Enumeration algorithms based on *fly-automata*

Bruno Courcelle

(joint work with Irène Durand)

Bordeaux University, LaBRI (CNRS laboratory)

Overview

Enumeration problem 1 : listing graphs : for example, the forbidden minors for some minor closed class.

Enumeration problem 2 : listing configurations in a given graph : for example, its p-colorings or its maximal induced planar subgraphs.

We only consider problems of type 2 for configurations defined in monadic second-order (MSO) logic and graphs of bounded tree-width or bounded clique-width. Graphs are defined by algebraic terms and processed by automata on these terms.

Our graph parameter is *clique-width* and the terms denoting graphs are those from which clique-width is defined because :

- it is easier to handle than (the very popular) tree-width for constructing automata, and more powerful : bounded tree-width implies bounded clique-width,
- it is defined in terms of elementary graph operations, hence is easier than the equivalent notion of rank-width,
- it works equally well on directed graphs.
- it can handle edge quantification via incidence graphs.

The system AUTOGRAPH (by Irène Durand) and the corresponding theory [B.C.&I.D.: *Automata for the verification of monadic second-order graph properties*, J. Applied Logic, 10 (2012) 368-409] are based on clique-width.

Theorem [B.C.]: For every k, every MSO graph property P can be checked by a *finite* automaton. This automaton recognizes the terms that: (1) are written over the finite set of operations that generate the graphs of *clique-width* at most k, and (2) define a graph satisfying P.

However, these automata are much too large to be tabulated.

Remedy: We use fly-automata (in French "automates programmés") whose states and transitions are described and not tabulated. Only the transitions necessary for an input term are computed, "on the fly".

As states are not listed, a fly-automaton can use an infinite set of states. It can recognize sets of words or terms that are not monadic second-order definable : the language $a^n b^n$, the set of terms of arbitrary clique-width that define regular graphs (all vertices of same degree).

It can also compute values: the number of p-colorings or of "acyclic" p-colorings of a graph (the graph induced by any two color classes is acyclic).

We can construct fly-automata in uniform ways from logical formulas. In this way, we develop a *theory* of (some aspects of) *dynamic programming.*

Review of definitions

Definition 1 : Monadic Second-Order Logic

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

MSO (expressible) properties : transitive closure, properties of paths,
connectedness, planarity (via Kuratowski), p-colorability.

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

G is 3-colorable :

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

G is not connected :

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

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

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

where "Z is R-closed" is defined by :

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

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

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

expressing that G[Y] is connected (Y is free in R).

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

Non-MSO-expressible properties

- G is *isomorphic to* $K_{p,p}$ for some p (p is *not fixed*; needs equipotence of two sets, hence quantification over binary relations to find if there is a bijection).

- G has a *nontrivial automorphism*, or is *regular* (has all vertices of same degree).

Reference : B.C. & J. Engelfriet *: Graph structure and monadic second-order logic,* Cambridge University Press, 2012

Definition 2 : Clique-width

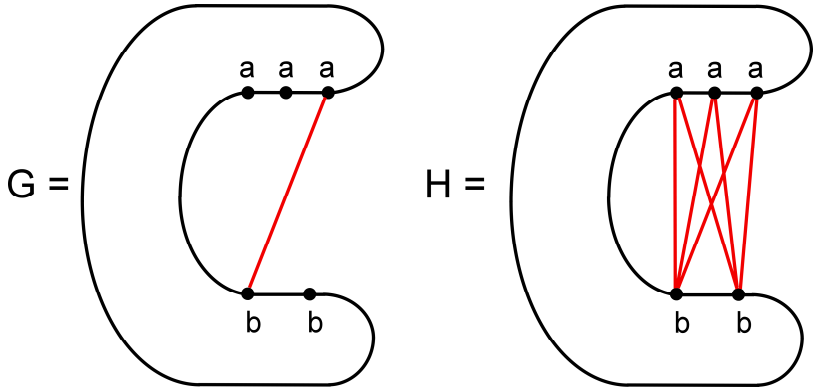
Defined from graph operations. Graphs are simple, directed or not, and labelled by *a* , *b* , *c, ...* . A vertex labelled by *a* is called an *a-vertex*.

