



Tractable constructions of finite automata from monadic second-order formulas

Bruno Courcelle, Irène Durand

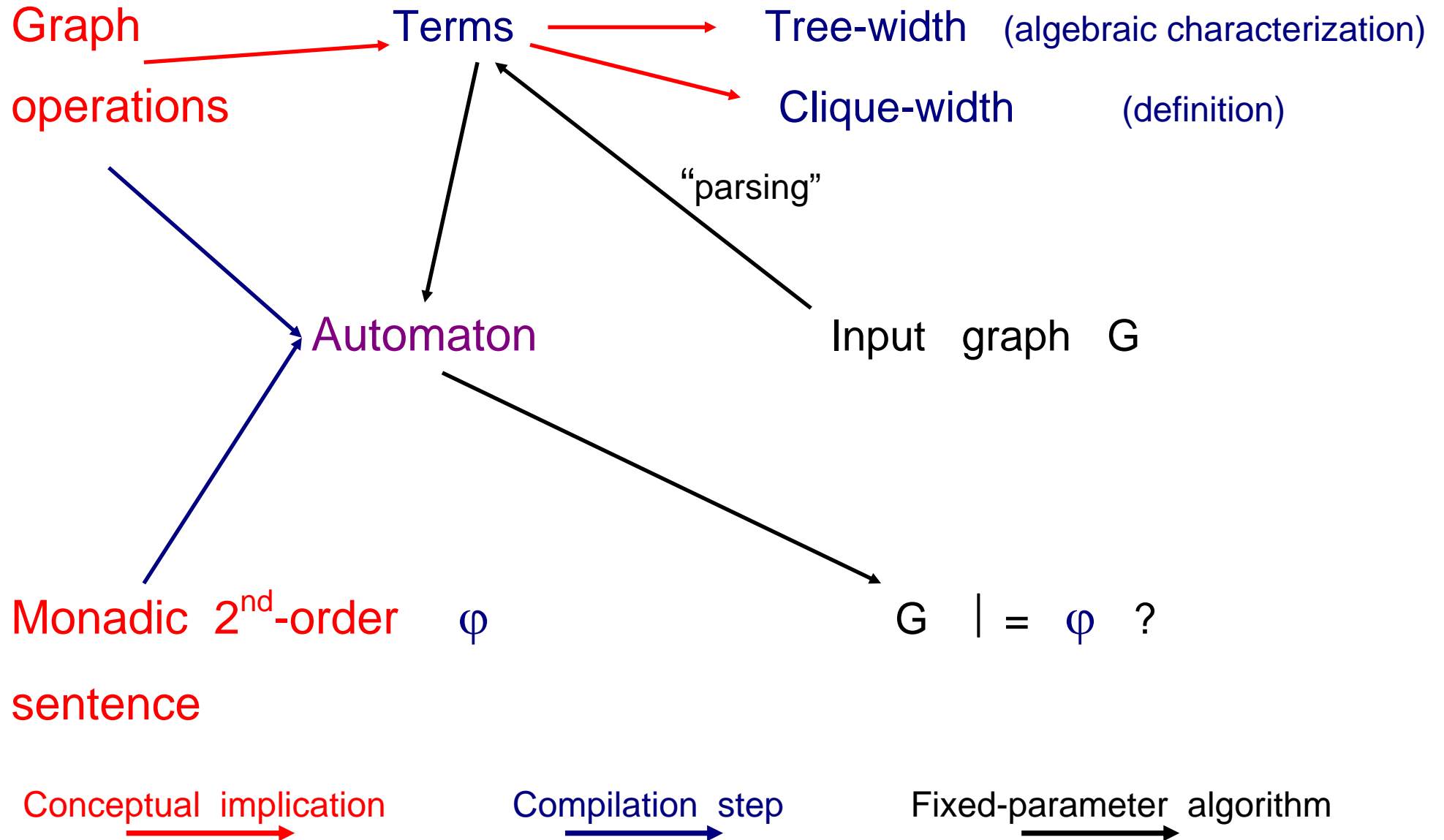
Université Bordeaux 1, LaBRI

References : B. Courcelle : Graph structure and monadic second-order logic,
book to be published by *Cambridge University Press*, readable on :

<http://www.labri.fr/perso/courcell/ActSci.html>

I. Durand : AUTOWRITE, a tool for term rewrite systems and tree
ENTCS 124 (2005) 29-49

An overview chart



Two graph algebras : “HR” and “VR”

Case 1 : “HR”, parameter **tree-width** \rightarrow Model-checking for MSO sentences *with* edge set quantifications.

Case 2 : “VR”, parameter **clique-width** \rightarrow Model-checking for MSO sentences without edge set quantifications.

Automata for Case 2 are easier to build.

Case 1 reduces to Case 2 :

Graph G of tree-width $k \geq 2 \rightarrow$ Incidence graph of G of tree-width k ,
hence of clique-width about 2^k .

But : Graph of **path-width** $k \geq 2 \rightarrow$ Incidence graph of path-width k ,
hence of **linear clique-width** $k+2$: no exponential !

Two difficulties :

Parsing, but in some concrete cases, graphs arise with their “natural” decompositions.

Construction of automata : fails in many cases by lack of memory.

