Graph algorithms based on infinite automata: logical descriptions and usable constructions

Bruno Courcelle

(joint work with Irène Durand)

Bordeaux-1 University, LaBRI (CNRS laboratory)

Overview

Algorithmic meta-theorems provide existence proofs of relatively efficient (FPT or XP) graph algorithms from logical descriptions of (difficult) problems. We give *usable* and *tested* constructions based on:

- problem descriptions in extensions of MSO (Monadic Second-Order) logic,
- hierarchical decompositions of graphs,
- automata with *infinitely many states.*

Our graph parameter is *clique-width*, 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 easily on directed graphs.

The system AUTOGRAPH (by I.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 : For each k, every MSO graph property P can be checked by a finite automaton that recognizes terms over the finite set of operations that generates the graphs of *clique-width* at most k.

This automaton is computable (i.e., “one can compute” its set of states and its transition table) in theory but not in practice because it is much too large as soon as $k \geq 3$. To overcome this difficulty, we have introduced 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 and so, it can recognize sets of words, terms or graphs that are not monadic second-order definable : the language $a^n b^n$, the set of 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.

We construct fly-automata in uniform ways from logical descriptions of problems. We develop a *theory* of (some aspects of) *dynamic programming.*

Definition (skip ?) : 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 (skip ?) : Clique-width

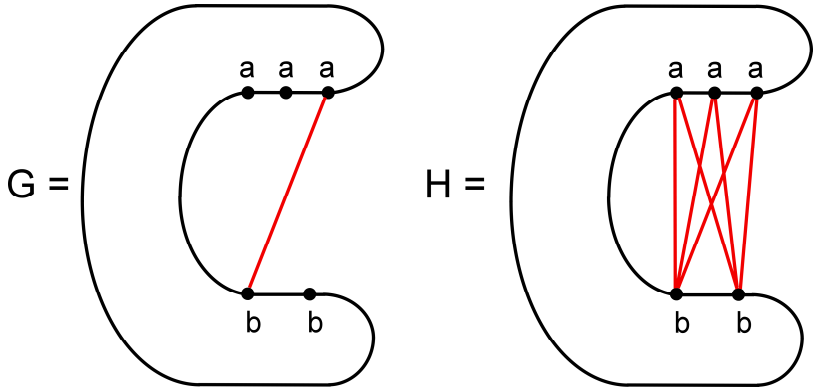
It is defined from graph operations. Graphs are simple, directed or not, and labelled by *a* , *b* , *c, ...* . A vertex labelled by *a* is called an *a-vertex*.

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 :

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

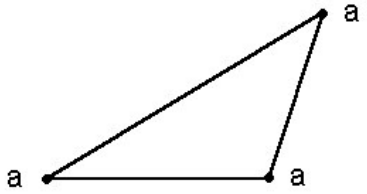
Basic graphs : those with a single vertex **a**, labelled by *a*.

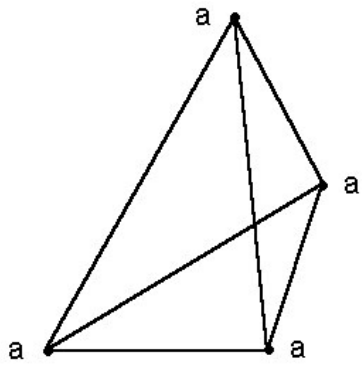
Definition: A graph G has clique-width (denoted by cwd(G)) $\leq$ k

$\Leftrightarrow$ G=G(t) is 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} = $Relab_{b \longrightarrow a}(Add_{a,b} (t_n \oplus \mathbf{b}))$





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

Automata take terms as inputs, not graphs : the parsing must be done before. (Graph automata do not exist in any satisfactory way).

