Recognizable sets of graphs: algebraic and logical aspects

Bruno Courcelle

LaBRI & Université de Bordeaux (ex -1 !)

References:


Summary

1. Recognizable & equational sets:
   an algebraic setting for language theory.

2. A graph algebra with infinitely many operations & clique-width.

3. Monadic second-order logic, recognizability & fixed-parameter tractable model-checking.

4. Constructing automata.
Two ways of considering graphs

A graph is a *logical structure*;
  graph properties can be expressed by logical formulas
  (FO = first-order, MSO = monadic second-order)

A graph is an *algebraic object*,
  an element of an algebra of graphs
  (similar to words as elements of monoids)
Graph algebras (finite graphs, up to isomorphism, as algebraic objects):

- **Equational sets** → context-free sets of graphs, without needing to define graph rewritings
- **Recognizable sets** → defined without automata
  (there is no “good” notion of graph automaton)
- **Algebraic understanding** of combinatorial notions:
  - tree-width, path-width, rank-width.
- **Clique-width:** a width measure defined algebraically.

The two graph algebras that fit to MSO logic yield:

- **Hyperedge-replacement** **gragras**, **tree-width**, FPT **MSO**\(_2\) model-checking.
Below: the algebra \( G \) that yields \textit{clique-width}.

The automata for model-checking MSO formulas on graphs of bounded clique-width are \textit{easier to build} than those for graphs of bounded tree-width.

And: all results about model-checking of MSO2 formulas (with edge set quantifications) for graphs of bounded \textit{tree-width} follow from the ones for graphs of bounded clique-width: we replace a graph \( G \) by its incidence graph \( \text{Inc}(G) \), we use: \[ \text{cwd}(\text{Inc}(G)) \leq \text{twd}(G) + 3 \] (T. Bouvier, LaBRI) and we note that MSO2 over \( G \) reduces to MSO over \( \text{Inc}(G) \).
1. Recognizable & equational sets:

an algebraic setting for Language Theory.

Equational sets (generalizing context-free languages)

Equation systems = Context-Free (Graph) Grammars

For words, the set of context-free rules

\[ X \to a \cdot X \cdot Y \; ; \; X \to b \; ; \; Y \to c \cdot Y \cdot Y \cdot X \; ; \; Y \to a \]

is equivalent to the system of two equations:

\[ X = a \cdot X \cdot Y \; \cup \; \{ b \} \]

\[ Y = c \cdot Y \cdot Y \cdot X \; \cup \; \{ a \} \]

where \( X \) is the language generated by \( X \) (idem for \( Y \) and \( Y \)).
In arbitrary algebras we consider equation systems like:

\[ X = f(k(X), Y) \cup \{ b \} \]

\[ Y = f(Y, f(g(Y), m(X))) \cup \{ a \} \]

where:

- \( f \) is a binary operation,
- \( g, k, m \) are unary operations,
- \( a, b \) denote basic objects.

An *equational set* is a component of the least solution of such an equation system. This notion is well-defined in any algebra.
The general algebraic setting

\[ F : \text{a finite or countably infinite set of operation symbols with fixed arities, called a signature.} \]

\[ M = < M, (f_M)_{f \in F} > : \text{an } F\text{-algebra.} \]

All signatures and algebras are finite or countable and effectively given (that is: encoded by finite words in a computable way).

**Examples**: *(words, trees and graphs will be finite).*

- Free monoid of words,
- free algebra of terms,
- algebra of terms with associative and/or commutative operations,
- monoid of traces (words with partial commutations).
Recognizable sets: finite signatures

\[ M = < M, (f_M)_f \in F > \] : an F-algebra where F is finite.

**Definition**: \( L \subseteq M \) is \((M-)recognizable\) if it is a union of equivalence classes of a finite congruence \( \approx \) on \( M \).

**Congruence** = equivalence relation such that:

\[ m \approx m' \text{ and } p \approx p' \Rightarrow f_M(m,p) \approx f_M(m',p'). \]

**Finite** means that \( \approx \) has finitely many classes \((M/\approx \text{ is finite})\).

Equivalently, \( L = h^{-1}(D) \) for a homomorphism \( h : M \to A \), where \( A \) is a finite F-algebra and \( D \subseteq A \).