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

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

$Add_{a,b}(G)$ is G augmented with undirected edges between every *a*-vertex and every *b*-vertex.

The number of added edges depends on the argument graph.



H = $Add_{a,b}(G)$; only 5 new edges added

Directed edges can be defined similarly.

Vertex relabellings :

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

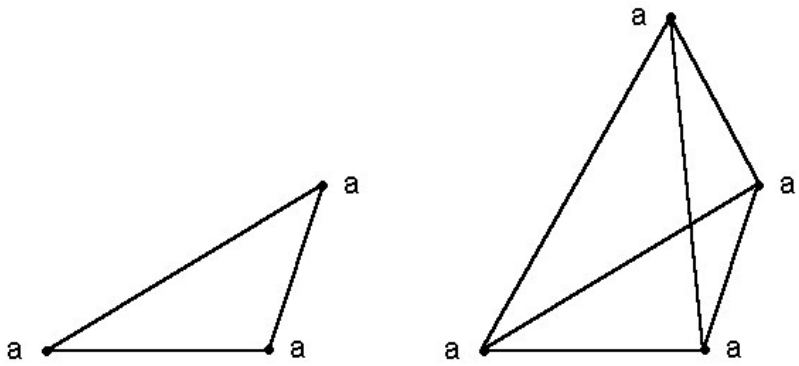
Basic graphs : those with a single vertex **a**, labelled by *a*.

Definition: A graph G has clique-width $\leq$ k (denoted by cwd(G))

$\Leftrightarrow$ G = G(t), defined by a term t using $\leq$ k labels.

Example : Cliques have clique-width 2.

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



The *parsing problem*: construction of terms, i.e., of decompositions.

As automata take terms as inputs, the parsing must be done before. Deciding if cwd(G) ≤ k (for input (G,k)) is NP-complete (same for tree-width).

There are FPT approximation algorithms, taking time $f(k).n^3$, that output the following, for given k and graph G with n vertices:

(i) either the answer that cwd(G) > k,

(ii) or a term witnessing that cwd(G) ≤ g(k).

Every FPT algorithm taking terms as inputs can be converted into an FPT algorithm taking graphs as inputs.

New definition 3 : Fly-automaton (FA)

A = < F, Q, δ, Out >

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

Q : finite or countable (effective) set of states (integers, pairs of integers, finite sets of integers: states can be encoded as finite words, integers in binary),

Out : Q → D (an effective domain, i.e., set of finite words), computable.

δ : computable (bottom-up) transition function

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

This automaton defines a computable function : T(F) → *D*

(or : T(F) → *P*(*D*) if it is not deterministic)

If *D* = { *True, False* }, it defines a decidable property, equivalently,

a decidable subset of T(F).

Deterministic computation of a nondeterministic FA :

bottom-up computation of *finite* sets of states (classical simulation of the determinized automaton): these states are the useful ones of the *determinized automaton*; these sets are *finite* because the transition function is finitely multivalued.

To be defined later : *Enumerating computation*

Example : The number of accepting runs of a nondeterministic automaton.

Let $A = < F, Q, \delta, Acc >$ be finite, nondeterministic.

Then $\#A := < F, [Q \rightarrow \mathbf{N}], \delta^{\#}, Out >$

$[Q \rightarrow \mathbf{N}]$ = the set of total functions : $Q \rightarrow \mathbf{N}$

