Practical verification of MSO properties of graphs of bounded clique-width

Irène Durand (joint work with Bruno Courcelle)

LaBRI, Université de Bordeaux

20 Octobre 2010

Décompositions de graphes, théorie, algorithmes et logiques, 2010
Objectives

Verify properties of graphs of bounded clique-width

Properties

- connectedness,
- $k$-colorability,
- existence of cycles
- existence of paths
- bounds (cardinality, degree, ...)
- ...

How: using term automata

Note that we consider finite graphs only
Graphs as relational structures

For simplicity, we consider simple, loop-free undirected graphs. Extensions are easy.

Every graph $G$ can be identified with the relational structure $(\mathcal{V}_G, edg_G)$ where $\mathcal{V}_G$ is the set of vertices and $edg_G \subseteq \mathcal{V}_G \times \mathcal{V}_G$ the binary symmetric relation that defines edges.

\[\mathcal{V}_G = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}\]

\[edg_G = \{(v_1, v_2), (v_1, v_4), (v_1, v_5), (v_1, v_7), (v_2, v_3), (v_2, v_6), (v_3, v_4), (v_4, v_5), (v_5, v_8), (v_6, v_7), (v_7, v_8)\}\]
Expression of graph properties

First order logic (FO):

- quantification on single vertices $x, y \ldots$ only
- too weak; can only express "local" properties
- $k$-colorability ($k > 1$) cannot be expressed

Second order logic (SO):

- quantifications on relations of arbitrary arity
- SO can express most properties of interest in Graph Theory
- too complex (many problems are undecidable or do not have a polynomial solution).
Monadic second order logic (MSO)

- SO formulas that only use quantifications on unary relations (i.e., on sets).
- can express many useful graph properties like connectedness, $k$-colorability, planarity...

Example: $k$-colorability

$Stable(X)$: $\forall u, v (u \in X \land v \in X \Rightarrow \neg edg(u, v))$

$Partition(X_1, \ldots, X_m)$:
$\forall x (x \in X_1 \lor \ldots \lor x \in X_m) \land \forall i \neq j \forall x (x \in X_i \Rightarrow x \notin X_j)$

$k$-colorability:
$\exists X_1, \ldots, X_k Partition(X_1, \ldots, X_k)$
$\land Stable(X_1) \land \ldots \land Stable(X_k)$

Interesting algorithmic consequences
The fundamental theorem

Theorem

Courcelle (1990) for tree-width,
Courcelle, Makowski, Rotics (2001) for clique-width

Monadic second-order model checking is fixed-parameter tractable
for tree-width and clique-width.

- Tree-width and clique-width: graph complexity measures
  based on graph decompositions
- a decomposition produces a term representation of the graph
- the algorithm is given by a term automaton recognizing the
  terms denoting graphs satisfying the property
- How can we find this automaton?
Representation of graphs by terms

- depends on the chosen width (here \textit{clique-width})
- other widths: tree-width, path-width, boolean-width, ... 

Let $\mathcal{L}$ a finite set of labels \{a, b, c, ...\}. Graphs $G = (\mathcal{V}_G, \mathcal{E}_G)$ s.t.
  - each vertex $v \in \mathcal{V}_G$ has a \textit{label}, $\text{label}(v) \in \mathcal{L}$.

Operations:
- constant $a$ denotes a graph with a single vertex labeled by $a$,
- $\oplus$ (binary): union of disjoint graphs
- $\text{add}_{a,b}$ (unary): adds the missing edges between every vertex labeled $a$ and every vertex labeled $b$,
- $\text{ren}_{a,b}$ (unary): renames $a$ to $b$

Let $\mathcal{F}_\mathcal{L}$ be the set of these operations and constants.
Every term $t \in \mathcal{T}(\mathcal{F}_\mathcal{L})$ defines a graph $G_t$ whose vertices are the constants (leaves) of the term $t$. 
Definition
A graph has **clique-width** at most $k$ if it is defined by some $t \in \mathcal{T}(\mathcal{F}_\mathcal{L})$ with $|\mathcal{L}| \leq k$.