\( \text{REC}(M) = \) the recognizable subsets of \( M \). This notion is relative to \( M \) (not only to the underlying set \( M \)). Well-defined in every algebra, without any automaton.
<table>
<thead>
<tr>
<th>Property</th>
<th>EQ (finite sig.)</th>
<th>REC (finite sig.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>closure for $\cup$</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>closure for $\cap, -$</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>$L_{eq} \cap K_{rec} \in EQ$</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>closure for hom, operations of $F$</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>closure for hom$^{-1}$</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>comparison</td>
<td></td>
<td>$\subseteq EQ$ (algebra generated by $F$)</td>
</tr>
<tr>
<td>membership (input given by a term)</td>
<td>undecidable</td>
<td>decidable</td>
</tr>
<tr>
<td>emptiness</td>
<td>decidable</td>
<td>decidable</td>
</tr>
</tbody>
</table>
For graphs: The two robust (in a precise logical and algebraic sense) graph algebras have *countably infinite* signatures.

No problem for *equational sets* because each equation system uses a finite signature, hence they are equational in a *finitely generated* subalgebra.

Recognizability requires an adapted definition. Below in abstract form; the application to graphs will come soon.
Recognizable sets: infinite signatures

\( \mathbf{M} = < M, (f_M)_{f \in F} > \) : an \( F \)-algebra where \( F \) is \textit{countably infinite}.

\( F, M, \) the functions \( f_M \) are \textit{effectively given} (encoded by integers in a computable way).

Each \( m \in M \) has a \textit{type} \( \pi(m) \) in a countable, effectively given set.

\( \pi \) is a homomorphism:

\[ \pi(f_M(m,m')) = \pi_f(\pi(m), \pi(m')) \] for some function \( \pi_f \) on types.

For defining the recognizability of a set \( L \), we require that \( \approx \) is \textit{type preserving}:

\[ m \approx m' \implies \pi(m) = \pi(m'), \]

\textit{locally finite}: it has finitely many classes of each type, and \( L \) is a union of classes (possibly of different types).
<table>
<thead>
<tr>
<th>Property</th>
<th>EQ</th>
<th>REC finite sign.</th>
<th>REC infinite sign.</th>
</tr>
</thead>
<tbody>
<tr>
<td>closure $\cup$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>closure $\cap$, -</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$L_{eq} \cap K_{rec} \in EQ$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>hom, ops of F</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>hom$^{-1}$</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>comparison</td>
<td>$\subseteq EQ$</td>
<td>incomparable with EQ</td>
<td></td>
</tr>
<tr>
<td>membership (given by a term)</td>
<td>undecidable</td>
<td>decidable</td>
<td>decidable</td>
</tr>
<tr>
<td>emptyness</td>
<td>decidable</td>
<td>decidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>
2. A graph algebra & clique-width.

Cographs: Undirected graphs generated by:

⊕ (disjoint union) and ⊗ (complete join) from

a : vertex without edges (up to isomorphism);

⊗ is defined by: \( G \otimes H = G \oplus H \) plus all
undirected edges between \( G \) and \( H \),

Cographs are defined by the equation: \( C = C \oplus C \cup C \otimes C \cup \{ a \} \)

Example:

\((a \otimes a \otimes a) \otimes ((a \otimes a) \oplus a)\)
The graph algebra \( G \) (that yields \textit{clique-width})

Graphs are \textit{simple}, directed or not.

We use labels: \( a, b, c, \ldots, d \).

Each vertex has one label; the labelling defines a partition of the vertex set.

A vertex labelled by \( a \) is an \textit{a-port}.

\textbf{One binary operation}: \textit{disjoint union}: \( \oplus \)

If \( G \) and \( H \) are not disjoint, we replace \( H \) by an isomorphic disjoint copy to define \( G \oplus H \). Hence \( G \oplus H \) is well-defined \textit{up to isomorphism}.

