Computations of graph polynomials
by \textit{fly-automata}

Bruno Courcelle

(joint work \textit{in progress} with Irène Durand)

Bordeaux University, LaBRI (CNRS laboratory)
Overview

We give algorithms based on MSO (Monadic Second-Order) logic and automata that will help to compute MSO-definable polynomials for graphs of bounded tree-width or clique-width.

We use infinite automata, called fly-automata, that compute their transitions. Their inputs are finite algebraic terms denoting graphs.
The functions to compute are, typically, for $\varphi$ MSO:

\[
\text{#}(X,Y).\varphi(X,Y) := \text{number of pairs } (X,Y) \text{ that satisfy } \varphi(X,Y) \text{ in graph } G,
\]

\[
\text{Sp}(X,Y).\varphi(X,Y) := \text{set of pairs } (|X|, |Y|) \text{ for } (X,Y) \text{ satisfying } \varphi(X,Y) \text{ in graph } G,
\]

\[
\text{MSp}(X,Y).\varphi(X,Y) := \text{multiset of pairs } (|X|, |Y|) \text{ for } (X,Y) \text{ satisfying } \varphi(X,Y) \text{ in graph } G, \quad (1)
\]

\[
\text{Max}(X).\varphi(X) := \text{max cardinality of } X \text{ that satisfies } \varphi(X) \text{ in } G.
\]

\[
\text{Sat}(X,Y).\varphi(X,Y) := \text{set of pairs } (X,Y) \text{ that satisfy } \varphi(X,Y) \text{ in } G.
\]

(1) : Cf. counting generalized colorings (Kotek, Makowsky, Zilber).
FPT algorithms are known for the first 4 cases and FPT ones in the size of the result for the last one (Grohe et al.).

Contradicting a common statement, automata can be used for that: we give a theoretical framework and report about implementation.
Relevance to graph polynomials

All classical graph polynomials are, in some sense, MSO-definable (Makowsky).

Examples: Matching polynomial

\[ M(G,u) := \sum m(G,k) \cdot u^k \]

\( m(G,k) \) is the number of \( k \)-matchings in \( G \) (sets of \( k \) pairwise disjoint edges) = \( \#(X) \cdot \varphi_k(X) \), where \( \varphi_k(X) \) says that \( X \) is a set of \( k \) pairwise disjoint edges (FO-definable for each \( k \)).
Sokal's multivariate polynomial, subsumes Tutte's.

\[ Z(G,u,x_E) := \sum u^{k(A)} \cdot x_A \]

summation is over all sets of edges \( A = \{a_1, \ldots, ap\} \),
\( x_{ai} \) is an undeterminate indexed by edge \( ai \),
\( x_A := x_{a_1} \cdot \ldots \cdot x_{ap} \) (commutative product)
\( k(A) := \) number of connected components of \( G[A] \).

We have \( \{k(A)\} = \text{Sp}(X). \varphi(X,A) \) where
\( \varphi(X,A) \) says that \( X \) has 1 vertex in each con. comp. of \( G[A] \).

The chromatic polynomial of \( G \) is \( Z(G,u, x_e := -1) \)

For Tutte’s polynomial \( T(G,u,v), \) we have

\[ (u-1)^{k(G)} \cdot (v-1)^n \cdot T(G,u,v) = Z(G,(u-1)(v-1), x_e := v-1) \]
Tutte’s polynomial

\[ T(G,u,v) := \sum t_{i,j} \cdot u^i \cdot v^j \]

\( t_{i,j} \) is the number of spanning trees of \textit{internal activity} \( i \) and \textit{external activity} \( j \), relative to a linear order on edges (from each term defining \( G \), we have such an order).

An MSO formula \( \varphi(X,Y,Z) \), where \( X,Y \) are sets of edges, is such that, if

\[ \text{MSp}(X,Y,Z).\varphi(X,Y,Z) = \ldots + p.(k,i,j) + \ldots \]

then:

\[ T(G,u,v) = \ldots + p.u^i v^j + \ldots \]

\( \varphi(X,Y,Z) \) says that \( X \) is a spanning tree \( T \) and \( Y \), \textit{resp.} \( Z \) are the \textit{internally}, \textit{resp. externally} active edges of \( G \) \textit{wrt} \( T \). (This counting works because \( Y \) and \( Z \) are uniquely determined from \( X \).)
Multivariate interlace polynomial (B.C., 2008)

\[ C(G, u, v, x_V, y_V) := \sum x_A \cdot y_B \cdot u^{f(A,B)} \cdot v^{g(A,B)} \]

\( G \Lambda B := G \) where the loops at the vertices in B are toggled,

\( f(A,B) := \text{rk}(G \Lambda B[A \cup B]) \) and \( g(A,B) := |A \cup B| - f(A,B) \),

\( \text{rk}(H) := \text{rank over GF(2) of the adjacency matrix of graph } H \).