$\delta^{\#}$ is easy to define such that the state reached at position

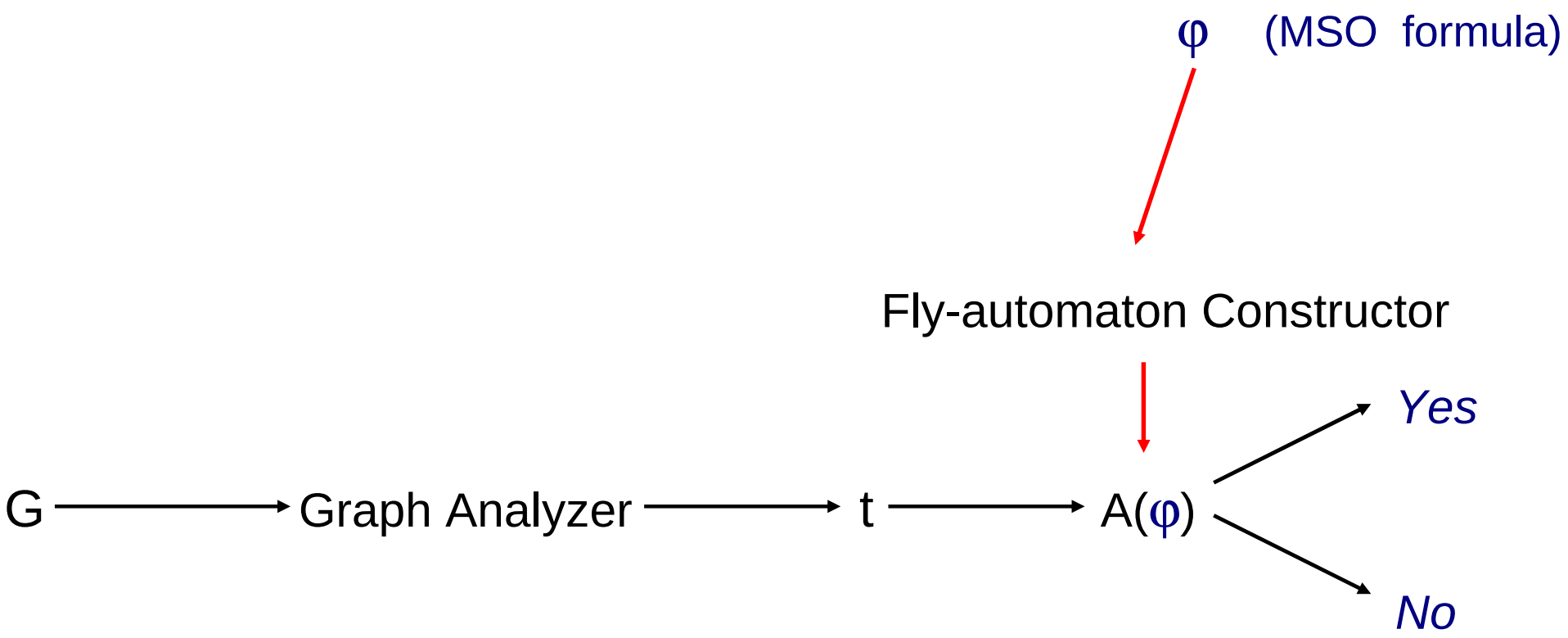
u in the input term is the function σ such that $\sigma(q)$ is

the number of runs reaching q at u.

$Out(\sigma)$ is the sum of $\sigma(q)$ for q in Acc.

$\#A$ is a fly-automaton obtained by a generic construction that extends to the case of an infinite fly-automaton A.

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



A(φ): an *infinite fly-automaton* over the countable set F of all graph operations that define clique-width. The time taken by A(φ) depends on t (on the number of labels that occur in this term), not only on the size of G.

Fly-automata that check graph properties

How to construct them ?

(1) Direct construction for a well-understood graph property or

(2) Inductive construction based on the structure of an MSO formula;

a direct construction is anyway needed for atomic formulas;

logical connectives are handled by transformations of automata :

products, projection (making them nondeterministic), determinization

(for negation).

Direct construction : 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.*

We can allow fly-automata with *infinitely* many states and, also, with *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 }.

Computation time of a fly-automaton

F : all graph operations, F_k : those using labels 1, …, k.

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, respectively, polynomial-time, FPT or XP for clique-width as parameter.

Recognizability Theorem [B.C & I.D.] : For each MSO property P, one can construct a single infinite FPT-FA over F that recognizes the terms $t \in T(F)$ such that P(G(t)) holds.