No such problem in a “decomposition approach”.
Unary operations: Edge-addition

Addition of undirected edges: $\text{Add}_{a,b}(G)$ is $G$ augmented with edges between every $a$-port and every $b$-port.

$H = \text{Add}_{a,b}(G)$; only 5 edges added

The number of added edges depends on the argument graph.
Addition of directed edges: $\text{Add}_{a,b}(G)$ is $G$ augmented with edges from every $a$-port to every $b$-port.

Vertex relabellings:

$\text{Relab}_{a \rightarrow b}(G) := G$ where each $a$-port is made into a $b$-port.

Nullary operations: basic graphs are those with a single vertex.

Every graph with $n$ vertices is the value of a term using $\leq n$ labels.

A graph $G$ has clique-width $\leq k$ $\iff$ it can be constructed from basic graphs with the operations $\oplus$, $\text{Add}_{a,b}$ (or $\text{Add}_{a,b}$) and $\text{Relab}_{a \rightarrow b}$ by using $\leq k$ labels.

Its clique-width $\text{cwd}(G)$ is the smallest such $k$. 
Example: Cliques have clique-width 2 (and unbounded tree-width)

\[ K_n \text{ is defined by } t_n \text{ where } t_{n+1} = \text{Relab}_b \rightarrow_a (\text{Add}_{a,b}(t_n \oplus b)) \]

Cliques are defined by the equation:

\[ K = \text{Relab}_b \rightarrow_a (\text{Add}_{a,b}(K \oplus b)) \cup a \]
Examples of graphs of bounded clique-width:

An undirected graph is a cograph $\iff$ it has clique-width at most 2.

*Distance hereditary graphs* have clique-width at most 3.

*Cactuses* (or *cactii*?) : biconnected components are cycles.

Examples of graphs of unbounded clique-width:

Planar graphs (even of degree 3),

Interval graphs.

**Fact**: Unlike tree-width, clique-width is sensible to edge directions:

Cliques have clique-width 2 but *tournaments* have *unbounded clique-width*. 
3. Monadic second-order (MSO) logic & recognizability.

Graphs as logical structures:

\[ G = ( V_G, \text{edg}_G(.,..) ) \]

\[ \text{edg}_G(u,v) \iff \text{there is an edge } u \rightarrow v \text{ (or } u - v) . \]

parallel edges are not distinguished.

Monadic second-order logic = First-order logic extended with (quantified) set variables denoting sets of vertices.

“A set of edges” is (here) a binary relation over \( V_G \).

Typical MSO properties: k-colorability, transitive closure, properties of paths, connectivity, planarity
Examples: $G = (V_G, \text{edg}_G(.,.))$, undirected

Syntax is clear; shorthands are used (example $X \cap Y = \emptyset$).

(1) **G is 3-colorable**: NP-complete property.

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

(2) **G is not connected**: 

$$
\exists Z \ (\exists x \in Z \land \exists y \notin Z \land (\forall u, v \ (u \in Z \land edg(u, v) \Rightarrow v \in Z))
$$
(3) **Transitive and reflexive closure** : \( \text{TC}(R ; x, y) : \)

\[
\forall X \{ \text{“}X \text{ is } R\text{-closed”} \land x \in X \Rightarrow y \in X \}
\]
where “\(X\) is \(R\)-closed” is defined by:

\[
\forall u,v (u \in X \land R(u,v) \Rightarrow v \in X)
\]

The relation \(R\) can be defined by a formula. Here:

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

\(Y\) is free in \(R\). This formula expresses that \(G[Y]\) is connected.

(4) **Minors** : \(G\) contains a fixed graph \(H\) as a minor with \(V_H = \{1, \ldots, p\}\) :

there exist disjoint sets of vertices \(X_1, \ldots, X_p\) in \(G\)

such that each \(G[X_i]\) is connected and,

whenever \(i - j\) in \(H\), there is an edge between \(X_i\) and \(X_j\).