The rank of \( H \) is \( \text{Max}(X). \varphi(X) \) for an MSO formula \( \varphi(X) \) written with the even cardinality set predicate \( \text{Even}(Y) \). (As graphs are ordered, this predicate is MSO-definable).
Graphs are defined by algebraic terms and processed by automata on these terms.

Our graph parameter is *clique-width* (cwd(.)) 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 (twd(.)) for constructing automata, and it is 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.
We can handle edge quantifications via incidence graphs:

If \( G = (V_G, \text{edg}_G(.,..)) \) then \( \text{Inc}(G) := (V_G \cup E_G, \text{inc}_G(.,..)) \)

where: \( \text{inc}_G(u,e) : \iff u \) is an end of \( e \).

MSO formulas over \( \text{Inc}(G) \) can use quantifications on edge sets of \( G \) and express more properties.

Proposition (T.Bouvier): \( \text{twd}(G) \leq k \implies \text{cwd}(\text{Inc}(G)) \leq k+3. \)

Hence, no exponential jump.

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.
Using automata

Theorem [B.C.]: For every $k$, every MSO graph property $P$ can be checked by a finite automaton, which recognizes the terms that:

1. are written over the finite set $F_k$ of operations that generate the graphs of $\text{clique-width}$ at most $k$, and
2. define a graph satisfying $P$.

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

Our remedy: We use fly-automata (in French “automates programmés”), whose states and transitions are described and not tabulated. Only the transitions necessary for a particular 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 terms of arbitrary clique-width defining regular graphs (all vertices of same degree).

It can 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 $G = (V_G, \text{edg}_G(.,.))$, undirected

$G$ is 3-colorable:

$$
\exists X,Y \ ( X \cap Y = \emptyset \ \land \\
\forall u,v \ \{ \ \text{edg}(u,v) \Rightarrow \\
\quad [(u \in X \Rightarrow v \notin X) \land (u \in Y \Rightarrow v \notin Y) \land \\
\quad (u \notin X \cup Y \Rightarrow v \in X \cup Y) ] \\
\} )
$$
$G$ is not connected:

$$\exists Z \ ( \exists x \in Z \land \exists y \not\in Z \land (\forall u,v (u \in Z \land \text{edg}(u,v) \Rightarrow v \in Z))$$

**Transitive and reflexive closure**: $\text{TC}(R, x, y)$:

$$\forall Z \{ \text{“Z is R-closed”} \land x \in Z \Rightarrow y \in Z \}$$

where “Z is R-closed” is defined by:

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

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

$$\forall x,y (x \in Y \land y \in Y \Rightarrow \text{TC(“u \in Y \land v \in Y \land \text{edg}(u,v)”, x, y})$$

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

Definition 2: Clique-width

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

One binary operation: disjoint union : \( \oplus \)

Unary operations: (1) edge addition denoted by \( \text{Add}_{a,b} \)

\( \text{Add}_{a,b}(G) \) is \( G \) augmented with undirected edges between every \( a \)-vertex and every \( b \)-vertex. The number of added edges depends on the argument graph.

\[
G = \quad H = \quad H = \text{Add}_{a,b}(G) ; \text{only} \; 5 \; \text{new edges added}
\]
Directed edges can be defined similarly.

(2) Vertex relabellings:

\( \text{Relab}_a \rightarrow_b (G) \) is \( G \) with every \( a \)-vertex is made into a \( b \)-vertex

**Nullary operations for basic graphs** with a single vertex \( a \), labelled by \( a \).

**Definition:** A graph \( G \) has **clique-width** \( \leq k \) (denoted by \( \text{cwd}(G) \) )

\( \iff G = G(t), \text{ defined by a term } t \text{ using } \leq k \text{ labels.} \)

**Example:** Cliques have clique-width 2.

\( K_n \) is defined by \( t_n \) where \( t_{n+1} = \)

\( \text{Relab}_b \rightarrow_a (\text{Add}_{a,b} (t_n \oplus b)) \)
New definition 3: Fly-automaton (FA)

\[ A = \langle F, Q, \delta, \text{Out} \rangle \]

- \(F\): 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 are encoded by finite words, integers are in binary),
- \(\text{Out} : Q \rightarrow D\), computable (\(D\): effective domain, a recursive set of words),
- \(\delta\): computable (bottom-up) transition function.

