On Context-free Graph Grammars

Bruno Courcelle

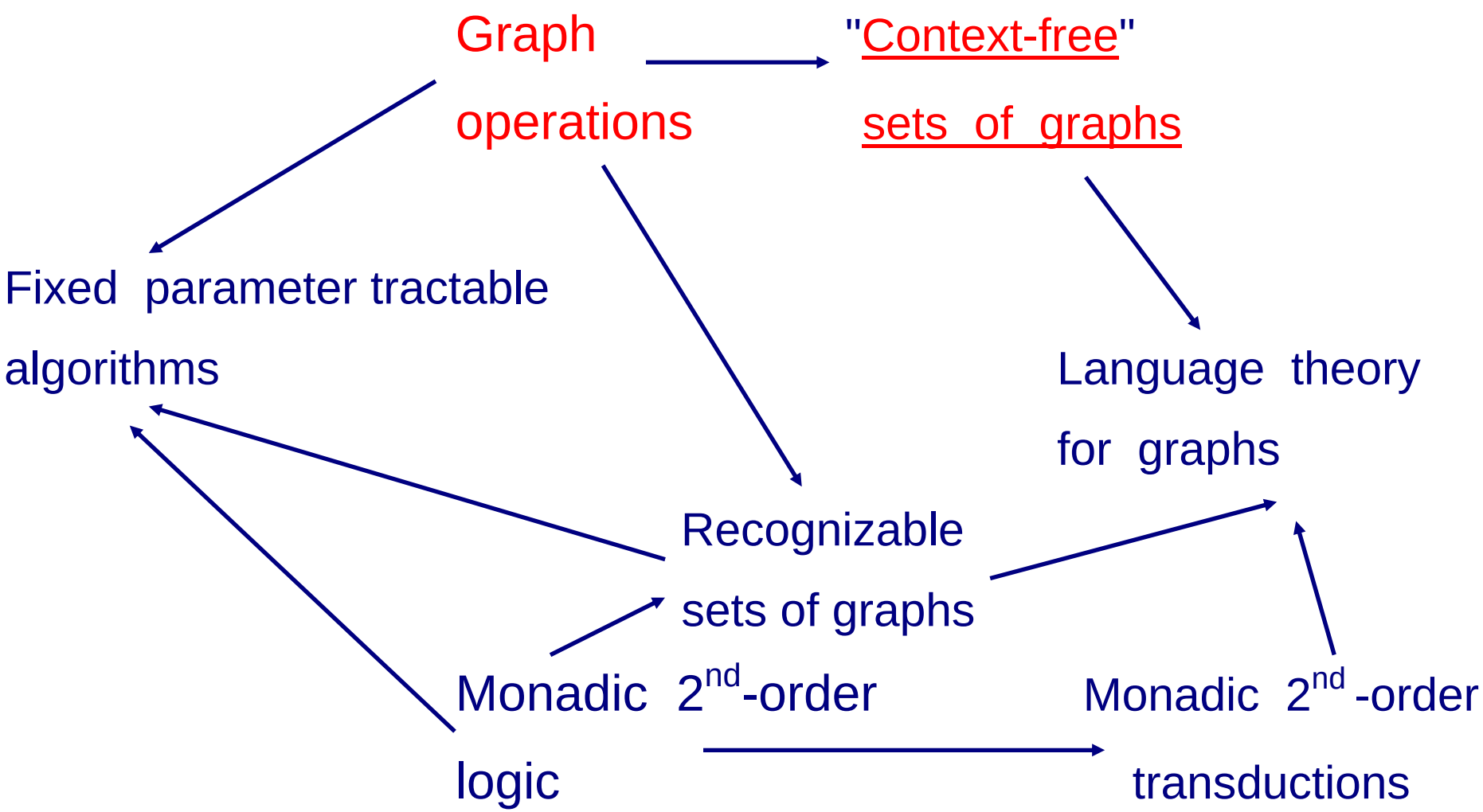
Université Bordeaux 1, LaBRI, and Institut Universitaire de France

Reference : 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

An overview chart



Grammars as mathematical objects

Linguistics : Chomsky's Hierarchy (can be refined)

Regular $\subset$ Context-free $\subset$ Context-sensitive $\subset$ Recursively enumerable

Compilation : Programming languages are described by context-free grammars + constraints (type checking etc …)

(The official motivation of lots of articles on context-free languages *unrelated* with compilation.)

Other motivations (some will apply to graphs) :

Counting objects in bijection with the words of a context-free unambiguous grammar.

Finite (compact) description of infinite (finite) sets of words, graphs, combinatorial objects.

Objects come with a structure (a derivation tree) : what is important is *not* the language but the mapping from words to derivation trees. These trees are essential in compilation but may also be used for drawing graphs generated by a context-free graph grammar.

Inductive proof methods can be based on grammars.

(Students are reluctant to prove that the grammars they produce are correct, but authors too : they never discuss how to prove that a grammar is correct).

Links with program schemes (formalization of program semantics)

-0-

What is a context-free graph grammar ?

What means “ context-free ” ?

1) Some “nonterminal” symbols S can be replaced according to a list of rules.

2) All rules S → m can be used independently of the context of S

3) The context is not modified by the replacement.