For each k, its restriction to the finite signature F_k is a finite automaton.

Consequences : (1) The same automaton (the same model-checking program) can be used for graphs of any clique-width.

(2) Can be implemented in non-trivial cases.

Some experiments using FA (by Irène Durand)

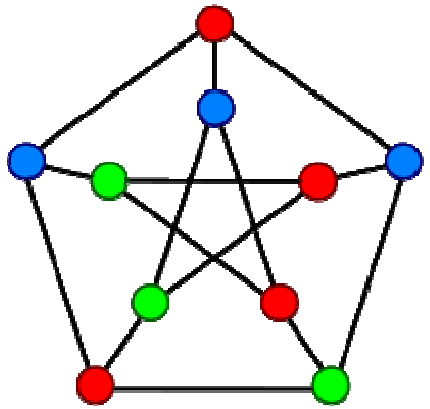
Number of 3-colorings of the 6 x 90 "modified" grid of clique-width 8 in 1 minute 9 seconds (with diagonals on the squares of the first column). For the similar 6 x 250 grid : < 6 minutes; for 6 x 360 : < 9 minutes.

4-*acyclic*-colorability of the Petersen graph (clique-width 5) in 1.5 minutes, from a term in $T(F_6)$.

(3-colorable but not acyclically;

red and **green** vertices

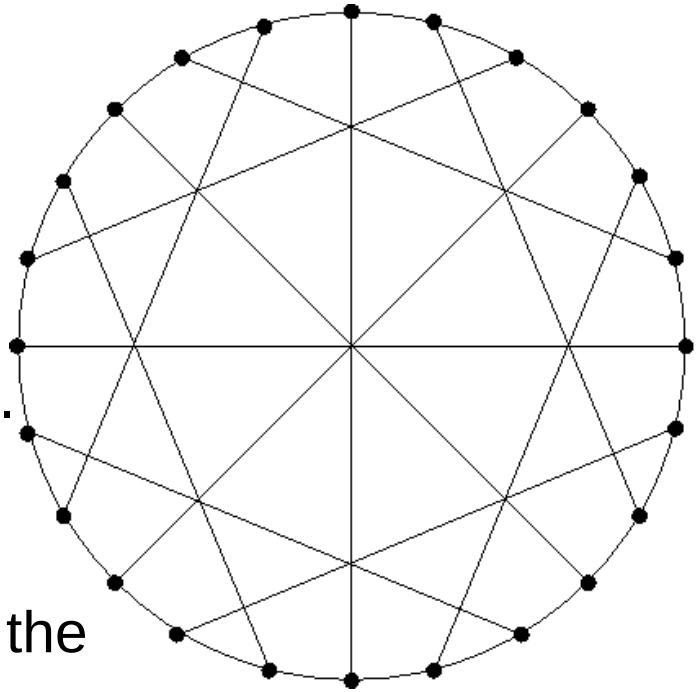
induce a cycle).



The McGee graph (of clique-width 8)

is defined by a term in $T(F_{10})$

of size 99 and depth 76.