Note that different terms may define identical graphs.
Term automata (Bottom-up)

\[ \mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta) \] with \( \Delta \) set of transitions \( f(q_1, \ldots, q_n) \to q \)

Automaton 2-STABLE
Signature: a b ren_a_b:1 ren_b_a:1 add_a_b:1 oplus:2*
States: <a> <b> <ab> <error>
Final States: <a> <b> <ab>

Transitions
- a \to <a>
- b \to <b>
- add_a_b(<a>) \to <a>
- ren_a_b(<a>) \to <b>
- ren_a_b(<b>) \to <b>
- ren_a_b(<ab>) \to <b>
- ren_b_a(<a>) \to <a>
- ren_b_a(<b>) \to <a>
- ren_b_a(<ab>) \to <a>
- oplus*(<a>,<a>) \to <a>
- oplus*(<b>,<b>) \to <b>
- oplus*(<a>,<b>) \to <ab>
- oplus*(<b>,<ab>) \to <ab>
- oplus*(<a>,<ab>) \to <ab>
- oplus*(<b>,<ab>) \to <ab>
- add_a_b(<ab>) \to <error>
- ren_a_b(<error>) \to <error>
- add_a_b(<error>) \to <error>
- ren_b_a(<error>) \to <error>
- oplus*(<error>,q) \to <error> for all q
Run of an automaton on a term

The term is recognized when we obtain a final state at the root.

\[ G \]

\[ t_G = add_{a\_b}(\oplus(a, b)) \]

\[ add_{a\_b}(\oplus(a, b)) \rightarrow add_{a\_b}(\oplus(<a>, b)) \rightarrow add_{a\_b}(\oplus(<a>, <b>)) \rightarrow add_{a\_b}(<ab>) \rightarrow add_{a\_b}(<b>) \rightarrow <b> \]
Free set variables \( P(X_1, \ldots, X_m) \)

Each \( X_i \) corresponds to a subset of vertices
To express membership of vertices to the \( X_i \), the constants (representing the vertices of the graph) are associated with a bit-vector \( k_1 \ldots k_m \). \( k_i = 1 \) iff the vertex belongs to \( X_i \).

\( Stable(X_1) \) : the subgraph induced by \( X_1 \) is a stable
\( \mathcal{A}_{Stable}(X_1) \) can be obtained from \( \mathcal{A}_{Stable()} \)

New signature:
\( a^0 a^1 b^0 b^1 \text{ ren}_a_b:1 \text{ ren}_b_a:1 \text{ add}_a_b:1 \text{ oplus}:2* \)

New constant transitions:
\( a^0 \rightarrow \# \quad a^1 \rightarrow <a> \)
\( b^0 \rightarrow \# \quad b^1 \rightarrow <b> \)

New non constant transitions:
\( \text{ren}\_\_\_*(\#) \rightarrow \# \quad \text{add}\_\_\_*(\#) \rightarrow \# \quad \text{oplus}(\#,q) \rightarrow q \text{ for all } q \)

\( \text{add}_a_b(\text{oplus}(\text{oplus}(a^1,b^0),a^1)) \rightarrow + \)
\( \text{add}_a_b(\text{oplus}(\text{oplus}(<a>,\#),<a>)) \rightarrow \)
\( \text{add}_a_b(\text{oplus}(\#,<a>)) \rightarrow \text{add}_a_b(<a>) \rightarrow <a> \)
Example of the *Path*(\(X_1, X_2\)) property

Graph \(G\), \(X_1\) and \(X_2\) two subsets of vertices of \(G\)

Predicate \(Path(X_1, X_2)\), true when \(X_1 \subseteq X_2\), \(|X_1| = 2\) and some path in \(G[X_2]\) links the two vertices of \(X_1\).