(5) **Planarity** is expressible : no minor \(K_5\) or \(K_{3,3}\) (Kuratowski - Wagner).
(6) *Has a cycle* (for a graph without loops):

\[ \exists x, y, z \left[ \text{edg}(x, y) \land \text{edg}(y, z) \land \text{“there is a path from } x \text{ to } z \text{ avoiding } y\text{”} \right] \]

(7) *Is a tree*: connected without cycles.

Non-expressible properties

- G is isomorphic to \( K_{p,p} \) for some \( p \): checking equal cardinality of two sets “needs” quantification over binary relations in order to find a bijection.

- G has a nontrivial automorphism, or has all vertices of same degree.

- Hamiltonicity needs edge-set quantifications.

*Remark*: Adding an *equicardinality* set predicate would spoil everything.
Two problems for a class $C$ of finite graphs and a logic

*Decidability*: does a given sentence hold in some (or all) graphs of $C$?

*Model-checking* is decidable: what is its time or space complexity?

<table>
<thead>
<tr>
<th>Language, class</th>
<th>Decidability</th>
<th>Model-checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>FO, all graphs</td>
<td>undecidable</td>
<td>polynomial time</td>
</tr>
<tr>
<td>MSO, clique-width $\leq k$</td>
<td>decidable</td>
<td>cubic time</td>
</tr>
<tr>
<td>MSO, unboundedcwd.</td>
<td>undecidable</td>
<td>Conjecture: not FPT (*)</td>
</tr>
</tbody>
</table>

(*) A related fact is proved by S. Kreutzer (LICS 2010) for unbounded tree-width and MSO formulas with edge quantifications. (The exact statement is very technical.)
Fixed-Parameter Tractability (FPT) for model-checking

An algorithm is \textit{FPT} if it takes time $f(k).n^c$ for some fixed function $f$ and constant $c$. The size of the input is $n$. The value $k$ is a \textit{parameter} of the input:

degree, diameter, tree-width, clique-width, etc.

The “hard part” of the time complexity depends on some function $f$ (arbitrary in the definition; in practice, it must be “limited”; the algorithm is then usable for small values of $k$).

Example: 3-colorability is NP-complete, even for planar graphs of degree $\leq 4$ (Dailey, 1980). \textit{Degree} is \textit{not} a good parameter, but \textit{tree-width} and \textit{clique-width} are.
MSO logic & recognizability: main results

For a sentence $\varphi$, $\text{Mod}(\varphi)$ is the set of finite models of $\varphi$.

1. Recognizability theorem: For every MSO sentence $\varphi$, $\text{Mod}(\varphi)$ is recognizable in the graph algebra $\mathcal{G}$. The type $\pi(\mathcal{G})$ of $\mathcal{G}$ is the set of its vertex labels.

2. Weak recognizability theorem: For every MSO sentence $\varphi$, for every $k$, $\text{Mod}(\varphi) \cap \text{CWD}(\leq k)$ is recognizable in $\mathcal{G}[F_k]$, the subalgebra of $\mathcal{G}$ generated by the finite signature $F_k$ of operations using labels $1, \ldots, k$. 
3. The recognizability theorem is *not a consequence* of the weak one. (The set $K$ of *kytes*, the $n \times n$ grids with a “tail” of length $n$, is not recognizable but $K \cap \text{CWD}(\leq k)$ is finite for each $k$, hence, recognizable).

4. Every set of square grids is recognizable. As there are uncountably many such sets, there is *no characterization* of recognizable sets of graphs by logical formulas or automata.

5. For algorithmic purposes, the weak theorem is enough. Its “Büchi-style” proof is given below, in terms of finite, implementable automata.
6. Proofs of the full theorem:

- either a “Feferman-Vaught-Shelah-style” proof manipulating “theories” (the set of formulas of quantifier-height $\leq k$ valid in a graph) from which a congruence is defined (two graphs have same theory).
- either a “Büchi-style” proof constructing infinite, “fly-automata” (“automates programmés”) that are still implementable.
4. Construction and use of automata

\[
\begin{align*}
\phi & \quad \text{(MSO formula)} \\
\text{Automaton Constructor} & \\
\text{Graph Analyzer} & \quad \text{Yes} \\
G & \rightarrow \quad \text{No} \\
 & \quad \text{Error: cwd}(G) > k \\
\end{align*}
\]

Steps are done “once for all”, independently of G.

\[A(\varphi, k)\]: finite automaton on terms \(t\).
Construction of $A(\varphi, k)$ ("Büchi-style" proof).