Deciding if $cwd(G) \leq k$ (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 $cwd(G) > k$,

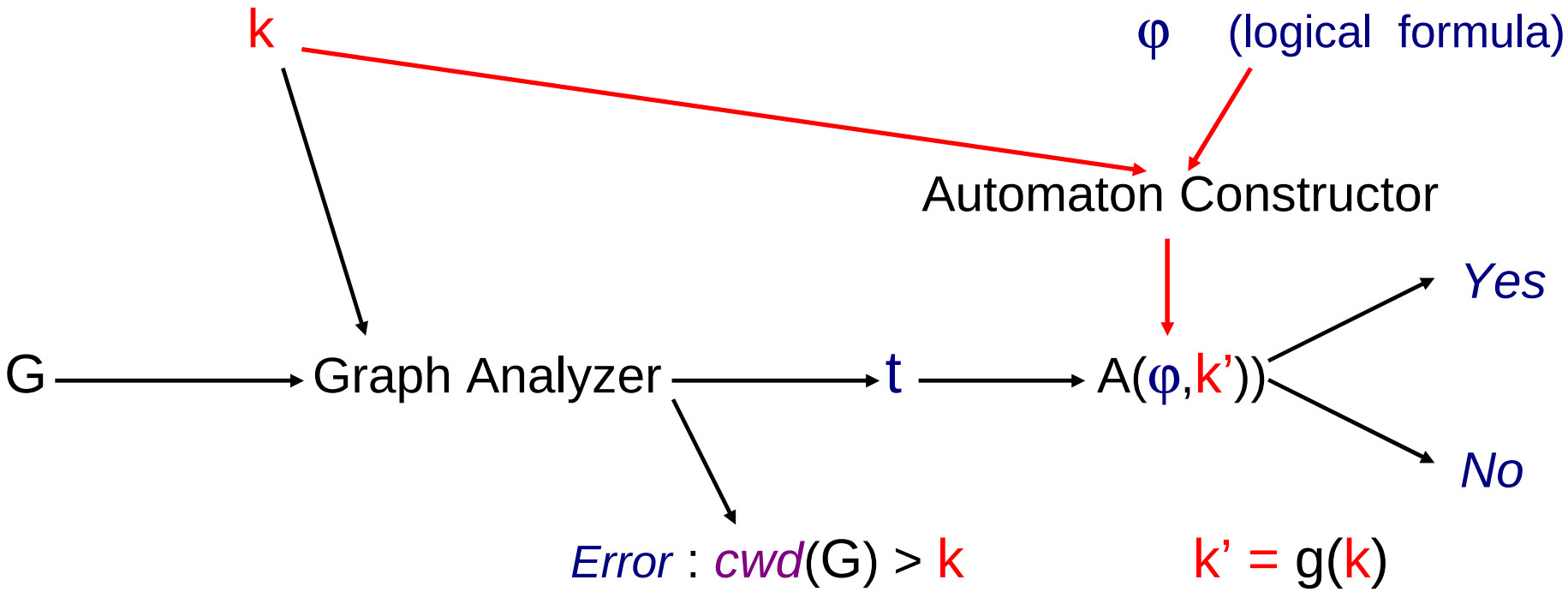
(ii) or a term witnessing that $cwd(G) \leq g(k)$.

Every FPT algorithm taking terms as inputs can be converted into an FPT algorithm taking graphs as inputs. (It parses and then checks).

However, graphs arising from concrete problems are not random. They usually have "natural" hierarchical decompositions from which terms of small tree-width or clique-width are not hard to construct.

This situation arises in compilation (flow-graphs of structured programs), in linguistics and in chemistry. It is thus interesting to develop specific parsing algorithms for graph classes relevant to particular applications.