What we propose :

- To use **predefined** deterministic automata for **basic** useful graph properties like : Path(X,Y), Stable(X), Clique(X), NoCycle(X), *etc.*
- To consider **existential quantifications** over **Boolean** combinations of basic formulas with **substitutions of set terms**
- **Not to determinize** automata
- To **interpret and not to compile** “combinatorially defined” automata
- To use **path-** and **linear clique-width** decompositions.

1. The graph algebra **VR**

Origin: Vertex Replacement *context-free* graph grammars

Associated complexity measure: **clique-width**.

Graphs are defined in terms of **very simple graph operations**.

Graphs are simple, loop-free, undirected (extension to directed case easy).

Vertex labels : a, b, c, \dots, d . Each vertex has one and only one label.

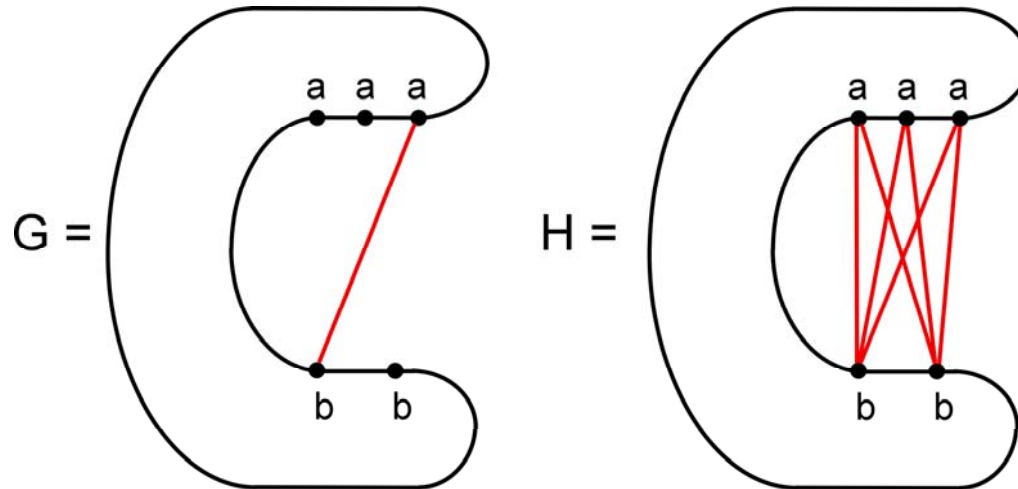
One binary operation: **disjoint union** : \oplus

Well-defined up to isomorphism : one takes disjoint copies ;

$G \oplus G$ is **not** equal to G

Unary operations: Edge addition denoted by $Add-edg_{a,b}$

$Add-edg_{a,b}(G)$ is G augmented with undirected edges between every a -labelled vertex and every b -labelled vertex



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

The number of added edges depends on the argument graph.

The directed version of $Add-edg_{a,b}$ adds directed edges from every a -labelled vertex to every b -labelled vertex

Vertex relabellings :

$Relab_{a \rightarrow b}(G)$ is G with every label a changed into b

Variant : $Relab_h(G)$ is G with every label a changed into $h(a)$ for some function $h : C \rightarrow C$; C is the *finite* set of labels.

Basic graphs : a one vertex labelled by a , for each a in C .

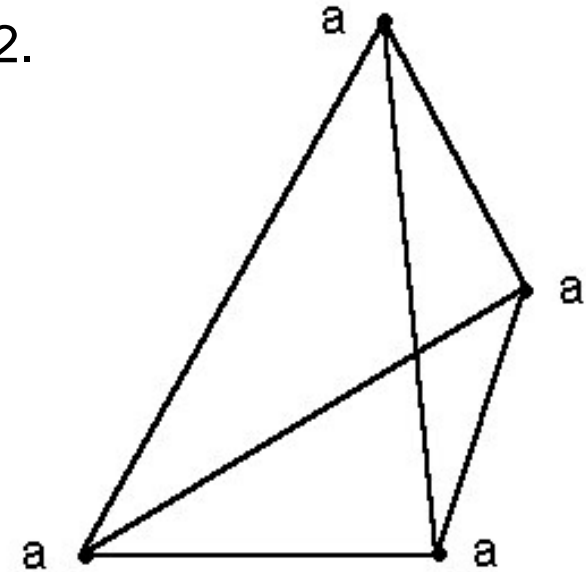
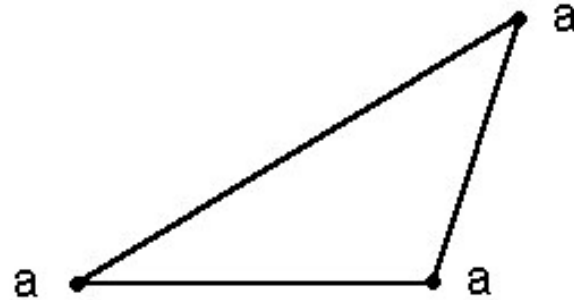
Definition: A graph G has *clique-width* $\leq k$

\Leftrightarrow it can be constructed from basic graphs with the operations

\oplus , $Add-edg_{a,b}$ and $Relab_{a \rightarrow b}$ with labels a, b in set C of k labels

Its (exact) *clique-width* $cwd(G)$ is the smallest such k .

Example : Cliques (*a*-labelled) have clique-width 2.



