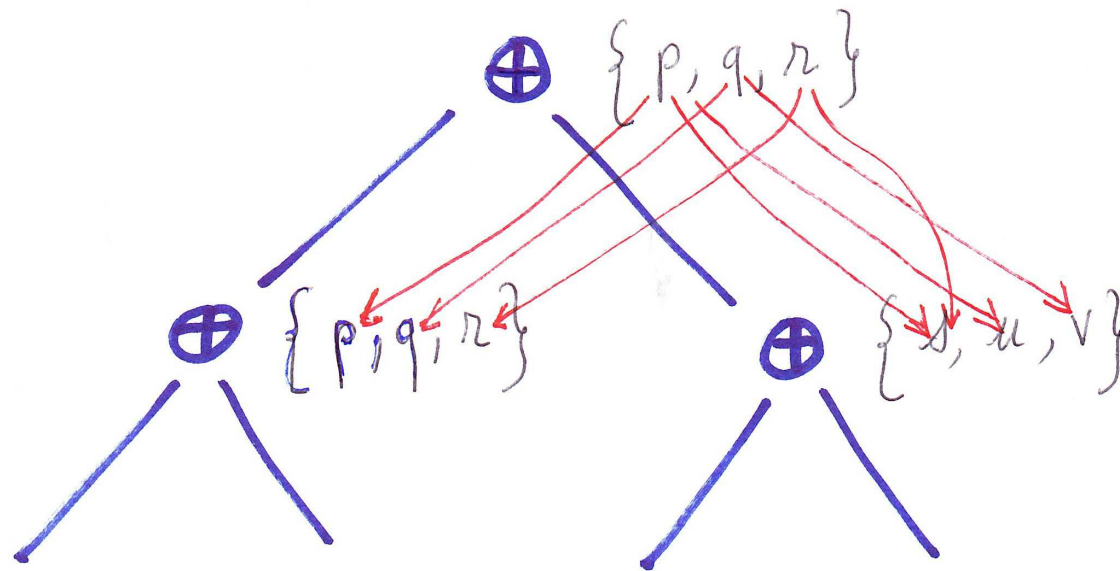
Optimizations : How to avoid intermediate computations
that do not contribute to the final result.

Theorem (Flum and Grohe) : One can compute $\text{Sat}_{\underline{X}}.P(\underline{X})$ in time $f(k).(n + \text{size of the result})$ where $\text{cwd}(G) \leq k$ and n is the size of the term.

The bottom-up inductive computation must “know” that certain states will not belong to any accepting run on the considered term.

Method : 3 pass algorithm

- 1 : **determinized bottom-up run** keeping pointers showing how states are obtained from others,
- 2 : **top-down run** starting from the accepting states at the root and marking the *useful states*,
- 3 : **bottom-up computation of attributes** only for the useful states.



$$\oplus[p, s] \rightarrow p$$

$$\oplus[r, s] \rightarrow r$$

$$\oplus[p, u] \rightarrow p$$

$$\oplus[q, v] \rightarrow q$$

This 3-pass algorithm is applicable for all our computations of attributes.

Example : Checking that a graph has a **unique 3-coloring**.

1st method : expressing that in MSO : possible but cumbersome.

2nd method : computing the total number of 3-colorings: we want result 6 (assume the graph is not 2-colorable) : OK but lengthy.

3rd method : “optimized” counting with reporting **Failure** if a useful intermediate result more than 6 is found.

This is applicable to : $\exists! \underline{X}.P(\underline{X})$ for every MSO property P.

Probabilities

Let $P(\underline{X})$ be an MSO property.

Assume that each vertex of G that is created by nullary symbol a is put in component X_i of \underline{X} with probability $p_{a,i}$.

We can compute the probability that \underline{X} satisfies $P(\underline{X})$.

We can also compute the polynomial in the indeterminates $p_{a,i}$ that gives for G this probability (either factorized or developed, obtained easily from $\text{Sat}_{\underline{X}}.P(\underline{X})$; **to be explored!**).

oOo

Thanks for any suggestion of application, variant or related question!

Conclusion

By uniform constructions, we get **XP** or **FPT** dynamic programming algorithms (that could be obtained independently). They are based on **fly-automata**, that can be quickly constructed from logical descriptions → *flexibility*.

These constructions are applicable to **bounded tree-width** and **MSO with set quantifications** by means of incidence graphs.

They are **implemented**. Tests have been made for colorability and connectedness problems.

Enumeration is our next theoretical and practical topic.

Also optimizations (**annotations**) will be investigated.

Appendix (if anybody wants)

Examples of MSO definability : G is 3-colorable :

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

G is not connected :

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

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