\[
X_1 = \{v_3, v_8\} \\
X_2 = \{v_1, v_3, v_4, v_7, v_8\} \\
v_8 - v_7 - v_1 - v_4 - v_3
\]
The following term describes the previous graph with one of the set variables assignment:

\[
\text{add\_c\_d}(
\text{add\_b\_d}(
\oplus(d^{01},
\text{ren\_d\_b}(
\text{add\_a\_d}(
\oplus(d^{00},
\text{add\_c\_e}(
\oplus(\text{add\_a\_b}(\text{add\_b\_c}(\oplus(a^{11}, \oplus(b^{01}, c^{00}))))),
\text{add\_a\_b}(\text{add\_b\_e}(\oplus(a^{00}, \oplus(b^{01}, e^{11}))))))))))))
\]
The $\text{Path}(X_1, X_2)$ can be expressed by the following MSO formula:

\[
\forall x [x \in X_1 \Rightarrow x \in X_2] \land \exists x, y [x \in X_1 \land y \in X_1 \land x \neq y] \land \\
\forall z (z \in X_1 \Rightarrow x = z \lor y = z) \land \\
\forall X_3 [x \in X_3 \land \forall u, v (u \in X_3 \land u \in X_2 \land v \in X_2 \land \text{edg}(u, v) \Rightarrow v \in X_3) \\
\Rightarrow y \in X_3]]
\]

of quantifier-height 5. Uppercase variables correspond to sets of vertices, and lowercase variables correspond to individual vertices.
The problem

Input:
- an MSO formula $\phi = P(X_1, \ldots, X_m)$ expressing a graph property
- a graph $G$ represented by a term $t_G$ with an assignment to $X_1, \ldots, X_m$

Question:
- Does $G$ satisfy the graph property expressed by $\phi$

Example
$Path(X_1, X_2)$ and the previous graph (with an assignment of the sets variables).
The general solution

1. Transform the MSO formula $\phi$ into an automaton $A_\phi$
2. Run $A_\phi$ on the term $t_G$ representing the graph.

In order to process an MSO formula, we must standardize $\phi$.

1. translate it into an equivalent formula
   - without first-order variables (same quantifier-height)
   - with existential quantifiers only
   - with boolean operations only (and, or, negation)
   - and simple atomic properties like $X = \emptyset$, $Sgl(X)$ (denoting that $X$ is a singleton set), $X_i \subseteq X_j$ for which an automaton is easily computable.

2. standardize the names of set variables.
Standardization of the formula (Example)

\[
\begin{align*}
Path(X_1, X_2) &= X_1 \subseteq X_2 \wedge P_1(X_1, X_2) \\
P_1(X_1, X_2) &= \exists X_3, X_4, P_2(X_1, X_2, X_3, X_4) \\
P_2(X_1, X_2, X_3, X_4) &= Sgl(X_3) \wedge Sgl(X_4) \wedge X_3 \subseteq X_1 \wedge X_4 \subseteq X_1 \wedge X_3 \\
&\quad \neq X_4 \wedge |X_1| = 2 \wedge P_4(X_2, X_3, X_4) \\
P_4(X_2, X_3, X_4) &= \neg P_5(X_2, X_3, X_4) \\
P_5(X_2, X_3, X_4) &= \exists X'_1, P_6(X'_1, X_2, X_3, X_4) \\
P_6(X'_1, X_2, X_3, X_4) &= X_3 \subseteq X_5 \wedge \neg X_4 \subseteq X_5 \wedge P_7(X'_1, X_2) \\
P_7(X'_1, X_2) &= \neg P_8(X'_1, X_2) \\
P_8(X'_1, X_2) &= \exists X_3, X_4, P_9(X'_1, X_2, X_3, X_4) \\
P_9(X'_1, X_2, X_3, X_4) &= Sgl(X_3) \wedge Sgl(X_4) \wedge X_3 \subseteq X'_1 \wedge X_3 \subseteq X_2 \wedge \\
&\quad X_4 \subseteq X_2 \wedge Edg(X_3, X_4) \wedge \neg X_4 \subset X'_1
\end{align*}
\]

Note that this translation is here done by hand
Automata for atomic formulas

It is necessary to implement for once the ad-hoc constructions for the automata corresponding to atomic formulas

- $Edg(X_1, X_2)$,
- $Sgl(X)$,
- $X_1 \subseteq X_2$,
- $X_1 = X_2$,
- ... 