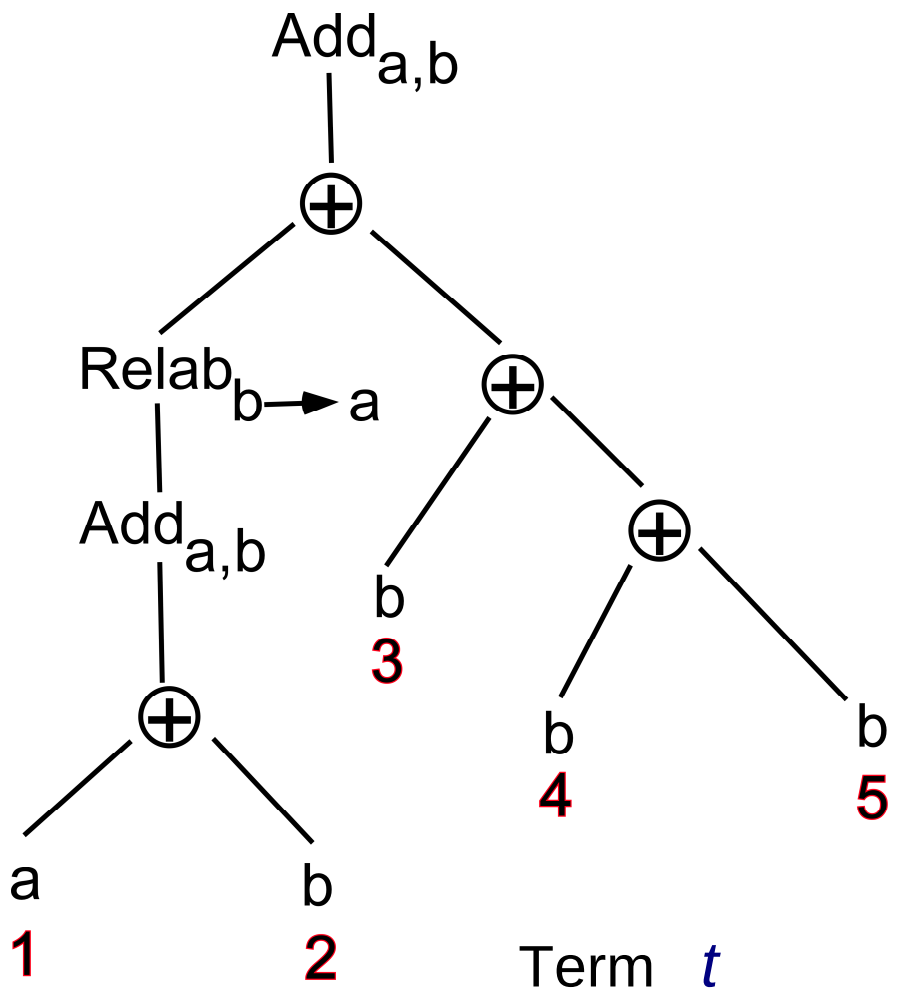
This graph is 3-acyclically colourable.

- checked in 40 minutes.

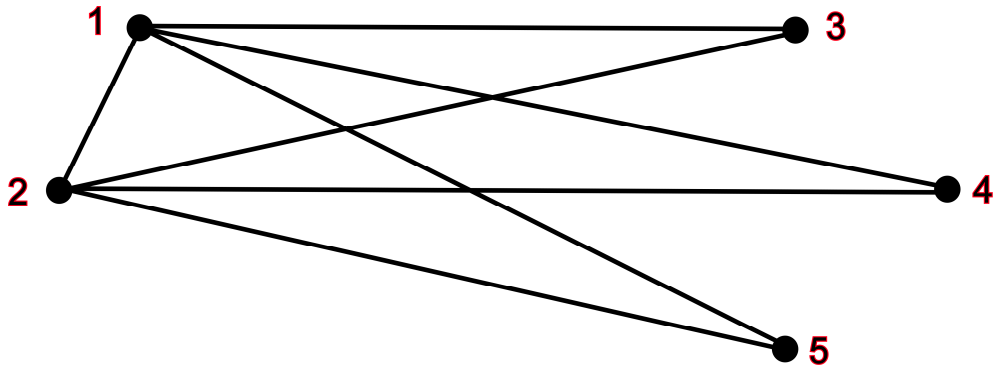
- even in 2 seconds by *enumerating* the

runs, and stopping as soon as an accepting one is found.