In context-free grammars, the context is the pair of words around a nonterminal

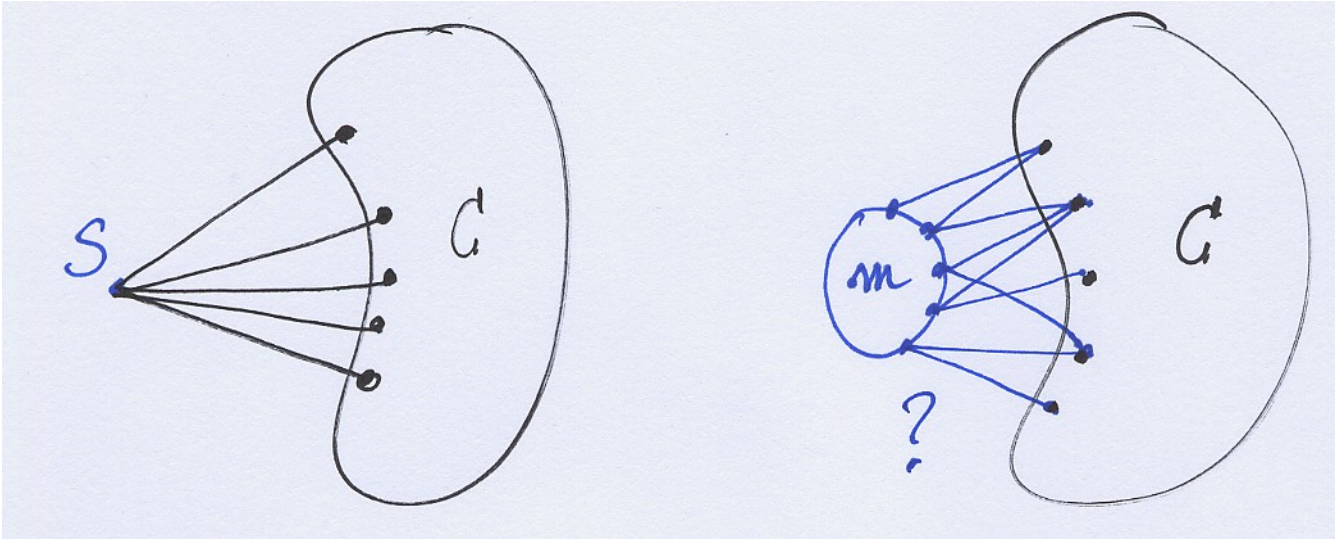
This “axiomatic” definition can be made formal.

What for graphs ?

Option 1 : Nonterminal vertices.

Application of rule

S → m



The context : A graph with particular vertices, those linked to the nonterminal vertex.

How to link the context **C** to the replacing graph **m** ?

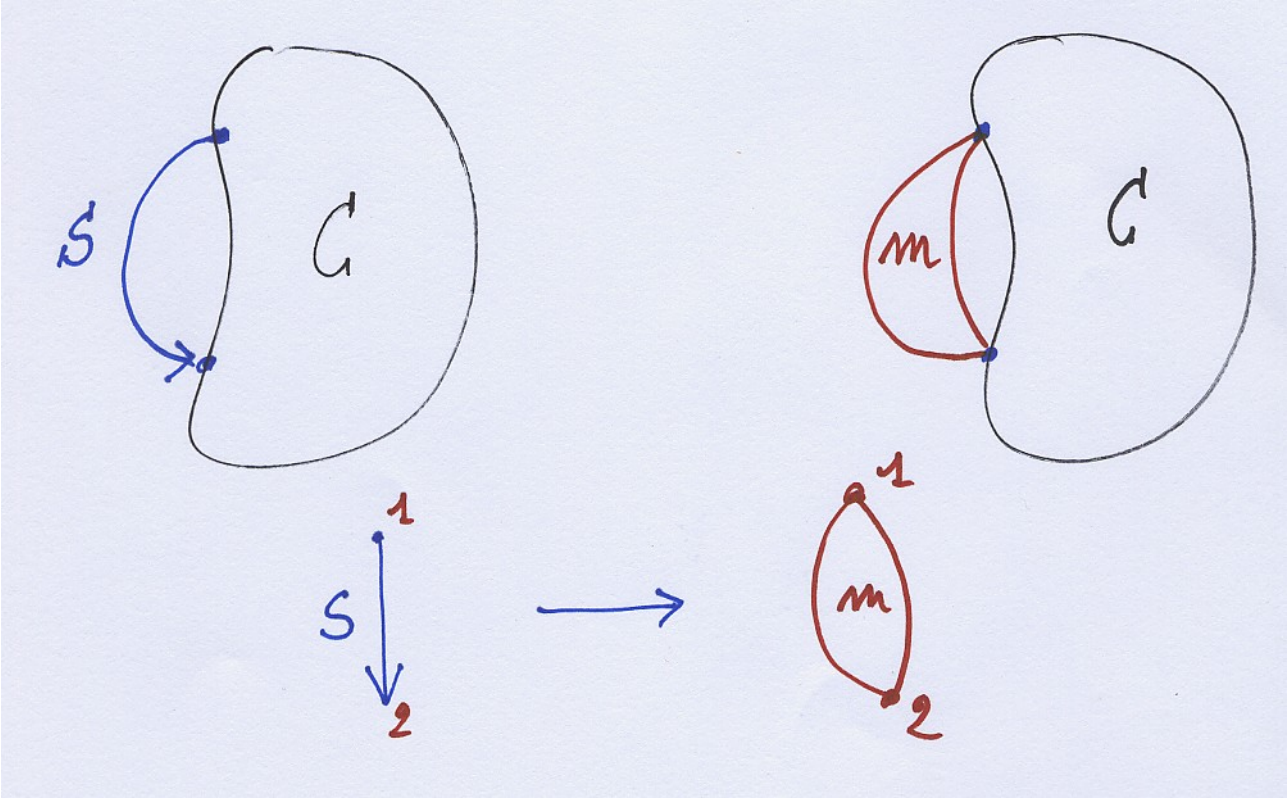
By labels attached to vertices and edges, and some complicated mechanisms

Difficulty : How to guarantee context-freeness ?

A nontrivial question (lots of articles) ; below a simple solution.

Option 2 : Nonterminal edges.