Some variable change or inverse homomorphisms may be applied in order to obtain all the desired versions. These transformations preserve determinism.

For instance, from an automaton for a property $P()$, we can easily obtain variants for $P(X)$, $P(\overline{X})$, $P(X_i)$, $P(X_i \cup X_j)$, $P(X_i \cap X_j)$, $P(\ldots, X_i, \ldots)$. 
The general algorithm for computing the automaton

If the formula is **atomic** (or if we already have an automaton for it) then return the corresponding automaton.

Otherwise:

- **disjunction** $\phi = \phi_1 \lor \phi_2$ : union of $A_{\phi_1}$ and $A_{\phi_2}$.
- **conjunction** $\phi = \phi_1 \land \phi_2$ : intersection of $A_{\phi_1}$ and $A_{\phi_2}$.
- **negation** $\phi = \neg \phi'$ : complementation of $A_{\phi'}$. ($A_{\phi'}$ must be determinized first).
- **existential formula** $\exists X_i, P(X_1, \ldots, X_m)$ : projection of $A_P(x_1, \ldots, x_m)$ on $(1, \ldots, i-1, i+1, m)$. creates nondeterminism
Autowrite

- Lisp software (currently 15000 lines)
- First designed to check call-by-need properties of term rewriting systems.
- Implements bottom-up term-automata and most of the well-known operations on such automata
  - union
  - intersection
  - determinization
  - minimization
  - complementation
  - projection
  - cylindrification
  - (inverse) homomorphism
  - ...
(setf *p9* (intersection-automata
  (list (setup-singleton-automaton *cwd* 4 3)
        (setup-singleton-automaton *cwd* 4 4)
        (setup-subset-automaton *cwd* 4 3 1)
        (setup-subset-automaton *cwd* 4 3 2)
        (setup-subset-automaton *cwd* 4 4 2)
        (complement-automaton
         (setup-subset-automaton *cwd* 4 4 1))
        (setup-edge-automaton *cwd* 4 3 4)))))

(setf *p8* (project-and-simplify-automaton *p9* '(0 1)))
(setf *p7* (complement-automaton *p8*))
(setf *p7p* (cylindrify-and-simplify-automaton *p7* '(2 3)))
(setf *p6* (intersection-automata
  (list *p7p*
        (setup-subset-automaton *cwd* 4 3 1)
        (complement-automaton
         (setup-subset-automaton *cwd* 4 4 1)))))
(setf *p5* (vprojection *p6* '(1 2 3))))
(setf *p5* ;; blows up for cwd=3
       (ndeterminize-automaton *p5*))
(setf *p5* (nsimplify-automaton *p5*))
(setf *p4* (complement-automaton *p5*))
(setf *p4p* (cylindrify-and-simplify-automaton *p4* 0))
(setf *p3* (intersection-automata
           (list *p4p*
                 (setup-subset-automaton *cwd* 4 3 1)
                 (setup-subset-automaton *cwd* 4 4 1)
                 (complement-automaton
                  (setup-equality-automaton *cwd* 4 3 4))
                 (setup-cardinality-automaton *cwd* 4 1 2))))
(setf *p2* (intersection-automata
           (list *p3*
                 (setup-singleton-automaton *cwd* 4 3)
                 (setup-singleton-automaton *cwd* 4 4))))
(setf *p1* (project-and-simplify-automaton *p2* '(0 1)))
(setf *p* (intersection-automata
           (list *p1* (setup-subset-automaton *cwd* 2 1 2))))
Results for the Path property

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A/\min(A)$</td>
<td>25 / 12</td>
<td>out</td>
</tr>
</tbody>
</table>

- Runs out of memory for $cwd = 3$, although we know that the minimal automaton has 124 states which is still reasonable.
- The problem comes from intermediate steps.
- The non-deterministic version of $A_{P_5}(x_2,x_3,x_4)$ has 308 states.
- Its complementation triggers its determinization which causes the blow up.
Second method: direct construction of the automaton

**Observation**: intermediate steps induce an exponential blow up although the final automaton is not so big.

**Idea**: give a direct construction of the automaton.

This method is **not** general.