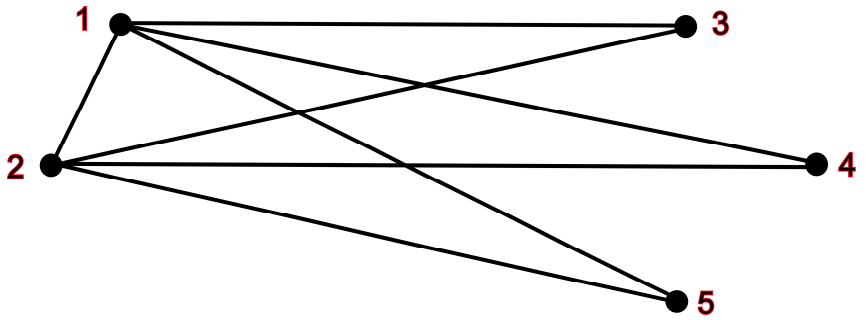
Existential quantifications and nondeterminism

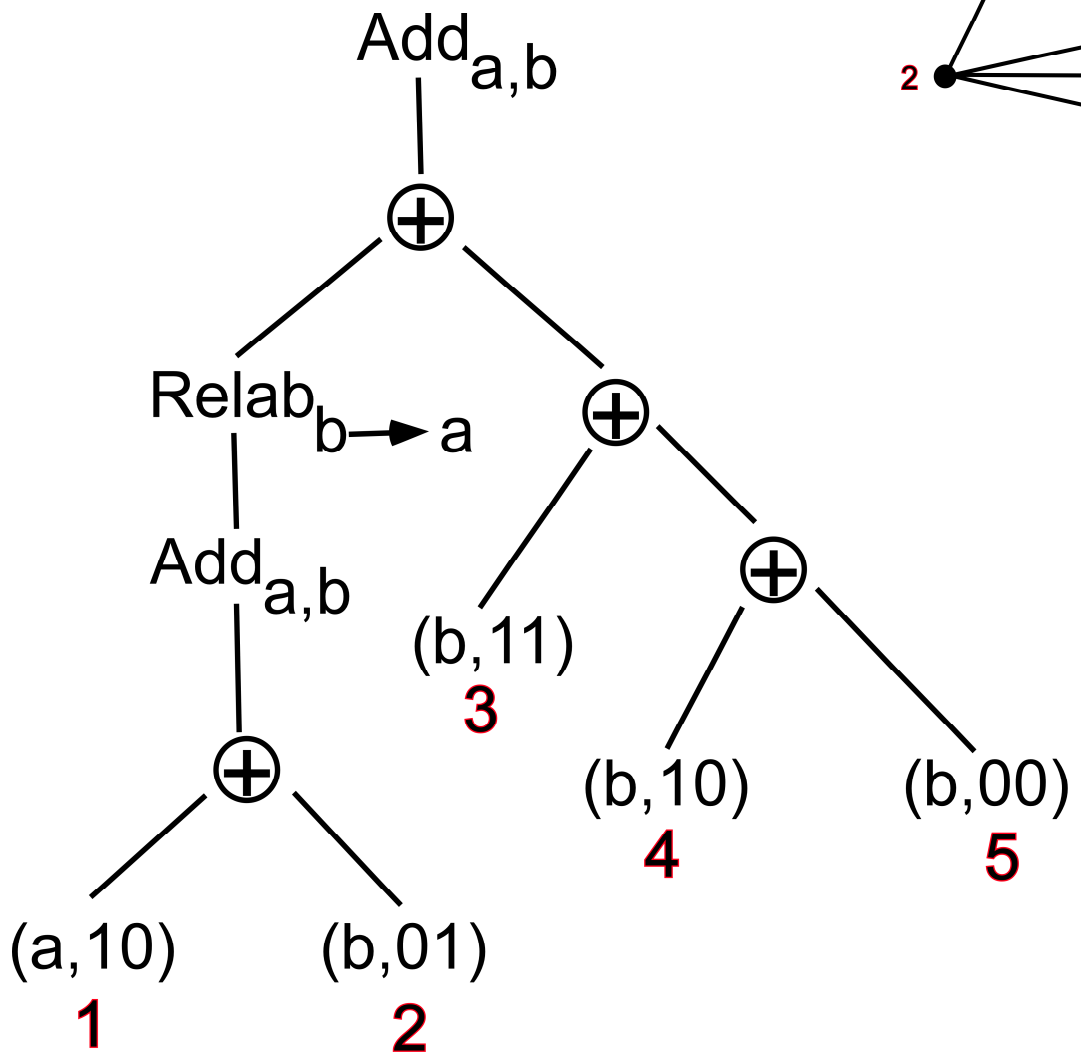


Term t



Graph G(t)





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

Term $t * (V_1, V_2)$

Consider a property $\exists X,Y.\ \varphi(X,Y)$ to be checked on graph G(t).