The MSO meta-theorem through *finite* automata: the basic scheme

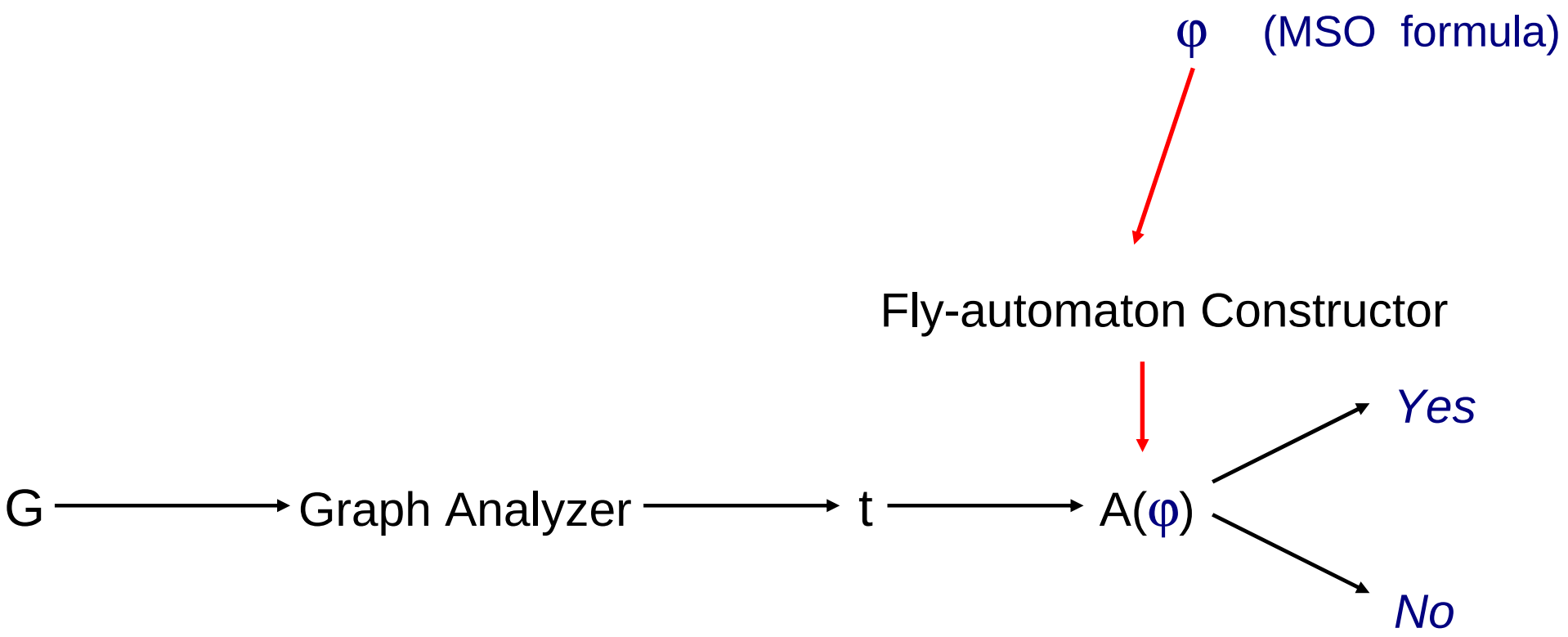


Steps ⟶ are done "once for all", independently of G

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

cwd can be replaced by tree-width or rank-width.

The MSO meta-theorem through *fly-automata*: a simpler scheme



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

Automata that check graph properties

We want to check a property P(G), for G = G(t), t $\in$ T(F).

For each *labelled* graph G, we define a piece of information q(G) that encodes properties of G and values attached to G, so that we have:

(i) inductive behaviour of q : for f $\in$ F and graphs G,H:

$$q(f(G,H)) = f^q(q(G), q(H))$$

for some computable function f^q .

(ii) P(G) can be decided from q(G).

Then q(G(t/u)) is computed bottom-up in a term t, for each node u. This information is relative to the graph G(t/u) (a subgraph of G) defined by the subterm t/u of t issued from u.

q(G(t/u)) is a state of a *finite or infinite deterministic bottom-up automaton*.

These automata formalize some form of dynamic programming.

Two possibilities: Direct construction for a well-understood graph property or automatic construction from an MSO formula.

Direct construction 1 : Connectedness.

The state at node u 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 the 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.*

In a *fly-automaton,* states and transitions are *computed* and not tabulated. 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 }.

Direct construction 2 : Regularity (not MSO)

A state is a tuple of counters that indicates, for each label *a*:

the number of *a*-vertices and

the common degree of all *a*-vertices.

The state is *Error* if two *a*-vertices have different degrees: the edge addition operations will add the same numbers of edges to these vertices, hence the considered graph cannot be regular.

Definition : 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.

Fly-automata are "implicitly determinized" and they run deterministically

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.

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 finite.

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

Some experiments using FA (by Irène Durand)

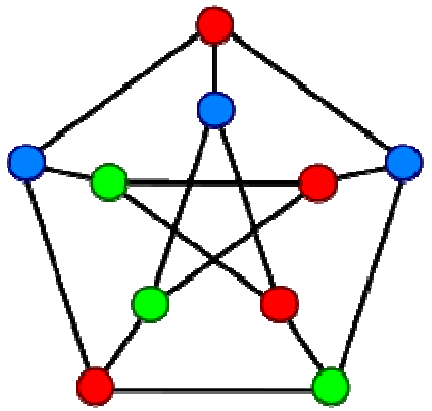
Number of 3-colorings of the 6 x 525 grid (of clique-width 8) in 10 minutes.