Nondeterministic case: \(\delta\) is \textit{finitely multi-valued}. 
This automaton defines a **computable function** \( T(F) \rightarrow D \)

(or \( T(F) \rightarrow P(D) \) if it is not deterministic)

If \( D = \{ \text{True, False} \} \), 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 $A = \langle F, Q, \delta, \text{Acc} \rangle$ be finite, nondeterministic.

Then $\#A := \langle F, [ Q \rightarrow \mathbb{N}], \delta^#, \text{Out} \rangle$

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

$\delta^#$ is easy to define such that the state reached at position $u$ in the input term is the function $\sigma$ such that $\sigma(q)$ is the number of runs reaching $q$ at $u$.

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

$\#A$ is a fly-automaton obtained by a generic construction that extends to the case of infinite fly-automata.
The algorithmic MSO meta-theorem through *fly-automata*

\[ \varphi \text{ (MSO formula)} \]

Fly-automaton Constructor

\[ A(\varphi) \]

\[ t \]

\[ G \rightarrow \text{Graph Analyzer} \rightarrow t \rightarrow A(\varphi) \]

- Yes
- No

\( A(\varphi) \) is an *infinite fly-automaton* over the countable set \( F \) of all graph operations that define clique-width. The time taken by \( A(\varphi) \) depends on the number of labels that occur in \( t \), not only on the size of \( G \) or \( t \).
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).
Example of a direct construction: Connectedness.

The state at position $u$ in term $t$ is the set of types (sets of labels) of the connected components of the graph $G(t/u)$. For $k$ labels ($k = \text{bound on clique-width}$), the set of states has size $\leq 2^k$. Proved lower bound: $2^\frac{k}{2}$.

$\rightarrow$ Impossible to compile the automaton (to list its transitions).

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

Some transitions:

$Add_{a,c} : \quad q \quad \xrightarrow{} \quad \{ \{a,b,c,d\}, \{b,d,f\} \}$,

$Relab_{b} : \quad q \quad \xrightarrow{} \quad \{ \{b\}, \{b,c,d\}, \{b,d,f\} \}$

Transitions for $\oplus$: union of sets of types.

Note: Also state $(p,p)$ if $G(t/u)$ has $> 2$ 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 \((\texttt{cwd})\) graph operations, \(F_k\) : those using labels 1, \(\ldots\), \(k\).

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

\[(k + n)^c \rightarrow\text{it is a P-FA (a polynomial-time FA)},\]

\[f(k).n^c \rightarrow\text{it is an FPT-FA},\]

\[a.n^{g(k)} \rightarrow\text{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 $F$ (the operations that generate all graphs) that recognizes the terms $t \in T(F)$ such that $P(G(t))$ holds.

For each $k$, its restriction to the finite signature $F_k$ (the operations that generate graphs of $cwd \leq k$) is a finite automaton.

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

(2) It 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 min. 9 sec. (modified with diagonals on the squares of the first column).

For the similar 6 x 250 grid : < 6 min. ; for 6 x 360 : < 9 min.

4-acyclic-colorability of the Petersen graph (clique-width 5) in 1.5 min., from a term in T(F₆).
(3-colorable but not acyclically; red and green vertices induce a cycle).
Existential quantifications and nondeterminism

Graph $G(t)$

Term $t$ over $F$
Term $t \ast (V_1, V_2)$ over $F^{[2]}$ ($[2]$ because of 2 Booleans).

$V_1 = \{1,3,4\}, \ V_2 = \{2,3\}$
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 $G(t^* (X,Y)) \neq \varphi(X,Y)$.