$k = \text{the number of vertex labels} = \text{the bound on clique-width}$

$F \ (= F_k) = \text{the corresponding set of operations and constants: }$

- $a$, $\emptyset$, $\oplus$, $\text{Add}_{a,b}$, $\overrightarrow{\text{Add}_{a,b}}$, $\text{Relab } a \longrightarrow b$

$G(t) = \text{the graph defined by a term } t \text{ in } T(F)$.

Its vertices are (in bijection with) the occurrences of the nullary symbols in $t$ that are not $\emptyset$. 
Example

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

1) we replace in $F$ each $a$ by the nullary symbol $(a, (w_1,\ldots,w_n))$ where $w_i \in \{0,1\}$ : we get $F^{(n)}$ (only nullary symbols are modified);

2) a term $s$ in $T(F^{(n)})$ encodes a term $t$ in $T(F)$ and an assignment of sets $V_1,\ldots,V_n$ to the set variables $X_1,\ldots,X_n$:
   
   if $u$ is an occurrence of $(a, (w_1,\ldots,w_n))$, then
   
   $w_i = 1$ if and only if $u \in V_i$.

3) $s$ is denoted by $t^*(V_1,\ldots,V_n)$
Example (continued)

Term \( t \ast (V_1, V_2) \)

\[
V_1 = \{1, 3, 4\}, \quad V_2 = \{2, 3\}
\]
By an induction on $\phi$, we construct for each $\phi(X_1,\ldots,X_n)$ a finite (bottom-up) deterministic automaton $A(\phi(X_1,\ldots,X_n), k)$ that recognizes:

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

**Theorem**: For each sentence $\phi$, the automaton $A(\phi, k)$ accepts in time $f(\phi, k)$. $|t|$ the terms $t$ in $T(F)$ such that $G(t) \models = \phi$

It gives a *fixed-parameter linear* model-checking algorithm for input $t$, and a *fixed-parameter cubic* one if the graph has to be parsed.
The inductive construction of $A(\varphi, k)$

Atomic formulas: discussed below.

For $\land$: product of two automata (deterministic or not)

For $\lor$: union of two automata (or product of two complete automata; product preserves determinism)

For *negation*: exchange accepting / non-accepting states for a complete *deterministic* automaton
**Quantifications:** Formulas are written without ∀

\[
\begin{align*}
&L( \exists X_{n+1} . \varphi(X_1, ..., X_{n+1}) ) = pr( L( \varphi(X_1, ..., X_{n+1}) ) ) \\
&A( \exists X_{n+1} . \varphi(X_1, ..., X_{n+1}), k ) = pr( A( \varphi(X_1, ..., X_{n+1}), k ) )
\end{align*}
\]

where \( pr \) is the *projection* that eliminates the last Boolean; \( \rightarrow \) a *non-deterministic* automaton.

Tools using *inverse homomorphisms* of automata:

- from \( A( \varphi(X_1, X_2), k ) \), we get \( A( \varphi(X_4, X_3), k ) \),
- from \( A( \varphi(X_1, X_2), k ) \), we get \( A( \varphi( X_3, X_1 \cup (X_2 \setminus X_4) ), k ) \),
- from \( A( \varphi, k ) \), we get \( A( \varphi[X_1], k ) \).
Some tools for constructing automata

Substitutions and inverse images (*cylindrifications*).