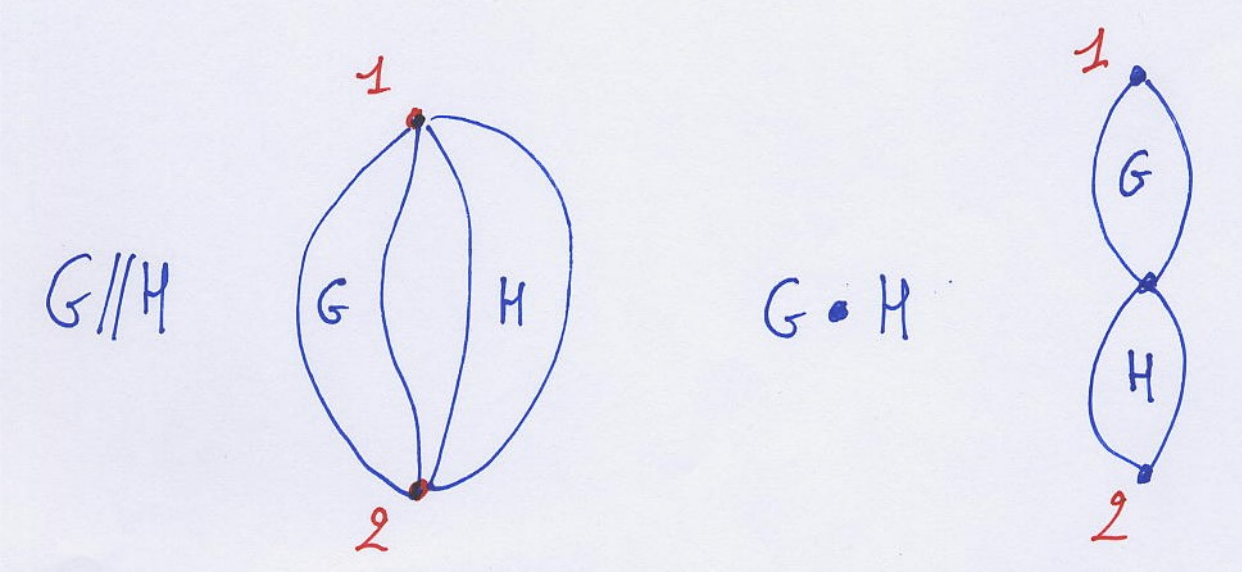
All such grammars are context-free.



Drawback : Limited generative power

Example : Series-Parallel graphs.

Graphs with distinguished vertices (*boundary vertices* or *sources*) 1 and 2, generated from **e** = 1 ⟶ 2 and the operations // (parallel-composition) and *series-composition*.

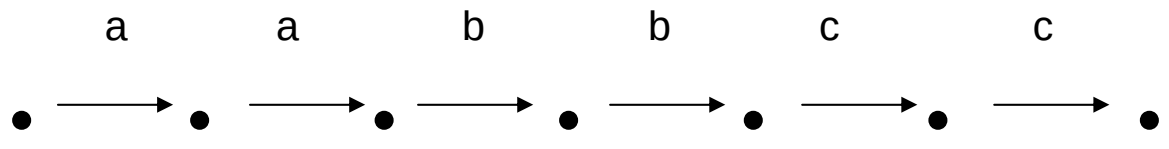


Equation : $S = S // S \cup S \bullet S \cup \mathbf{e}$

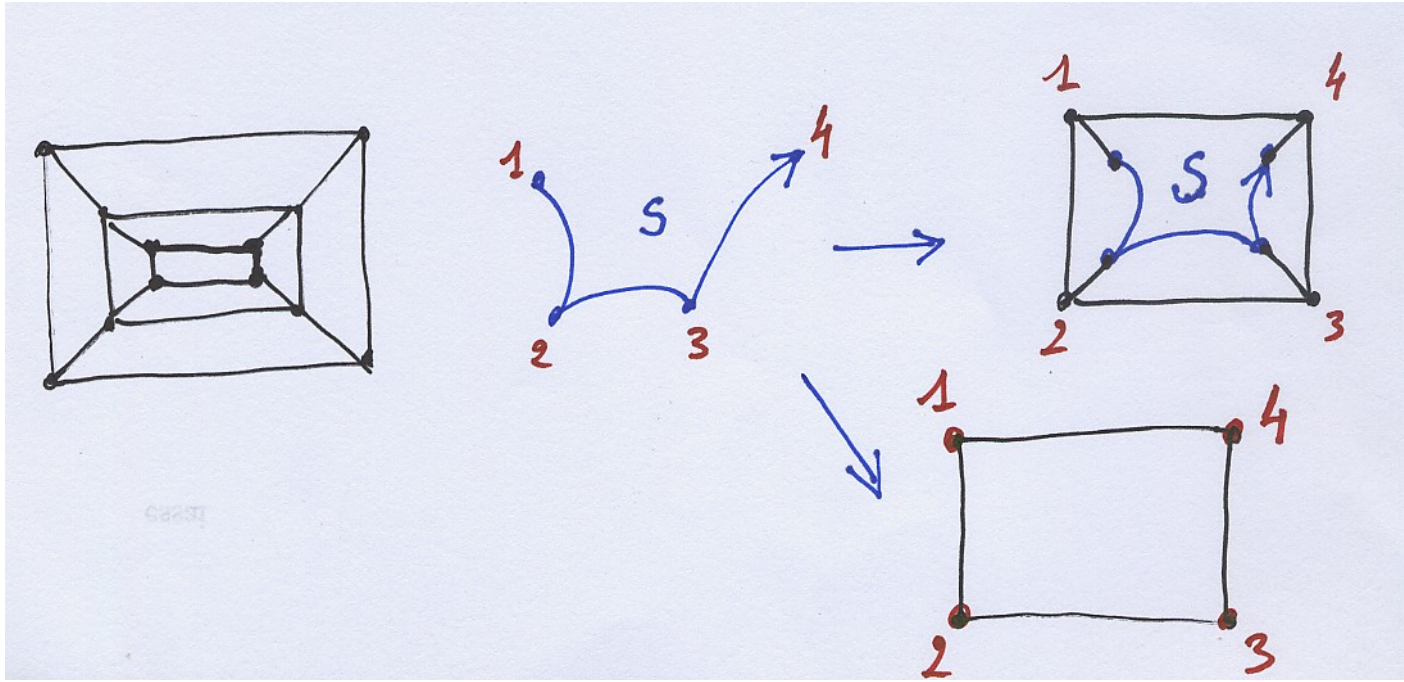
Option 3 : Nonterminal hyperedges (even to generate graphs)

Examples : the non-context-free language $a^n b^n c^n$ is a context-free set of edge

labelled graphs :



Recursive pictures :



Limitation (all 3 options): All finite (even planar) graphs : not a context-free set.