4-acyclic-colorability of the Petersen graph (clique-width 6) in 1.5 minutes.

(3-colorable but not acyclically;

red and **green** vertices

induce a cycle).

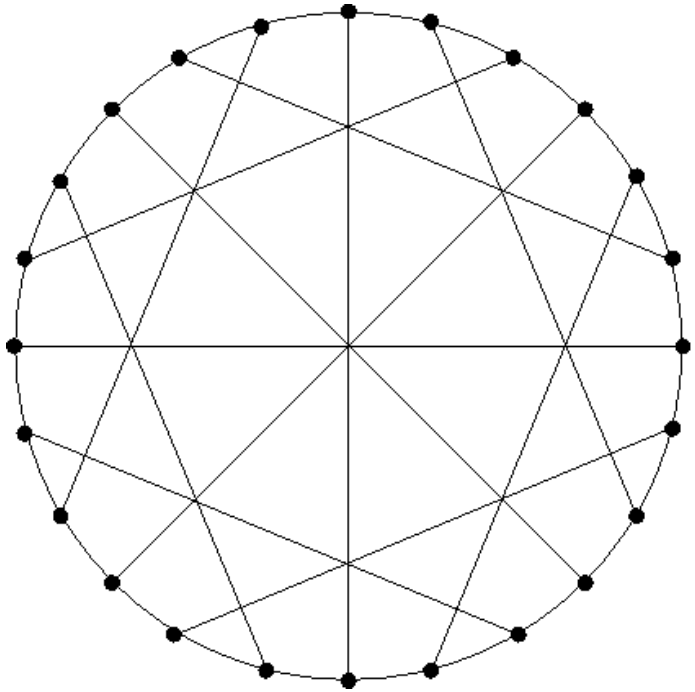


The McGee graph

is defined by a term

with 10 labels (optimal?)

of size 99 and depth 76.



It is 3-acyclically colourable.

Checked in 40 minutes.

(Even in 2 seconds by enumerating the accepting runs,

and stopping as soon as a success is found).

Constructions of automata

(A) For Boolean and first-order constructions of properties and functions.

$P \wedge Q$, $P \vee Q$, $\neg P$, $g(\alpha_1, \ldots, \alpha_p)$ where g is poly-time computable (can be a relation such as a comparison of numbers),

$P[X \cap Y]$: property of subgraph induced on $X \cap Y$ (*set term*)

First-order (FO) quantifications : $\exists \underline{x}.P(\underline{x})$, $\underline{x}$ =tuple of FO var.

Set of satisfying assignments : Sat $\underline{x}.P(\underline{x})$ (*a query*)

Number of satisfying assignments : $\# \underline{x}.P(\underline{x})$.

Set of values $\alpha(\underline{x})$ such that $P(\underline{x})$ is true.

Type of automata	Finite	P-FA	FPT-FA	XP-FA
$P \wedge Q, P \vee Q, \neg P, P[X \cap Y]$	Finite	P	FPT	XP
$g(\alpha_1, \ldots, \alpha_p), \alpha(X \cap Y)$		P	FPT	XP
$\exists \underline{x}.P(\underline{x}), \forall \underline{x}.P(\underline{x})$	Finite	P	FPT	XP
Sat $\underline{x}.P(\underline{x})$, # $\underline{x}.P(\underline{x})$	P	P	FPT	XP
SetVal $\alpha(\underline{x})$ / $P(\underline{x})$)		P	FPT	XP

Finite : Finite signature and sets of states.

We have “nice preservations” of the types of automata.

Main proof ideas : (generalized) existential quantifications :

$\exists\, \underline{x}.P(\underline{x})$, $\#\, \underline{x}.P(\underline{x})$, Sat $\underline{x}.P(\underline{x})$,

where $\underline{x}$ is a p-tuple of first-order variables.

Handling existential quantifications and sets of satisfying assignments needs to transform a deterministic automaton A for $P(\underline{x})$ into a nondeterministic one B (that decides $\exists\, \underline{x}.P(\underline{x})$). But, its *nondeterminism degree* is $\leq n^p$, hence *polynomially bounded* in the number n of vertices.

We get a P-FA, an FPT-FA or an XP-FA if A is so.

Same idea for Sat $\underline{x}.P(\underline{x})$. For $\#\, \underline{x}.P(\underline{x})$, we transform B into a *deterministic FA* that counts the number of its accepting runs.