K_n is defined by t_n where $t_1 = \mathbf{a}$

$$t_{n+1} = \text{Relabb} \rightarrow a (\text{Add-edga,b} (t_n \oplus \mathbf{b}))$$

Example : Cographs (*a*-labelled) are generated by \oplus and \otimes defined by:

$$G \otimes H = \text{Relabb} \rightarrow a (\text{Add-edga,b} (G \oplus \text{Relaba} \rightarrow b (H)))$$

= $G \oplus H$ with “all edges” between G and H .

2. Monadic Second-Order (MSO) Logic

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

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

Examples of formulas for $G = (V_G, \text{edg}_G(.,.))$, undirected

p-colorability (NP-complete property)

$$\exists X_1, \dots, X_p \left(\text{Partition}(X_1, \dots, X_p) \wedge \text{Stable}(X_1) \wedge \dots \wedge \text{Stable}(X_p) \right)$$

p-acyclic colorability

$$\exists X_1, \dots, X_p \left(\text{Partition}(X_1, \dots, X_p) \wedge \text{Stable}(X_1) \wedge \dots \wedge \text{Stable}(X_p) \wedge \dots \right. \\ \left. \dots \wedge \text{NoCycle}(X_i \cup X_j) \wedge \dots \right)$$

Non connectivity, whence connectivity, more generally, transitive closure :

$$\exists X (\exists x \in X \wedge \exists y \notin X \wedge \forall u,v (u \in X \wedge \text{edg}(u,v) \Rightarrow v \in X))$$

Contains H as a minor :

H simple, loop-free. *Vertices(H) = {v₁, ..., v_p }*

$$\exists X_1, \dots, X_p (\text{Disjoint}(X_1, \dots, X_p) \wedge \text{Connected}(X_1) \wedge \dots \wedge \text{Connected}(X_p) \wedge \dots \\ \dots \wedge \text{Link}(X_i, X_j) \wedge \dots)$$

where $\text{Link}(X_i, X_j)$ means : there is an edge between some vertex of X_i
and some vertex of X_j .

One puts $\text{Link}(X_i, X_j)$ in the sentence for each edge $v_i \text{ ---- } v_j$ of H .

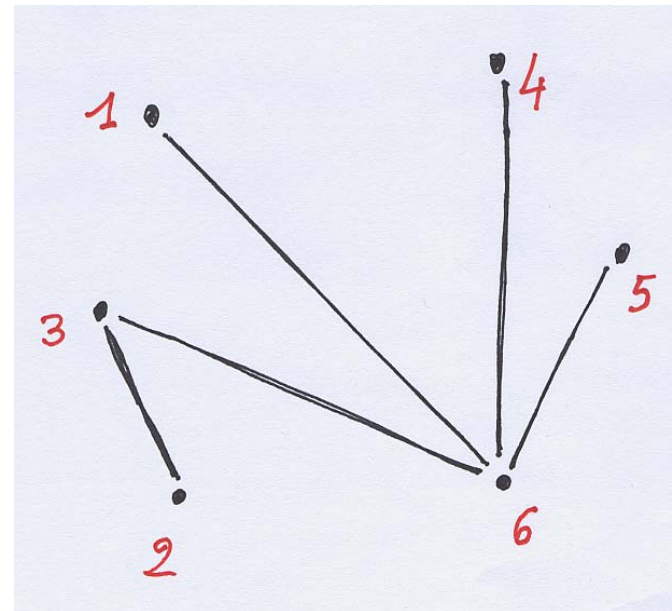
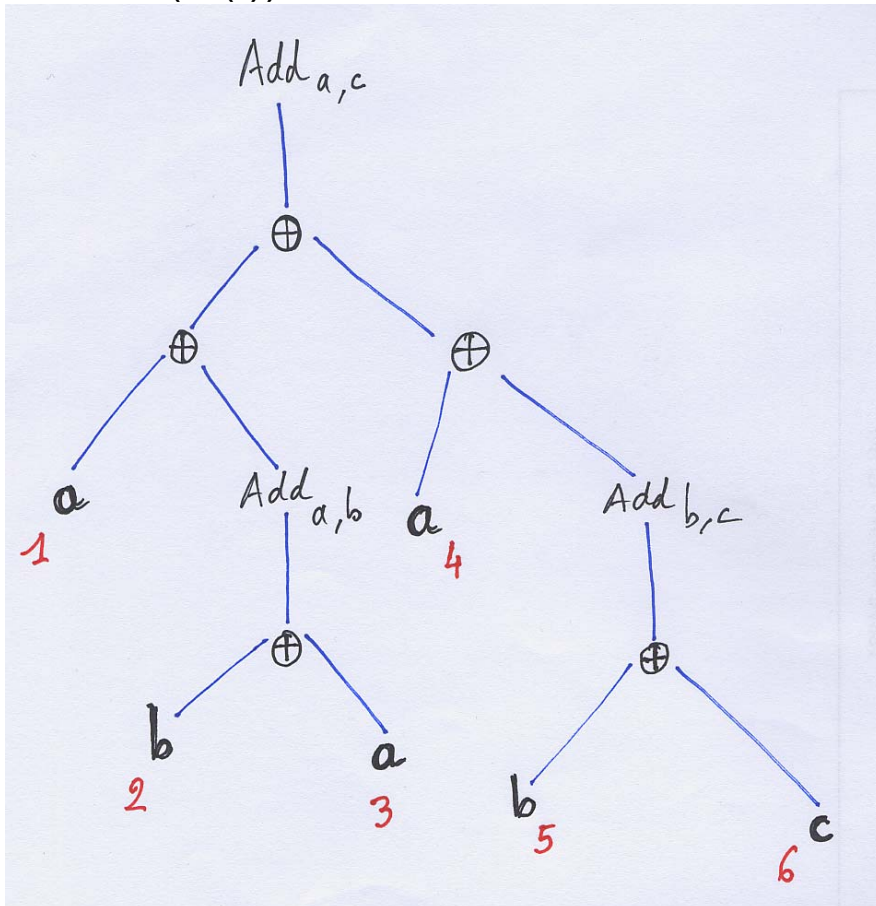
3. Büchi-style construction of automata for VR terms.