Equational sets of an algebra = the context-free sets.

Equation systems → Context-Free (Graph) Grammars

in an algebraic setting

In the case of words, the set of context-free rules

$$S \to a\ S\ T\ ; \quad S \to b\ ;\ T \to c\ T\ T\ T\ ; \quad T \to a$$

is equivalent to the system of two set equations:

$$S = a\ S\ T \cup \{ b \}$$

$$T = c\ T\ T\ T \cup \{ a \}$$

where S is the language generated by S (idem for T and T).

For graphs (or other objects) we consider systems of equations like:

$$S = f(k(S), T) \cup \{b\}$$

$$T = f(T, f(g(T), m(T))) \cup \{a\}$$

where :

f	is a binary operation,
g, k, m	are unary operations on graphs,
a, b	denote basic graphs (up to isomorphism).

An *equational set* is a component of the least (unique) solution of such an equation system. This is well-defined in any algebra (Least Fixed Point Theorem).

Many properties are valid at the general algebraic level.

Two graph algebras

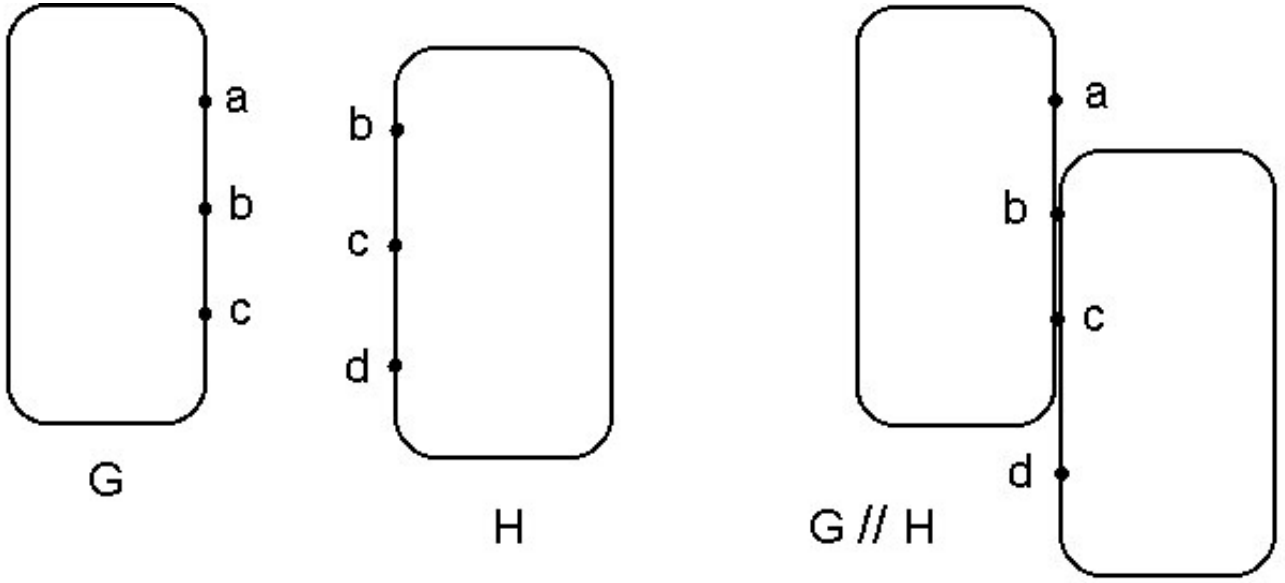
HR operations : **H**yperedge **R**eplacement hypergraph grammars ;
(associated complexity measure : tree-width)

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

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 :

$Ren_{a \leftrightarrow b}(G)$ exchanges source labels *a* and *b*

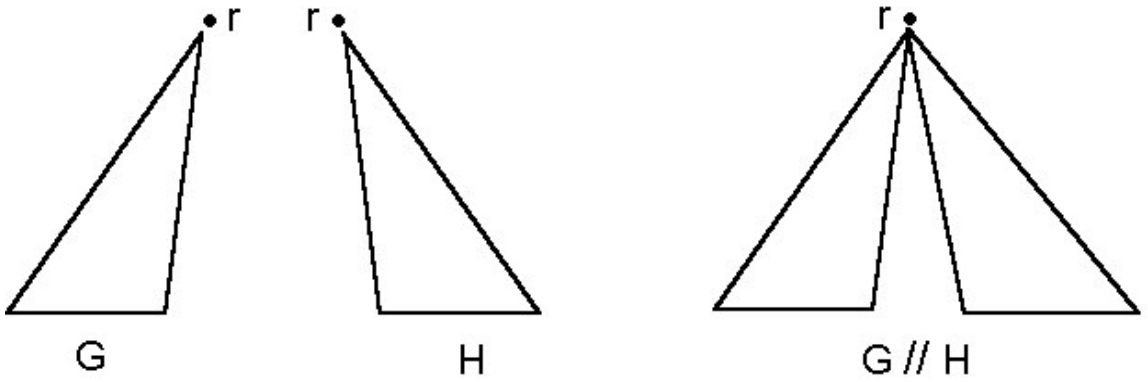
(replaces *a* by *b* if *b* is not the label of a source)

Constant symbols denote *basic graphs* : the connected graphs with at most one edge.

Remark : For generating hypergraphs, one takes more constant symbols for denoting hyperedges. The operations are the same.