We construct a deterministic automaton A over $F^{[2]}$ recognizing the terms $t*(X,Y)$ such that $G(t*(X,Y)) \mathrel{!}= \varphi(X,Y)$.

We delete the Booleans in the nullary symbols of $F^{[2]}$: we obtain a nondeterministic automaton B over F (called a projection : A → B).

The different runs of B correspond to trying the different possible pairs (X,Y) when looking for a satisfying one.

B recognizes the terms t such that $G(t) \mathrel{!}= \exists X,Y.\ \varphi(X,Y)$.

Enumeration techniques

Enumeration of accepting states

- stopping as soon as one is obtained
- less space but more time for checking negation (failure to recognition),
- listing the assignments $\underline{X}$ satisfying $\varphi(\underline{X})$: we maintain with each state, at each position, its “origin”: the partial assignment that produced it.

If an Error state is found in a partially constructed run, we abort its completion.

Enumerators

An *enumerator* is a triple E = (D, *reset, next*) where D is an effective (countable) set, *reset* and *next* are two programs guaranteed to terminate. E defines a finite list List(E) of elements of D.

List(E) may contain repetitions,

next produces one more element or reports “end of list”,

reset reinitializes the program *next.*

Remark : Enumerators can be extended to produce infinite lists.

Basic enumerators : For each nullary a, E_a produces the list of states q (not Error) arising from a (by the nondeterministic automaton B that checks $\exists \underline{X}.\varphi(\underline{X})$ and that is obtained from the deterministic automaton A checking $\varphi(\underline{X})$, by deleting the sequences of Booleans w in the nullary symbols (a,w) of the signature $F^{[p]}$ of A).

Alternatively, $\underline{E}_a$ produces the list of pairs (q,w) : we keep track of the w that produced q (its “origin”).

Transforming and combining enumerators.

Making a copy of E : $copy_u(E)$ indexed by u, a position of the given term.

Making E into nr(E), nonredundant: produces the same elements without repetitions (nr(E) uses the list of already generated elements).

Applying a unary function h : D → D'

If E enumerates elements of D, then $h \circ E$ produces the images by h of the elements of List(E).

Cartesian product.

If E enumerates elements of D, E' elements of D', we want to list the pairs (d,d') where $d \in \text{List}(E) = d_1, \ldots,\ d' \in \text{List}(E') = d'_1, \ldots$

Possible orders:

"Line order" (lexico) : $(d_1, d'_1), (d_1, d'_2), \ldots, (d_2, d'_1), (d_2, d'_2), \ldots$

"Column order" : $(d_1, d'_1), (d_2, d'_1), \ldots, (d_1, d'_2), (d_2, d'_2), \ldots$

"Diagonal order" : $(d_1, d'_1),\ (d_1, d'_2), (d_2, d'_1),$

$(d_1, d'_3), (d_2, d'_2), (d_3, d'_1), \ldots..$

$E\ x_{Line}\ E'$, $E\ x_{Col}\ E'$, $E\ x_{Diag}\ E'$ realize these enumerations.

Given a term t and an automaton A that checks $\varphi(\underline{X})$, one builds a (big) enumerator E_t by combining basic ones with Cartesian compositions, $h_O(.)$ and possibly nr(.).

If t=f(s), then $E_t = h_O(E_s)$ where h is based on transitions for f.

If t=f(s,s'), then $E_t = h_O(E_s \times E_{s'})$ where h is similar.

Running E_t by calling its *next* component iteratively produces the desired list (unless the system lacks of memory).

Weighted enumeration.

Each element d of D has a weight (a size) s(d).

An s-enumerator E produces a list d_1, d_2, … such that $s(d_1) \leq s(d_2) \leq s(d_3) \leq \ldots$.