We fix k the number of vertex labels (hence the bound on clique-width).

F = the corresponding set : a , \oplus , $Add\text{-}edg_{a,b}$, $Relab\ a \longrightarrow b$

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

$Vertices(G(t))$ = the set of occurrences of constant symbols in t



Formulas are without first-order variables and \forall

Construction: For each sentence φ an automaton $A(\varphi)$ that defines the set of terms in $\mathbf{T}(F)$ such that $G(t) \models \varphi$

By induction on the structure of the sentence φ

For $\exists X. \varphi(X)$, we need $A(\varphi(X))$.

More generally, we need $A(\varphi(X_1, \dots, X_n))$.

A term t in $\mathbf{T}(F)$ defines a graph $G(t)$ with vertex set
 $=$ the set of occurrences of constants.

For representing assignments

$$v: \{ X_1, \dots, X_n \} \rightarrow P(\text{Vertices}(G(t)))$$

we replace in F each constant \mathbf{a} by the constants
 $(\mathbf{a}, (w_1, \dots, w_n))$ with $w_i \in \{0, 1\}$: we get $F^{(n)}$.

A term s in $\mathbf{T}(F^{(n)})$ encodes a term t in $\mathbf{T}(F)$ and an
assignment $v: \{ X_1, \dots, X_n \} \rightarrow P(\text{Vertices}(G(t)))$:

if u is an occurrence of $(\mathbf{a}, (w_1, \dots, w_n))$, then $w_i = 1$ iff $u \in X_i$.

Such a term s is denoted by t^*v .

A term $t*v$ in $\mathbf{T}(F^{(n)})$ defines the graph $G(t)$ and some assignment $v : \{X_1, \dots, X_n\} \rightarrow P(\text{Vertices}(G(t)))$.

From F and φ we will construct a finite deterministic automaton $A(\varphi(X_1, \dots, X_n))$ that recognizes :

$$L(\varphi(X_1, \dots, X_n)) := \{ t*v \in \mathbf{T}(F^{(n)}) \mid (G(t), v) \models \varphi \}$$

Main inductive steps

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

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

where pr is the “*projection*” that eliminates the last Boolean.

One obtains a *nondeterministic* automaton.

For \wedge and \vee : product of two complete automata (deterministic or not).

For *negation* : exchange accepting/non-accepting states for a *deterministic* automaton.

The case of atomic formulas is discussed below.

The number of states is an *h-iterated exponential*, where *h* = maximum nesting of negations.

This is not avoidable (Weyer, Frick and Grohe).

Substitutions and inverse images (“cylindrifications”).

If we know $A(\varphi(X_1, X_2))$, we can get easily $A(\varphi(X_4, X_3))$:

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

h maps $(\mathbf{a}, (w_1, w_2, w_3, w_4))$ to $(\mathbf{a}, (w_4, w_3))$

We take

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

This preserves determinism and number of states.

From $A(\varphi(X_1, X_2))$, we can get $A(\varphi(X_3, X_1 \cup (X_2 \setminus X_4)))$ by h^{-1} with h mapping $(\mathbf{a}, (w_1, w_2, w_3, w_4))$ to $(\mathbf{a}, (w_3, w_1 \vee (w_2 \wedge \neg w_4)))$.

Set term

Basic cases : Atomic formula : $\text{edg}(X_1, X_2)$ for directed edges

The automaton $A(\text{edg}(X_1, X_2))$ with k^2+k+3 states.

Vertex labels are in C with k elements.

$\text{edg}(X_1, X_2)$ means : $\text{Single}(X_1) \wedge \text{Single}(X_2) \wedge \text{Link}(X_1, X_2)$

States : 0, Ok, $a(1)$, $a(2)$, ab , Error, for a, b in C , $a \neq b$

Meanings of states (at node u in t ; its subterm t/u defines $G(t/u) \subseteq G(t)$).

0 : $X_1 = \emptyset$, $X_2 = \emptyset$

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

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

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

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

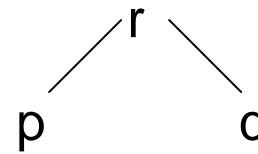
Error : all other cases

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 Error

For the binary operation \oplus :



If $p = 0$ then $r := q$

If $q = 0$ then $r := p$

If $p = a(1)$, $q = b(2)$ and $a \neq b$ then $r := ab$

If $p = b(2)$, $q = a(1)$ and $a \neq b$ then $r := ab$

Otherwise $r := \text{Error}$

For unary operations *Add-edge*_{a,b}

r
|
p

If $p = ab$ then $r := Ok$ else $r := p$

For unary operations *Relab*_{a→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 $p = Error$ or 0 or Ok or $c(i)$ or cd or dc where $c \neq a$

then $r := p$

Another construction using Backwards Translation:

The mapping : t in $\mathbf{T}(F) \longmapsto G(t)$ is an MSO transduction.