Construction of trees :

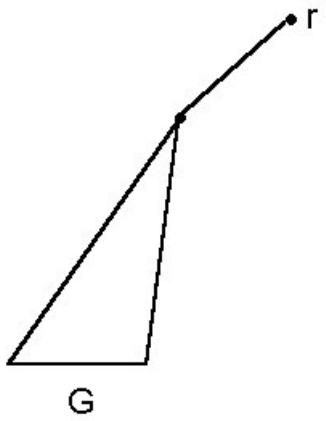
with two source labels, *r* (root) and *n* (new root): Fusion of two trees at their roots :



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

e = ***r*** ●——● ***n***

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



Equation : $T = T \,//\, T \cup Ext(T) \cup \mathbf{r}$

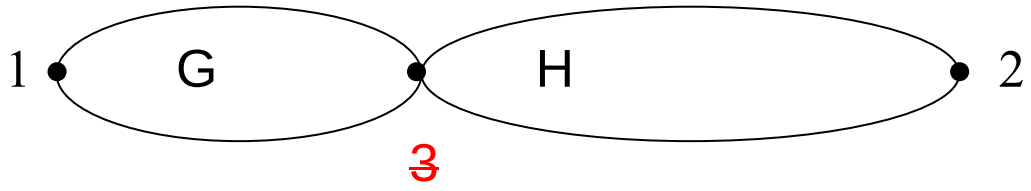
Series-parallel graphs,

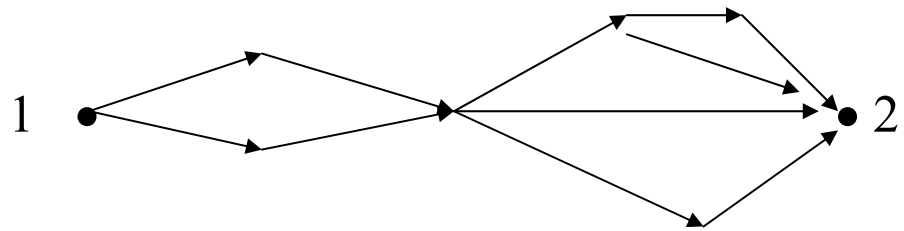
They are generated by the constant e = 1 ⟶ 2 ,

// (parallel-composition) and series-composition defined from other operations by :

$$G \bullet H = \mathit{Forget}_3(\mathit{Ren}_{2 \leftrightarrow 3}(G) \,//\, \mathit{Ren}_{1 \leftrightarrow 3}(H))$$

Example :





The defining equation (equivalent to the grammar described above) :

$$S = S \,//\, S \;\cup\; S \bullet S \;\cup\; e$$

VR operations : Another graph algebra

Origin : **V**ertex **R**eplacement graph grammars

Associated complexity measure: clique-width.

Graphs are simple, directed or not.

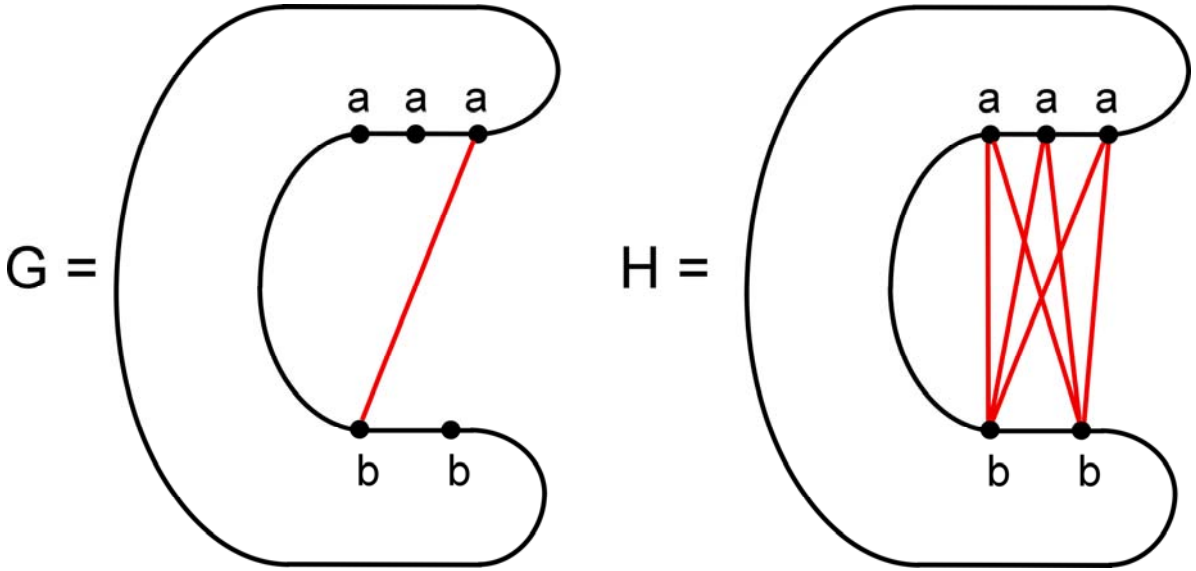
k labels : *a* , *b* , *c*, ..., *h*. Each vertex has one and only one label ;

a label *p* may label several vertices, called the *p-ports*.

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

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

$Add\text{-}edg_{a,b}$(G) is G augmented with directed or undirected edges from every *a*-port to every *b*-port.



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

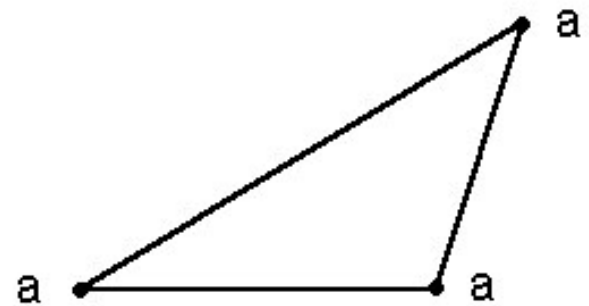
The number of added edges depends on the argument graph.