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

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 $G(t) \neq \exists X,Y. \varphi(X,Y)$. 
By an induction on $\varphi$, we construct for each $\varphi(X_1,\ldots,X_n)$ a

FA $A(\varphi(X_1,\ldots,X_n))$ that recognizes:

$$L(\varphi(X_1,\ldots,X_n)) := \{ t \star (V_1,\ldots,V_n) \in T(F^{(n)}) / (G(t), V_1,\ldots,V_n) \models \varphi \}$$

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

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

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

where $\text{pr}$ is the *projection* that eliminates the last Boolean;

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

30
Determinized runs of $B$ defined by deterministic FAs $C$

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

$$\{ \text{state } q \text{ of } B / \text{some run reaches } q \text{ at position } u \}$$

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

$$\{ (q,m) / m = \text{the number of runs that reach } q \text{ at } u \}$$
equivalently, the corresponding multiset of states $q$, cf. $\exists X. P(X)$

For $SpX. P(X)$: the state of $C$ at position $u$ is

$$\{ (q,S) / S = \text{the set of tuples of cardinalities of } \text{the “components of } X \text{ below } u \text{” that yield } q \text{ at } u \}.$$}

For $MspX. P(X)$: $S$ is the corresponding multiset.
For MinCard X.P(X) : the state of C at position u is

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

SatX.P(X) := the set of all tuples X that satisfy P(X),
the state of 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 this cases:

We call the component $s$ in a 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 attributes 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…

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

Theorem (Flum and Grohe): One can compute $\text{Sat}_X.P(X)$ 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.
\[\oplus[p, q, r] \rightarrow p \]
\[\oplus[p, s] \rightarrow p \]
\[\oplus[q, v] \rightarrow q \]
\[\oplus[r, s] \rightarrow 2 \]
This 3-pass algorithm is applicable for all our computations of attributes.

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

1\textsuperscript{st} method: expressing that in MSO: possible but cumbersome.

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

3\textsuperscript{rd} method: “optimized” counting with reporting Failure if a useful intermediate result shows that more than 6 coloring will be found.

This is applicable to: \( \exists ! X. P(X) \) for every MSO property \( P \).
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 $\mathbf{X}$ satisfying $\varphi(\mathbf{X})$: 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 $E = (D, \text{reset}, \text{next})$ where $D$ is an effective (countable) set, $\text{reset}$ and $\text{next}$ are two programs guaranteed to terminate. $E$ defines a *finite list* $\text{List}(E)$ of elements of $D$. $\text{List}(E)$ may contain repetitions, $\text{next}$ produces one more element or reports “end of list”, $\text{reset}$ reinitializes the program $\text{next}$.

*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 $\exists X.\varphi(X)$ and that is obtained from the deterministic automaton $A$ checking $\varphi(X)$, by deleting the sequences of Booleans $w$ in the nullary symbols $(a,w)$ of the signature $F^{[p]}$ of $A$).

Alternatively, $E_a$ 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 $E : \text{copy}_u(E)$ indexed by $u$, a position of the given term.

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

Applying a unary function $h : D \rightarrow D'$

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” : $(d_1, d'_1), (d_1, d'_2), (d_2, d'_1),$

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

$\mathbf{E \times_{\text{Line}} E'}, \quad \mathbf{E \times_{\text{Col}} E'}, \quad \mathbf{E \times_{\text{Diag}} E'}$ realize these enumerations.
Given a term \( t \) and an automaton \( A \) that checks \( \varphi(X) \), one builds a (big) enumerator \( E_t \) by combining basic ones with Cartesian compositions, \( ho(.) \) and possibly \( nr(.) \).

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

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

Running \( E_t \) by calling its \textit{next} component iteratively produces the desired list (unless the system lacks of memory).
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: \( \land \), \( \lor \), negation, \( \exists X. \varphi(X) \).
Procedures to build automata that compute functions:

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

$Sp_X.\varphi(X)$ : the spectrum = the set of tuples of cardinalities of the components of the $X$ that satisfy $\varphi(X)$, 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 $6 \times M$ of clique-width 8.

<table>
<thead>
<tr>
<th>M</th>
<th>2-col. det</th>
<th>2-col. enum</th>
<th>3-col. det</th>
<th>3-col. enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.03 s</td>
<td>6 s</td>
<td>10 s</td>
<td>6 s</td>
</tr>
<tr>
<td>8</td>
<td>0.03 s</td>
<td>9 s</td>
<td>Fails</td>
<td>9 s</td>
</tr>
<tr>
<td>20</td>
<td>0.2 s</td>
<td>3 min</td>
<td>Fails</td>
<td>3 min</td>
</tr>
</tbody>
</table>

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.
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.