1) If we know $A(\varphi(X_1, X_2), k)$, we get $A(\varphi(X_4, X_3), k)$ because:

$$L(\varphi(X_4, X_3)) = h^{-1}(L(\varphi(X_1, X_2)))$$

where $h$ maps $(a, (w_1, w_2, w_3, w_4))$ to $(a, (w_4, w_3))$. We take

$$A(\varphi(X_4, X_3), k) = h^{-1}(A(\varphi(X_1, X_2)), k)$$

This construction preserves determinism and the number of states.

2) From $A(\varphi(X_1, X_2), k)$, we get $A(\varphi(X_3, X_1 \cup (X_2 \setminus X_4)), k)$ by $h^{-1}$

with $h$ mapping $(a, (w_1, w_2, w_3, w_4))$ to $(a, (w_3, w_1 \lor (w_2 \land \neg w_4)))$
Relativization to subsets by inverse images.

If $\varphi$ is a closed formula expressing a graph property $P$, its relativization $\varphi[X_1]$ to $X_1$ expresses that the subgraph induced on $X_1$ satisfies $P$. To construct it, we replace recursively

$$\exists y. \theta \text{ by } \exists y. y \in X_1 \land \theta,$$

etc...

However, there is an easy transformation of automata:

Let $h$ map $(a,0)$ to $\emptyset$ and $(a,1)$ to $a$.

$$L(\varphi[X_1]) = h^{-1}(L(\varphi))$$

Hence:

$$A(\varphi[X_1], k) := h^{-1}(A(\varphi, k))$$
The inductive construction (continued):

For atomic formulas and basic graph properties $\varphi(X_1,\ldots,X_n)$, we build complete deterministic automata over $F^{(n)}$ for recognizing the set of terms: $t^*(V_1,\ldots,V_n)$ in $L(\varphi(X_1,\ldots,X_n))$.

**Intuition**: in all cases, the state reached at node $u$ represents a finite information $q(u)$ about the graph $G(t/u)$ and the restriction of $V_1,\ldots,V_n$ to the vertices below $u$ (vertices = leaves)

1) if $u = f(v,w)$, we want that $q(u)$ is defined from $q(v)$ and $q(w)$ by a fixed function: $\rightarrow$ the transition function;

2) whether $(G(t), V_1,\ldots,V_n)$ satisfies $\varphi(X_1,\ldots,X_n)$ must be checkable from $q(root)$ : $\rightarrow$ the accepting states.
Atomic and basic formulas:

\[ X_1 \subseteq X_2, \quad X_1 = \emptyset, \quad \text{Single}(X_1), \]

\[ \text{Card}_{p,q}(X_1) : \text{cardinality of } X_1 \text{ is } = p \mod q, \]

\[ \text{Card}_{<q}(X_1) : \text{cardinality of } X_1 \text{ is } < q. \]

\[ \rightarrow \text{Easy constructions of automata with few states:} \]
\[ \text{respectively } 2, 2, 3, q, q+1 \text{ states.} \]

Example: for \( X_1 \subseteq X_2 \), the term must have no constant \((a, 10)\).
**Atomic formula**: \( \text{edg}(X_1, X_2) \) for directed edges

\( \text{edg}(X_1, X_2) \) means: \( X_1 = \{ x \} \land X_2 = \{ y \} \land \text{edg}(x, y) \)

Vertex labels belong to 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 \( u \) of \( t \); its subterm \( t/u \) defines \( G(t/u) \subseteq G(t) \)).

0 : \( X_1 = \emptyset, X_2 = \emptyset \)

Ok *Accepting state*: \( X_1 = \{ v \}, X_2 = \{ w \}, \text{edg}(v, w) \text{ in } G(t/u) \)

\( a(1) \) : \( X_1 = \{ v \}, X_2 = \emptyset, v \text{ has label } a \text{ in } G(t/u) \)

\( a(2) \) : \( X_1 = \emptyset, X_2 = \{ w \}, w \text{ has label } a \text{ in } G(t/u) \)

\( ab \) : \( X_1 = \{ v \}, X_2 = \{ w \}, v \text{ has label } a, w \text{ has label } b \) (hence \( v \neq w \)) and \( \neg\text{edg}(v, w) \text{ in } G(t/u) \)