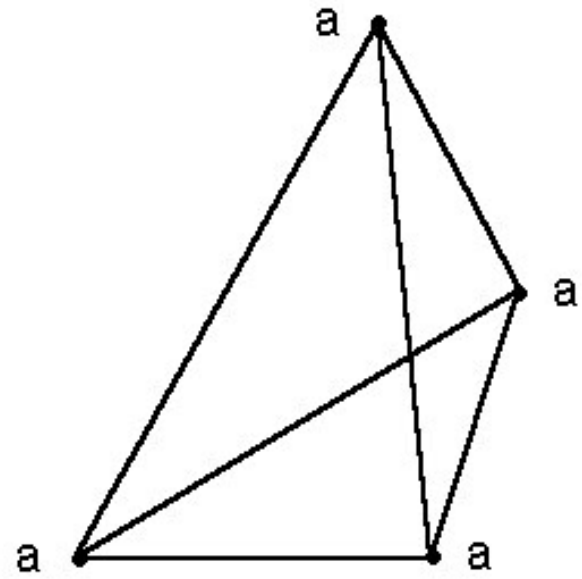
Vertex relabellings :

$Relab_{a \longrightarrow b}(G)$ is G with every vertex labelled by *a* relabelled into *b*

Basic graphs are those with a single vertex.

Example : Cliques have clique-width 2.





K_n is defined by t_n where $t_{n+1} = Relab_{b \longrightarrow a}(Add\text{-}edg_{a,b}(t_n \oplus \mathbf{b}))$

Two algebras of graphs **HR** and **VR**

Hence, two notions of context-free sets, defined as the equational sets of the algebras **HR** and **VR**.

Why not a third algebra ? :

We have robustness results :
Independent logical characterizations, stability under certain logically defined transductions, generation from trees.

Which properties follow from the algebraic setting ?

Answers : Closure under union, // , $\oplus$ and the unary operations.
Emptiness and finiteness are decidable (finite sets are computable)
Parikh's Theorem
Derivation trees, denotation of generated graphs by terms,
Upper bounds to tree-width and clique-width.

Which properties do not hold as we could wish ?

Answers : The set of all (finite) graphs is neither HR- nor VR-equational.

Not even is the set of all square grids (planar graphs of degree 4)

Parsing is sometimes NP-complete.

Comparison of the two classes :

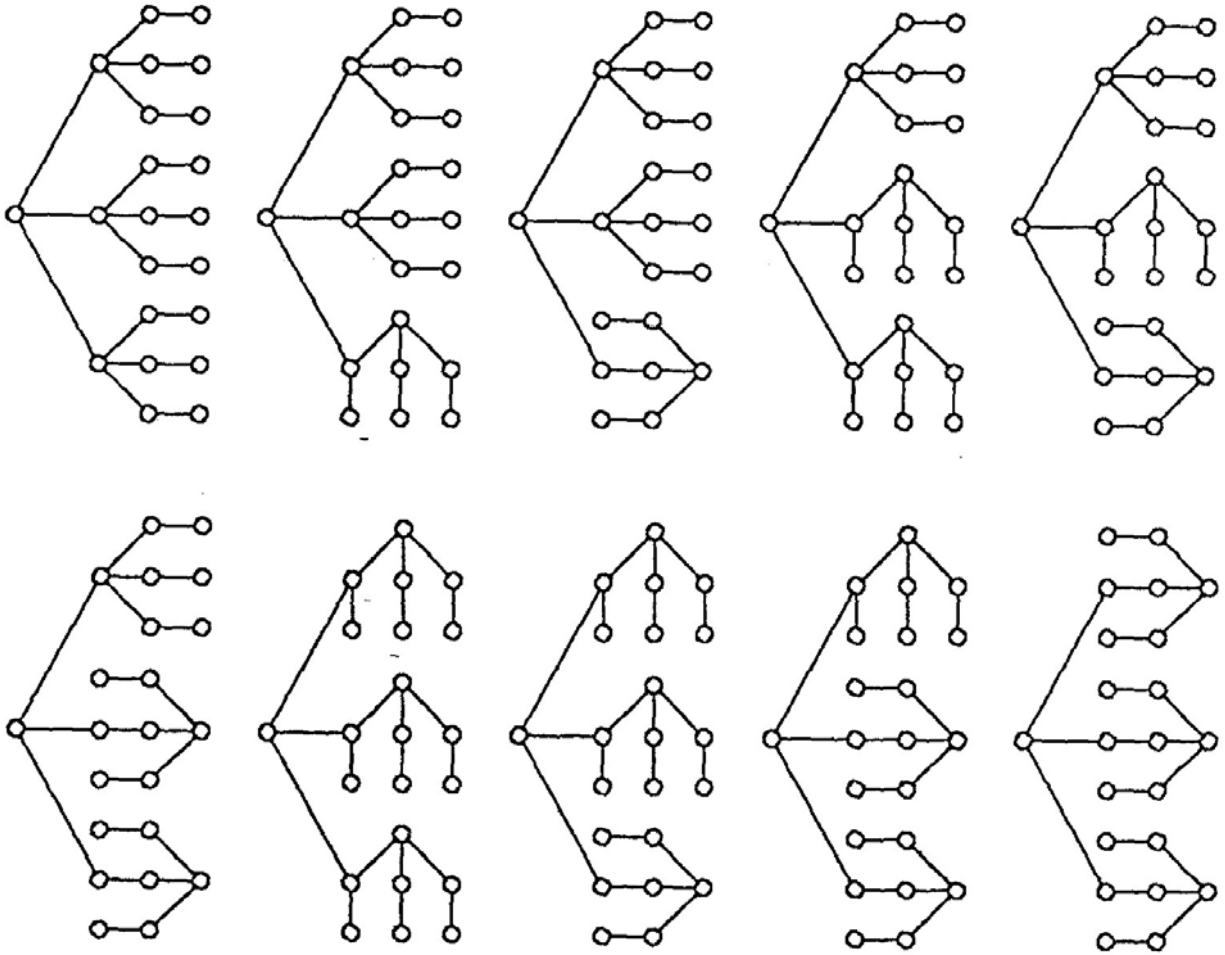
Equat(**HR**) $\subseteq$ Equat(**VR**)

= sets in Equat(**VR**) , all graphs of which are without some fixed $K_{n,n}$ as subgraph.

$K_{n,p}$: All edges between a set of *n* vertices and a set of *p* vertices.