The set $L(\varphi)$ of terms t in $\mathbf{T}(F)$ such that $G(t) \models \varphi$ is defined by an MSO formula $\varphi^\#$ obtained by *Backwards Translation*.

By the **Recognizability Theorem** (Doner *et al.*) for terms, $L(\varphi)$ is definable by a finite automaton.

Short proof, but $\varphi^\#$ has **larger quantifier-height than φ** .

Hence bad in view of concrete implementations.

Implementation: The automaton constructed from φ and k frequently too large to be compiled. Problems with size of memory for intermediate automata, even if the *unique minimal deterministic* automaton has manageable number of states.

D. Soguet *et al.*, using MONA, and I. Durand with AUTOWRITE (figures in **blue**) have constructed automata for the following cases :

	<i>Clique-width 2</i>	<i>Clique-width 3</i>	<i>Clique-width 4</i>
MaxDegree ≤ 1	24 states	123 states	621 states
MaxDegree ≤ 3	91 states	Space-Out	
Degree $\leq 4(x)$	48 states	233 states	
Path(X,Y)	12 states	128 states	2197 states
Connected	11 states	Space-Out	
IsConnComp(X)	48 states	Space-Out	
Has ≤ 4 -VertCover	111 states	1037 states	
HasClique ≥ 4	21 states	153 states	
2-colorable	8 states	56 states	

What to do against this difficulty ?

(1) To use **predefined** deterministic automata for **basic** useful graph properties like : Path(X,Y), Stable(X), Clique(X), NoCycle(X), etc.

We define automata *directly* from the properties, without using the logical descriptions.

(2) To consider “only” **existential quantifications** over **Boolean** combinations of basic formulas with **substitutions of set terms**.

Typical examples : ***p*-acyclic colorability**

$$\exists X_1, \dots, X_p \left(\text{Partition}(X_1, \dots, X_p) \wedge \text{Stable}(X_1) \wedge \dots \wedge \text{Stable}(X_p) \wedge \dots \right. \\ \left. \dots \wedge \text{NoCycle}(X_i \cup X_j) \wedge \dots \right)$$

MSO constrained domination : $\exists X. (\text{Dom}(\text{AllVertices} \setminus X, X) \wedge \varphi(X))$

where $\text{Dom}(X_1, X_2)$ means : Every vertex in X_1 is linked to some vertex in X_2 .

(3) Not to determinize automata

Proposition: If A is a nondeterministic automaton with N states over a set of at most binary function symbols, with **nondeterminism degree d** , then, the membership in $L(A)$ of a term of size m can be done in time $O(m \cdot d \cdot N^2)$.

Corollary : Let P be a set of basic MSO properties with deterministic automata of at most $N(k)$ states for each $k =$ bound on clique-width.

For a sentence $\psi : \exists X_1, \dots, X_p(\varphi(X_1, \dots, X_p))$, where φ is a Boolean combination of q properties $P(t_1, \dots, t_s)$ from P and the t_i 's are set terms over X_1, \dots, X_p , then, for every term t in $\mathbf{T}(F)$:

$G(t) \models \psi$ can be checked in time $O(|t| \cdot (2^p + N(k)^{2q}))$.

(4) Use “linear clique-width” terms (cf. path-width w.r.t. tree-width):

Automata have less transitions, so we get :

$G(t) \models \psi$ can be checked in time $O(|t| \cdot (2^p + N(k)^q))$

Instead of :

$G(t) \models \psi$ can be checked in time $O(|t| \cdot (2^p + N(k)^{2q}))$.

Of course, we need more labels, but this may be no problem with the next idea.

(5) The last “secret weapon” to fight the “Dragon” :

To *interpret and not to compile* “combinatorially defined” automata

Some basic graph properties:

Property	Partition (X_1, \dots, X_p)	edg(X,Y)	Stable(X)	Conn(X) for degree $\leq p$	Dom(X,Y) Link(X,Y)	Path(X,Y)	Nocycle(X)
N(k)	2	k^2+k+2	2^k	$2^{O(p.p.k.k)}$	2^{2k}	$2^{O(k.k)}$	$2^{O(k.k)}$

Remark : For connectivity without degree limit, one can construct a deterministic automaton with $2^{(2^{O(k)})}$ states and the minimal one has more than $2^{(2^{(k/2)})}$ states.

Last words : MSO logic with edge set quantifications,
tree-width as parameter and the graph algebra **HR**

Edge set quantifications increase the expressive power of
MSO logic

Incidence graph of G undirected, $\mathbf{Inc}(G) = (V_G \cup E_G, \mathbf{inc}_G(.,.)).$

$\mathbf{inc}_G(v,e) \Leftrightarrow v$ is a vertex (in V_G) of edge e (in E_G).

Monadic second-order (**MSO₂**) formulas written with **inc** can use
quantifications on sets of edges.