Error : all other cases
\textit{Transition rules}

For the constants based on $a$:

$(a,00) \rightarrow 0$ ; $(a,10) \rightarrow a(1)$ ; $(a,01) \rightarrow a(2)$ ; $(a,11) \rightarrow \text{Error}$

For the binary operation $\oplus$:

\[
\begin{array}{c}
\text{If } p = 0 \text{ then } r := q \\
\text{If } q = 0 \text{ then } r := p \\
\text{If } p = a(1), \ q = b(2) \text{ and } a \neq b \text{ then } r := ab \\
\text{If } p = b(2), \ q = a(1) \text{ and } a \neq b \text{ then } r := ab \\
\text{Otherwise } r := \text{Error}
\end{array}
\]
For unary operations $\overrightarrow{Add}_{a,b}$

$$\overrightarrow{Add}_{a,b} \quad \rightarrow \quad r$$

If $p = ab$ then $r := \text{Ok}$ else $r := p$

For unary operations $\overrightarrow{Relab}_{a} \rightarrow b$

If $p = a(i)$ where $i = 1$ or $2$ then $r := b(i)$

If $p = ac$ where $c \neq a$ and $c \neq b$ then $r := bc$

If $p = ca$ where $c \neq a$ and $c \neq b$ then $r := cb$

If $p = \text{Error}$ or $0$ or $\text{Ok}$ or $c(i)$ or $c(i)$ or $cd$ or $dc$ where $c \neq a$ then $r := p$
We get also the Recognizability Theorem:

\[ A(\varphi, k) \] is a subautomaton of \[ A(\varphi, k+1) \]. The deterministic automata \[ A(\varphi, k) \] can be merged into a single \textit{infinite} deterministic automaton \[ A(\varphi) \] over \( F \) (the countable signature of all graph operations).

The state \( q(t) \) reached by \( A(\varphi) \) on any \( t \in T(F) \) belongs to a finite set built from \( \pi(G(t)) \), the type of the graph \( G(t) \) defined by \( t \).

Example of such a set: \( \{\text{Ok, Error}\} \cup P(\pi(G(t)) \times \pi(G(t))) \). If \( t \) and \( t' \) define isomorphic graphs, then \( q(t) = q(t') \).

The (global) congruence proving recognizability can be taken:

\[ G \approx G' \iff q(t) = q(t') \] where \( G = G(t) \) and \( G' = G(t') \)
Practical difficulties and remedies.

Parsing: 1. Checking if a graph has clique-width \( \leq k \) is NP-complete (with \( k \) in the input; Fellows et al.)

2. The \textit{cubic approximate} parsing algorithm (by Oum et al.) based on \textit{rank-width} is difficult to implement.

3. Szeider and Heule reduce to SAT the computation of clique-width and get exact values (and the corresponding terms) for graphs with at most 30 vertices and clique-width at most 12.

4. For certain classes of graphs of bounded \textit{clique-width} defined by forbidden induced subgraphs, optimal clique-width terms can be constructed in polynomial time, by using, in many cases, \textit{modular decomposition}.

5. Heuristics remain to be found.
Sizes of automata:

1. The number of states of $A(\varphi, k)$ is bounded by an $h$-iterated exponential where $h$ is the number of quantifier alternations of $\varphi$.

2. There is no alternative construction giving a fixed bound on nestings of exponentiations (Meyer & Stockmeyer, Frick & Grohe).

3. The construction by induction on the structure of $\varphi$ may need intermediate automata of huge size, even if the unique minimal deterministic automaton equivalent to $A(\varphi, k)$ has a manageable number of states.
An issue: *Fly-automata* (to be presented by Irène Durand)

States and transitions are not listed in huge tables: they are *specified* (in uniform ways for all $k$) by “small” programs. These automata can be nondeterministic.