(B) Monadic second-order constructions

The spectrum Sp$\underline{X}$.P($\underline{X}$) of a property P($\underline{X}$) is the set of tuples of cardinalities of the components of the tuples $\underline{X}$ that satisfy P($\underline{X}$).

The multispectrum MSp$\underline{X}$.P($\underline{X}$) is the corresponding multiset of tuples of Sp$\underline{X}$.P($\underline{X}$) hence, for $\underline{X}$ = X (one component):

the set of pairs (m ,i) such that i > 0 is

the number of sets X of cardinality m that satisfy P(X).

For a p-tuple $\underline{X}$, a multispectrum is a function $[\,0,n\,]^p \rightarrow [\,0,2^{p.n}\,]$;

it can be encoded in size $O(n^p.\log(2^{p.n})\,) = O(n^{p+1})$.

Type of automaton A for P($\underline{X}$)	Finite	P-FA	FPT-FA	XP-FA
$\exists \underline{X}$.P($\underline{X}$), $\forall \underline{X}$.P($\underline{X}$)	Finite	P	FPT	XP
MSp $\underline{X}$.P($\underline{X}$), Sp $\underline{X}$.P($\underline{X}$), # $\underline{X}$.P($\underline{X}$), MaxCard X.P(X)	P	P	FPT	XP

P-FA means : A is P-FA and B (the associated nondeterministic FA) has a polynomially bounded nondeterminism degree. Similar for FPT and XP (with FPT- or XP-bounded nondeterminism degree). There are more contraints for the preservation of types of automata than for FO constructions.

Note: the automata are constructed in the same way in all cases; the nondeterminism degree concerns only the computation time.

Some non-MSO examples : (1) *Equitable p-coloring* :

$\exists X_1,\ldots,X_p$ (Partition($X_1,\ldots,X_p$) $\wedge$ Stable[X_1] $\wedge$...$\wedge$ Stable[X_p]

$\wedge$ $|X_1| = \ldots = |X_{i-1}| \geq |X_i| = \ldots = |X_p| \geq |X_1| - 1$).

It is FPT (for fixed p).

(2) *Partition into 2 regular graphs* :

$\exists X$ (Reg[X] $\wedge$ Reg[X^c])

Reg[X] means that the subgraph induced on X is regular;

X^c is the complement of X. It is XP.

(3) *Counting p-colorings* with particular properties, e.g., acyclic or equitable.

(4) Minimizing the use of a particular color: this gives a "distance to *p*-colorability" for a graph that *p+1*-colorable but not p-colorable.

In general, we can handle properties and functions of the forms

$$\exists \underline{X}.P(\underline{X}),\ \text{MSp}\ \underline{X}.P(\underline{X}),\ \text{Sp}\ \underline{X}.P(\underline{X}),\ \#\ \underline{X}.P(\underline{X})$$

where $P(\underline{X})$ is a Boolean combination of properties for which we have constructed fly-automata (Reg, NoCycle, Stable, etc…).

The system AUTOGRAPH (by I. Durand)

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 (combinations of descriptions)

product : for $P \wedge Q$, $P \vee Q$, $g(\alpha_1, \ldots, \alpha_p)$

A → A/X : for $P \to P[X]$, (P in induced subgraph on X) and

A → $A/(X \cap Y) \cup (Y \cap Z)^c$ for relativization to *set terms.*

image automaton: A → h(A) : in the transitions of A, each function symbol f is replaced by h(f) ; makes h(A)

nondeterministic : for $P(\underline{X}) \to \exists \underline{X}.P(\underline{X})$

Procedures to build automata that compute functions:

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

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

MSp$\underline{X}$.P($\underline{X}$) : the corresponding multiset.

SetVal$\underline{X}$.α($\underline{X}$)/P($\underline{X}$) : the set of values of α($\underline{X}$) for the tuples $\underline{X}$ that satisfy P($\underline{X}$).

For each case, a procedure transforms FA for P($\underline{X}$) and α($\underline{X}$) into FA that compute the associated functions. (These transformations do not depend on P($\underline{X}$) and α($\underline{X}$).)

Conclusion

In most cases, we get XP (or FPT ones) algorithms, that can be obtained independently.

These algorithms are based on fly-automata, that can be quickly constructed from logical descriptions → flexibility.

These constructions are implemented. Tests have been made for colorability and connectedness problems.

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