For each property, one must give a description of the automaton

- description of the states,
- description of the transitions rules

Such a description exists for the **path property**

<table>
<thead>
<tr>
<th>$cwd$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{A}/\text{min}(\mathcal{A})$</td>
<td>25 / 12</td>
<td>213 / 124</td>
<td>4792 / 2015</td>
<td>out</td>
</tr>
</tbody>
</table>

Number of states of the unique minimal automaton:

$2^{cwd^2/2} < |Q| < 2^{cwd^2+2}$

For $cwd = 5$:

$33554432 < |Q|$

**Comment**: the automata are simply too **big**!
Fly automata

**Principle**: the transitions are represented by a function (in our case a Lisp function); the complete sets of transitions, states and final states are never computed in extenso.

**fly automaton** \( \mathcal{A} = (\mathcal{F}, \text{final}, \delta) \): abstraction of the usual automaton (with stored transitions)

```lisp
(defun fly-path-automaton (cwd)
  (make-fly-automaton
   (setup-signature cwd 2)
   (lambda (root states) ;; f(q1 ... qn) -> q
     (path-transitions-fun root states)))
   (lambda (state)
     (path-final-p state))))
```
Fly automata

- runs on all our data
- no limitation on the clique-width to create the automaton
- limitations come when running the automaton on very deep terms (stack exhaustion)

The implementation of operations on fly automata uses intensively the functional programming paradigm.

```lisp
(defmethod complement-automaton ((f fly-automaton))
  (make-fly-automaton
   (signature f)
   (get-transitions f)
   (complement-finalstate-fun f)))
```
Fly-automata versus Table-automata

**Table-automata**

- compiled version of fly-automata
- faster for recognizing a term
- use space for storing the transitions table
- the space depends on the clique-width

**Fly-automata**

- use constant space
- slower for term recognition because of the calls to the transition function
- the time depends on the clique-width

**Use**

- a table-automaton when the transitions table can be computed
- a fly-automaton otherwise
Experimental results

Connectedness on graphs $P_N$ ($cwd = 3$)
Some properties

Direct constructions of the automata for the following properties.

Polynomial

- Stable()
- Partition($X_1, \ldots, X_m$)
- $k$-Cardinality()

Non polynomial

- $k$-Coloring($C_1, \ldots, C_k$) compilable up to $cwd = 4$ (for $k = 3$)
- Connectedness() compilable up to $cwd = 3$
- Clique() compilable up to $cwd = 4$
- Path($X_1, X_2$) compilable up to $cwd = 4$
- Forest() (no cycle) not compilable
Some more properties

With the previous properties, using homomorphisms and boolean operations, we obtain automata for

- $k$-Colorability() compilable up to $k = 3$ ($cwd = 2$), $k = 2$ ($cwd = 3$)
- $k$-Acyclic-Colorability() not compilable (uses Forest)
- $k$-Chord-Free-Cycle()
- $k$-Max-Degre()
- Vertex-Cover($X_1$) $2^{cwd}$ states
- $k$-Vertex-Cover()
Experimental results

3-colorability on square-grids $N \times N$ (clique-width $N + 1$)
Experimental results

3-colorability on rectangular grids $6 \times N$ (clique-width 8)
Results and future work

<table>
<thead>
<tr>
<th>Property</th>
<th>graph</th>
<th>cwd</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-ac-colorability</td>
<td>petersen</td>
<td>7</td>
<td>17mn</td>
</tr>
<tr>
<td>3-colorability</td>
<td>grid 6x33</td>
<td>8</td>
<td>85mn</td>
</tr>
</tbody>
</table>

**Size of the graphs** Limit around 1,000,000 vertices
⇒ terms of size 4,000,000
need to increase stack size because the run of an automaton on a term is recursive

- more graph properties
- tests on real graphs and random graphs
- graph decomposition using few labels (**parsing problem**)
- the concept of **fly-automata** is general and could be applied to other domains