Example of a state for connectedness:

$$q = \{ \{a\}, \{a,b\}, \{b,c,d\}, \{b,d,f\} \},$$

$a,b,c,d,f$ are vertex labels; $q$ is the set of *types* of the connected components of the current graph. ($\text{type}(H) = \text{set of labels of its vertices}$)

Some transitions:

- $\text{Add}_{a,c}: q \rightarrow \{ \{a,b,c,d\}, \{b,d,f\} \},$
- $\text{Relab}_{a \rightarrow b}: q \rightarrow \{ \{b\}, \{b,c,d\}, \{b,d,f\} \}$

Transitions for $\bigoplus$: union of sets of types.
Using fly-automata works for formulas without quantifier alternation but that can use “new” atomic formulas for “basic” properties.

Examples: p-acyclic colorability

\[ \exists X_1,\ldots,X_p \ (\text{Partition}(X_1,\ldots,X_p) \land \text{NoEdge}(X_1) \land \ldots \land \text{NoEdge}(X_p) \land \ldots \land \text{NoCycle}(X_i \cup X_j) \land \ldots) \]

(all \( i < j \); set terms \( X_i \cup X_j \) avoid some quantifications).

Minor inclusion: \( H \) simple, loop-free. \( \text{Vertices}(H) = \{ v_1,\ldots,v_p \} \)

\[ \exists X_1,\ldots,X_p \ (\text{Disjoint}(X_1,\ldots,X_p) \land \text{Conn}(X_1) \land \ldots \land \text{Conn}(X_p) \land \ldots \land \text{Link}(X_i, X_j) \land \ldots) \]

Existence of “holes”: odd induced cycles (to check \textit{perfectness}; one checks “anti-holes” on the edge-complement of the given graph).
Appendix: Graph operations that characterize tree-width

Graphs have distinguished vertices called sources, (or terminals or boundary vertices) pointed to by source labels from a finite set: \{a, b, c, ..., d\}.

Binary operation(s): Parallel composition

\(G \parallel H\) is the disjoint union of \(G\) and \(H\) and sources with same label are fused.

(If \(G\) and \(H\) are not disjoint, we use a copy of \(H\) disjoint from \(G\)).
Unary operations:

Forget a source label

\(\text{Forget}_a(G)\) is \(G\) without \(a\)-source: the source is no longer distinguished (it is made "internal").

Source renaming:

\(\text{Ren}_a \leftrightarrow b(G)\) exchanges source labels \(a\) and \(b\)

(replaces \(a\) by \(b\) if \(b\) is not the label of any source)

Nullary operations denote basic graphs: 1-edge graphs, isolated vertices.

Terms over these operations define (or denote) graphs (with or without sources). They can have parallel edges.
**Example**: Trees

Constructed with two source labels, \( r \) (root) and \( n \) (new root).

**Fusion** of two trees at their roots:

![Trees fusion diagram]

**Extension** of a tree by parallel composition with a new edge, forgetting the old root, making the "new root" as current root:

\[
e = r \quad \quad \quad \quad \quad \quad \quad n
\]

\( \text{Ren}_n \leftrightarrow r \quad (\text{Forget}_r (G // e)) \)

Trees are defined by: \( T = T // T \cup \text{extension}(T) \cup r \)
Relation to tree-decompositions and tree-width

Graph G

Tree T

Tree-decomposition (T,f) of G

Dotted lines - - - - link copies of a same vertex.

Width = max. size of a box - 1.  Tree-width = min. width of a tree-dec.
**Proposition:** A graph has tree-width $\leq k \iff$ it can be constructed from edges by using the operations $\parallel$, $\text{Ren}_a \leftrightarrow b$ and $\text{Forget}_a$ with $\leq k+1$ labels $a, b, \ldots$.

**Proposition:** Bounded tree-width implies bounded clique-width $(\text{cwd}(G) \leq 2^{2\text{twd}(G)+1}$ for $G$ directed), but not *conversely*. 
From an algebraic expression to a tree-decomposition

Example: \( cd \xlongequal{\text{Ren}_a} c (\text{ab} \xlongequal{\text{Forget}_b} \text{ab} \xlongequal{\text{bc}}) \)  

\( \text{(ab denotes an edge from a to b)} \)

The tree-decomposition associated with this term.