The existence of a perfect matching or a Hamiltonian circuit is expressible
by an **MSO₂** formula, but **not by an MSO** formula.

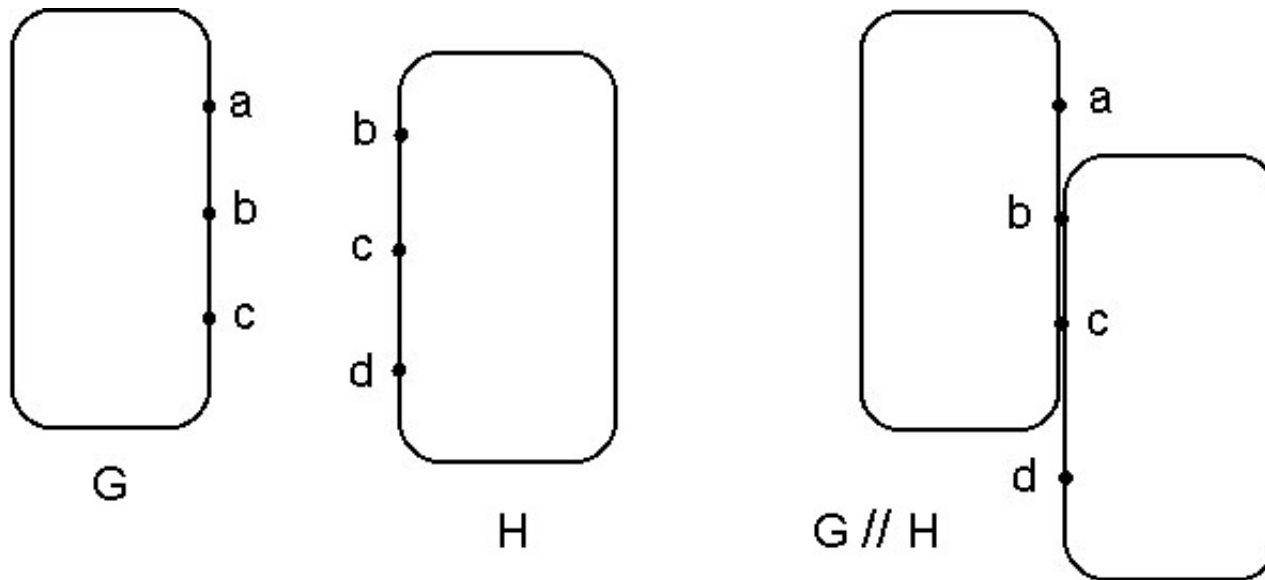
HR operations : Origin : **H**yperedge **R**eplacement hypergraph grammars ;
associated complexity measure : **tree-width**

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

Binary operation(s) : **Parallel composition**

$G // H$ is the disjoint union of G and H and sources with same label are **fused**.

(If G and H are not disjoint, one first makes a copy of H disjoint from G).



Unary operations :

Forget a source label

$Forget_a(G)$ is G without a -source: the source is no longer distinguished ;
(it is made "internal").