For Cartesian product, if E s-enumerates elements of D and E' s'-enumerates those of D', we want an s"-enumerator of the pairs (d,d') where s"(d,d') := s(d) + s'(d').

A modification of the diagonal construction can do that.

Application : Enumeration of the tuples $\underline{X}$ that satisfy $\varphi(\underline{X})$ by increasing order of size, where the size is the sum of cardinalities of the sets composing $\underline{X}$.

The system AUTOGRAPH (by I. Durand)

(1) Fly-automata for basic graph properties :

Clique, Stable (no edge), Link(X,Y), NoCycle,

Connectedness, *Regularity*, Partition(X, Y, Z), etc…

and functions :

#Link(X,Y)= number of edges between X and Y,

Maximum degree.

Procedures for combining fly-automata, corresponding to logical constructions : $\wedge$, $\vee$, negation, $\exists \underline{X}.\ \varphi(\underline{X})$.

Procedures to build automata that compute functions:

#$\underline{X}.\varphi(\underline{X})$: the number of tuples $\underline{X}$ that satisfy $\varphi(\underline{X})$ in the input term (hence, in the associated graph),

Sp$\underline{X}.\varphi(\underline{X})$: the spectrum = the set of tuples of cardinalities of the components of the $\underline{X}$ that satisfy $\varphi(\underline{X})$, etc…

Enumeration: construction of an enumerator from a term and a fly-automaton.

These constructions are "uniform" with respect to the input automata.

Some tests

Checking colorability of grids 6 x M of clique-width 8.

M	2-col. det	2-col. enum	3-col. det	3-col. enum
7	0.03 s	6 s	10 s	6 s
8	0.03 s	9 s	Fails	9 s
20	0.2 s	3 min	Fails	3 min

Counting 2-colorings : for M = 200, in 2 seconds (2).

Counting 3-colorings : for M= 5, in 3 seconds (6 204 438).

Fails for M = 6.

Works for M = 360 for modified grids.

Enumerating 3-colorings :

M = 20 : Construction of enumerator in 3 minutes

Then, first result in 0.5 second.

Edge quantifications and tree-width

If $G = (V_G , edg_G(.,.))$, then Inc(G) := $(V_G \cup E_G , inc_G(.,.))$ is its *incidence graph*: $inc_G(u,e) \Leftrightarrow$ u is an end of e.

MSO formulas over Inc(G) can use quantifications on edges and thus, express more properties.

Proposition (T.Bouvier) : tree-width(G) $\leq$ k $\Rightarrow$ cwd(Inc(G)) $\leq$ k+3.

k+3 improves a previous exponential upper bound. Hence tools for "bounded clique-width" and MSO formulas are applicable to "bounded tree-width" and MSO formulas using edge quantifications.

As tree-width(G) = O(cwd(Inc(G)), incidence graphs "only work" for bounded tree-width

Another construction based on automata

Theorem [B.C]: Given $\varphi(\underline{X})$ and a term t of width k, after a preprocessing taking time O(n.log(n)), one can enumerate with linear delay the tuples $\underline{X}$ that satisfy φ in G(t) (having n vertices).

See : Linear delay enumeration and MSO logic, DAM 157 (2009)

We build from t and A a directed acyclic graph D(t,A) that embeds all runs of B on t. We enumerate with linear delay the accepting runs from D(t,A). The runs contains the tuples they come from, so we can get them.

To have linear delay in the *sizes of the output tuples*, we eliminate the parts of D(t,A) whose runs come from empty tuples, because traversing these parts takes time that does not correspond to an increase of the produced tuples.

We use a term t of height $O(\log(n))$, that is not necessarily optimal for width.

This method has not been implemented.

Conclusion

These algorithms are based on fly-automata, that can be quickly constructed from logical descriptions (and basic automata)

→ flexibility.

The system AUTOGRAPH implements these constructions. Tests have been made for colorability and connectedness problems.

Thank you for suggesting interesting problems that could fit in this framework.