Compact descriptions of finite sets

Set T_2



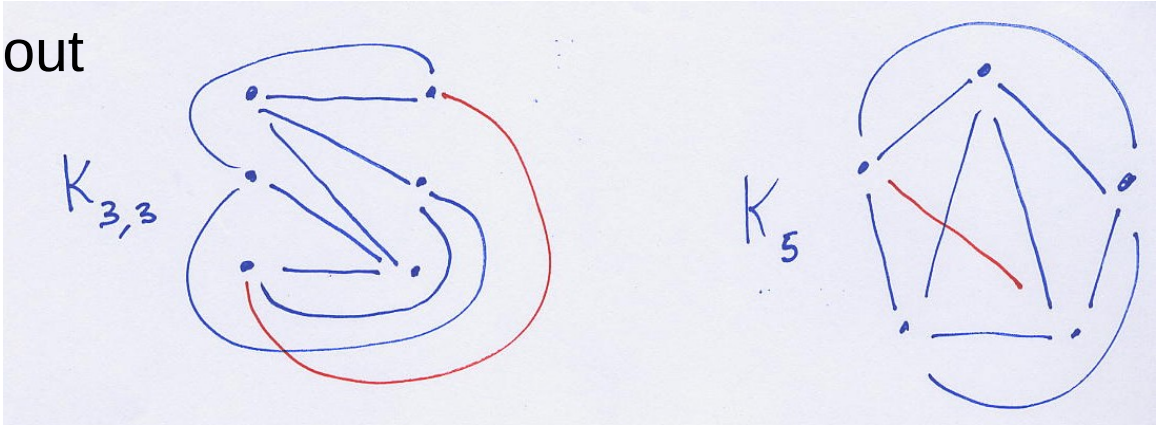
What do they come from ?

Graphs described by "forbidden subgraphs or minors"

Planar graphs = graphs without

K_5 and $K_{3,3}$ as "minors"

(some notion of subgraph).



Theory developped by Robertson, Seymour and many others.

In many cases finite but very large numbers of forbidden configurations.

Graphs on the torus ("doughnut") : *thousands* of forbidden graphs.

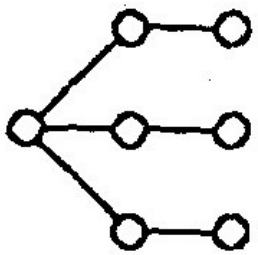
Certainly not random sets.

Grammars should be able to enlighten the regularities.

The set T_2 : the *trees* that are the forbidden minors for the property "pathwidth ≤ 2" (graphs having a kind of linear decomposition).

T_k is the corresponding set for "path-width $\leq$ k" where :

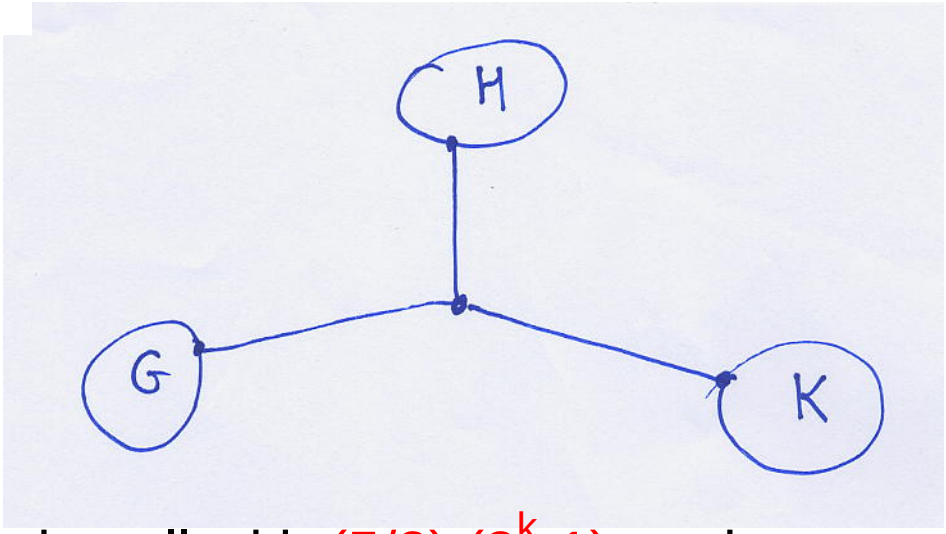
T_1 consists of



$T_{k+1} = S(T_{k+1}, T_{k+1}, T_{k+1})$

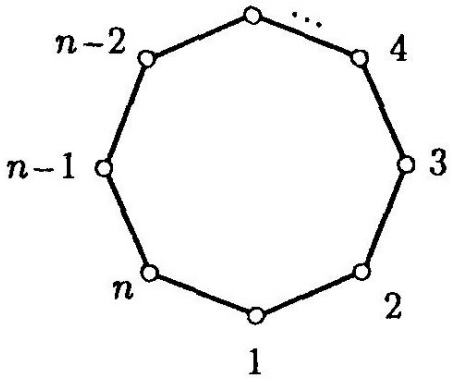
S(A,B,C) = set of star-compositions :

for all $G \in A$, $H \in B$, $K \in C$.

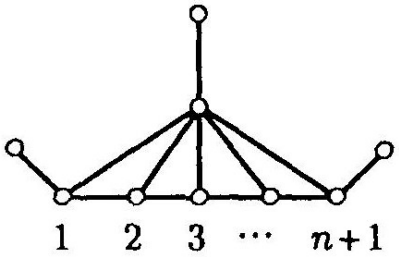


Each set T_k has more than $(k!)^2$ graphs, all with $(5/2).(3^k-1)$ vertices, but has an HR grammar (equation system) of size O(k).

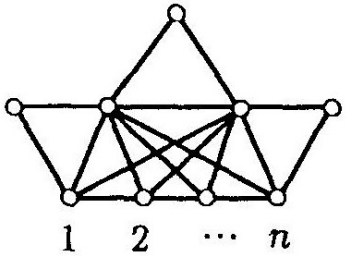
Other example : the forbidden induced subgraphs for *interval graphs*. There are infinitely many, but they form an equational set of the HR algebra.