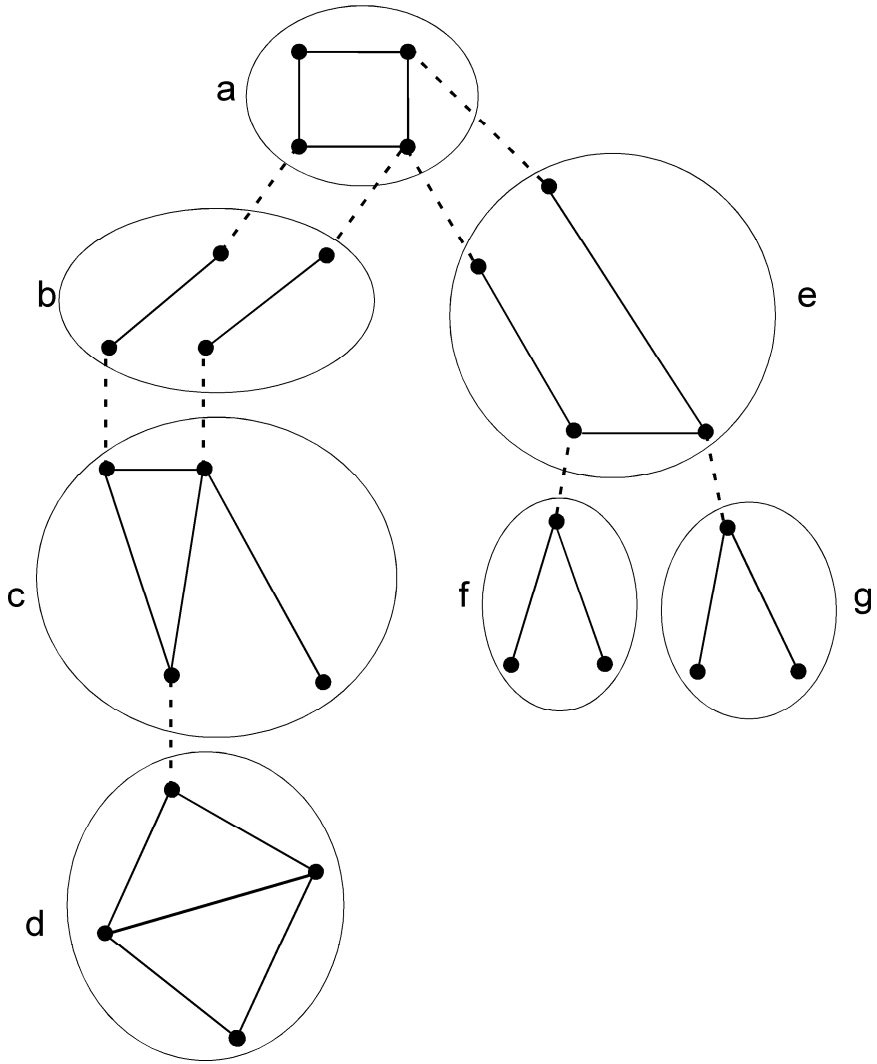
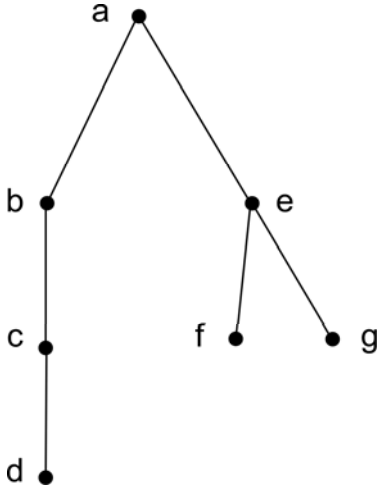
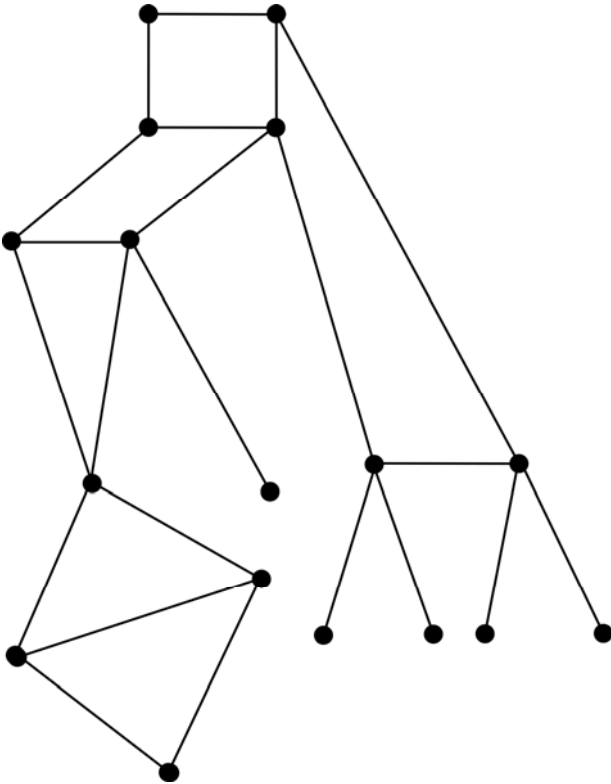
Source renaming :

$Rena_{a \leftrightarrow b}(G)$ exchanges source labels a and b
(replaces a by b if b is not the label of a source)

Nullary operations denote *basic graphs* :

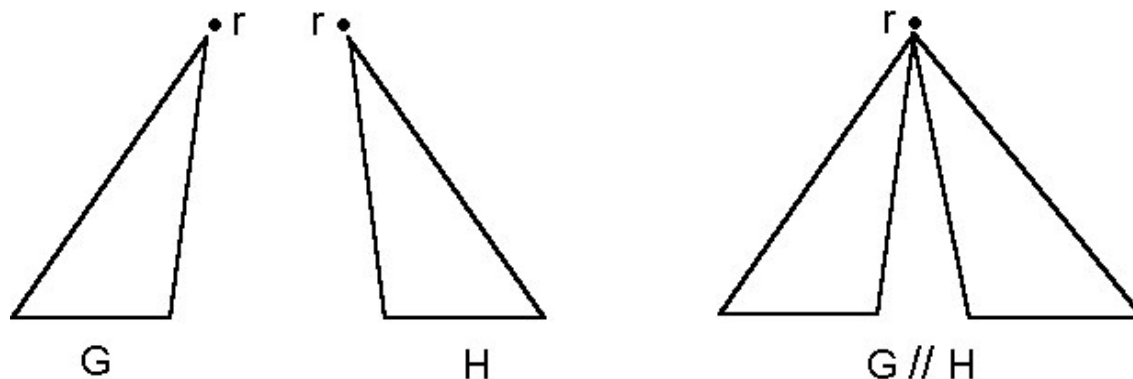
the connected graphs with at most one edge.

Tree-decompositions



Proposition: A graph has **tree-width $\leq k$** if and only if it can be constructed from basic graphs with $\leq k+1$ labels by using the operations **$//$** , **$Ren_{a \leftrightarrow b}$** and **$Forget_a$** .

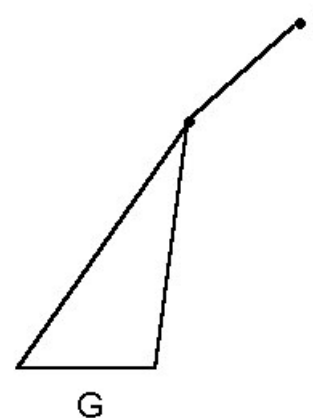
Example : Trees are of tree-width 1, constructed with two source labels, r (root) and n (new root):
Fusion of two trees at their roots :



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

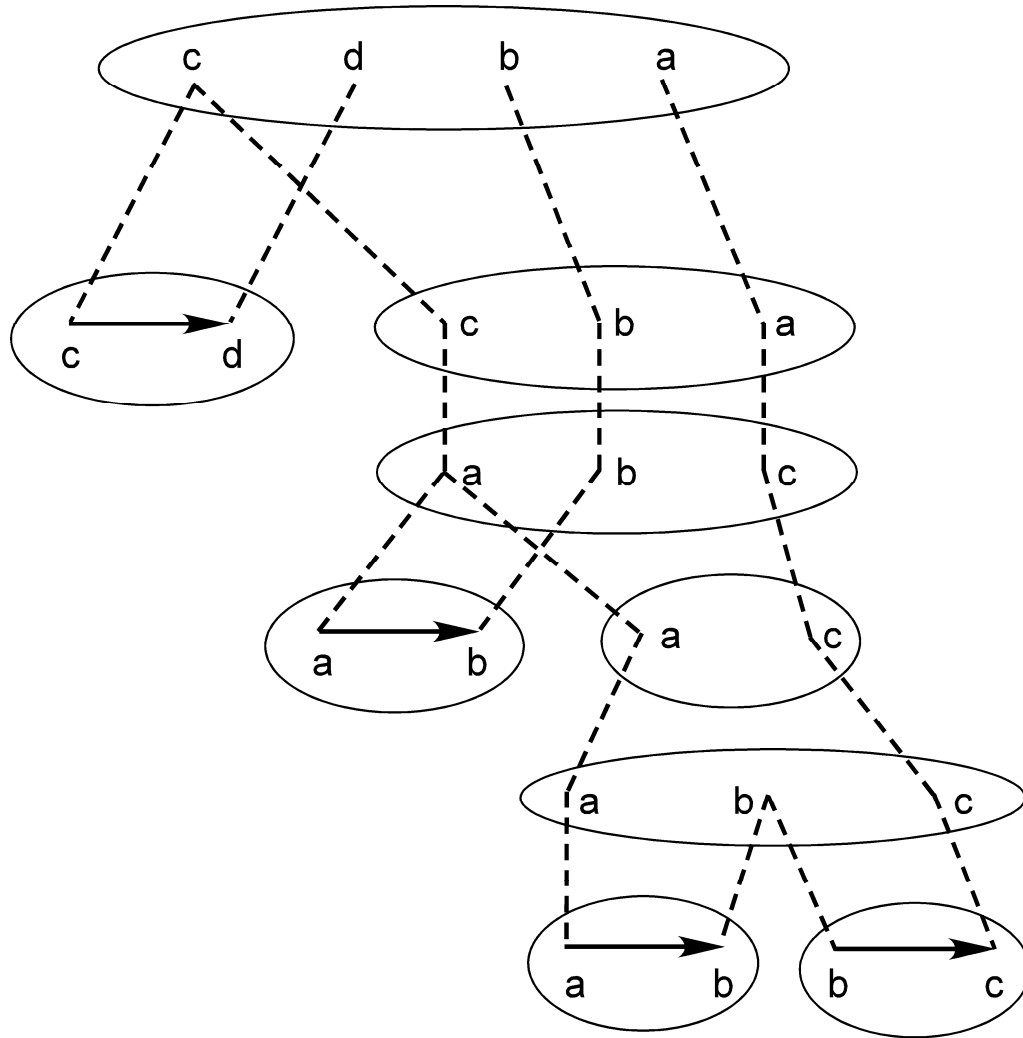
$$e = r \bullet \text{---} \bullet n$$

$$Ren_{n \leftrightarrow r} (Forget_r (G // e))$$



From an algebraic expression to a tree-decomposition

Example : $cd // \text{Ren}_{a \leftrightarrow c} (ab // \text{Forget}_b (ab // bc))$ (Constant ab denotes an edge from a to b)



The tree-decomposition associated with this term.

Automata can be constructed as shown for terms over the **VR** operations.

There is a difficulty with the bijection between occurrences in the term and the vertices and edges of the graph.

There are two possibilities :

- (1) Vertices are in bijection with the occurrences of *Forgeta*. Then, the edges are at the leaves of the syntactic tree, *below* the nodes representing their ends. The basic automaton for *adjacency* has $2^{O(k.k)}$ states. Too bad for a basic property.
- (2) Vertices are at the leaves, the edges are at nodes *above* the nodes representing their ends. Because of *//* which fuses vertices of the argument graphs, each vertex is represented by several leaves. (Equality of vertices is then an equivalence relation).

We get the same exponential blow-up.

For representing **path-decompositions**, // is not needed. This drawback disappears (Solution (2) is then OK). The operations of the **VR** algebra can be used because :

$$\text{LinearCliqueWidth}(G) \leq \text{PathWidth}(G) + 2.$$

The **VR** operations are simple, and they can represent with “linear (comb) terms” graphs of **unbounded path-width** (but bounded linear clique-width).

Furthermore, the constructed automata have **less transitions** for the same numbers of states, because they only use unary operations.