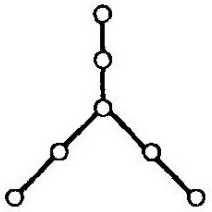
$G_n ; n \geq 4$



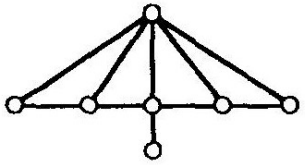
$K_n ; n \geq 1$



$L_n ; n \geq 1$



B_1



B_2

Open problem :

Design systematic methods to construct "small" context-free HR- or VR-grammars (or of other types) to represent sets of forbidden configurations.

Tools : Monadic second-order logic + algebraic notions (equational and recognizable sets) + graph theoretic arguments.

Inductive proofs based on context-free grammars / equation systems

Example

$T \rightarrow aTb \;;\; T \rightarrow c \quad / \quad T = aTb \cup c$

Property 1 : Every word generated by T has odd length.

From grammar : For every word w in {a,b,c}*, by induction on n such that $T \rightarrow^{n} w$ (n derivation steps).

From equation system : Let K be the set of words of odd length.

Fact : $aKb \subseteq K$ and $c \in K$.

This gives the result by the Least Fixed-Point Theorem.

Same equation : $T = aTb \cup c$

Property 2 : $T \subseteq K' := X^* - X^*abX^*$ (no factor ab ; X ={a,b,c}*)

False that : $a\,K'\,b \subseteq K'$

A stronger inductive property is needed. One can use

$$K'' := K' \cap (X^* - X^*a) \cap (X - bX^*)$$

Theorem : Let G be a context-free grammar defined by equations :

$$X_1 = p_1, \ldots, x_n = p_n.$$

Let K be a regular language. Then $L(G, X_1) \subseteq K$

$\Leftrightarrow$ there exist regular languages $K_1, \ldots, K_n$ such that :

$$K_1 \subseteq K \text{ and } p_i(K_1,\ldots,K_n) \subseteq K_i \text{ for each } i.$$

The property $L(G, X_1) \subseteq K$ can be proved by lemmas concerning only regular languages.

A similar situation holds for graphs, where “regular language” is replaced by “set of graphs characterized by a monadic second-order sentence.”

Attribute grammars

Motivation from compilation

Nonterminal symbols are equipped with “attributes” taking values in “types” (integer, real, array, etc…) or “register” (for code generation).

Context-free rules are equipped computation rules of attributes.

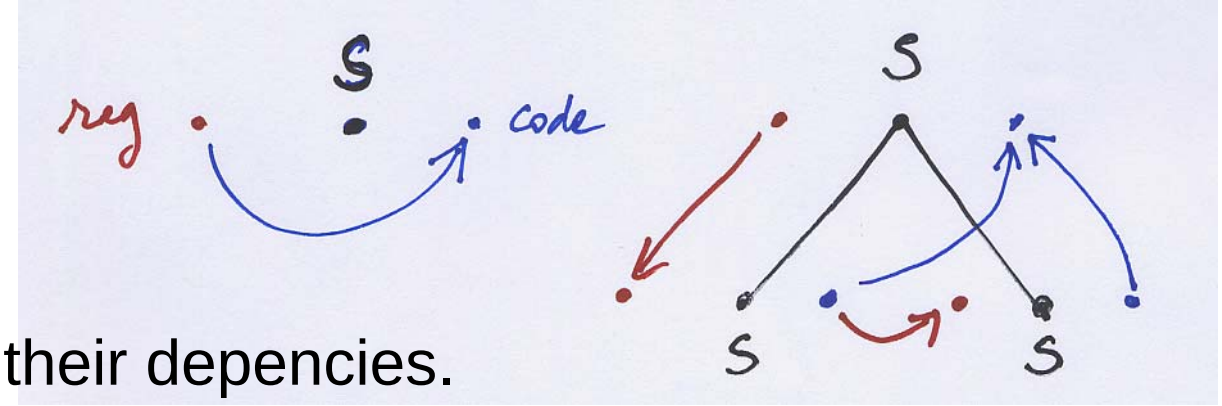
Principle : For every derivation tree, the dependency graph of attributes must have no circuits.

Rather than giving (too strong) syntactic restrictions guaranteeing that, the non-circularity test is performed after attribute dependencies are defined.

Example :

S → *variable*

S→ S + S

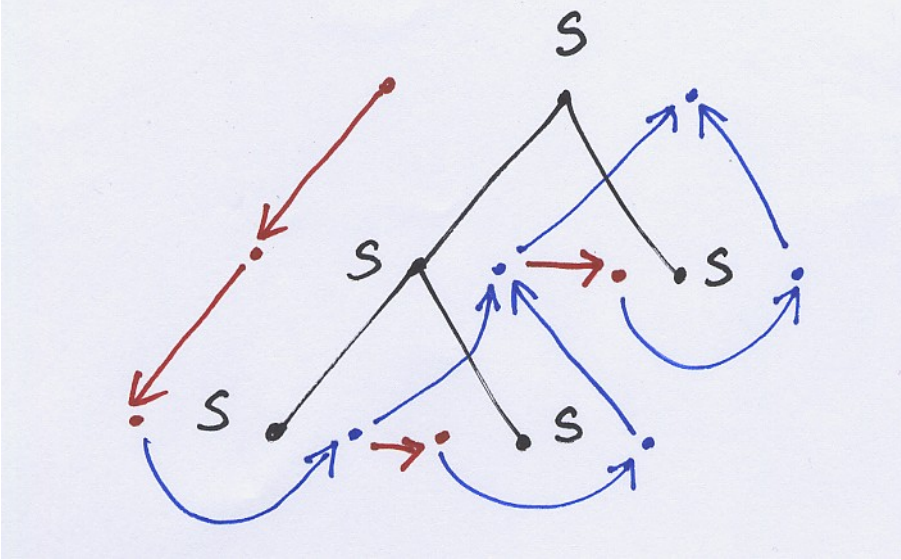


Two attributes and their depencies.

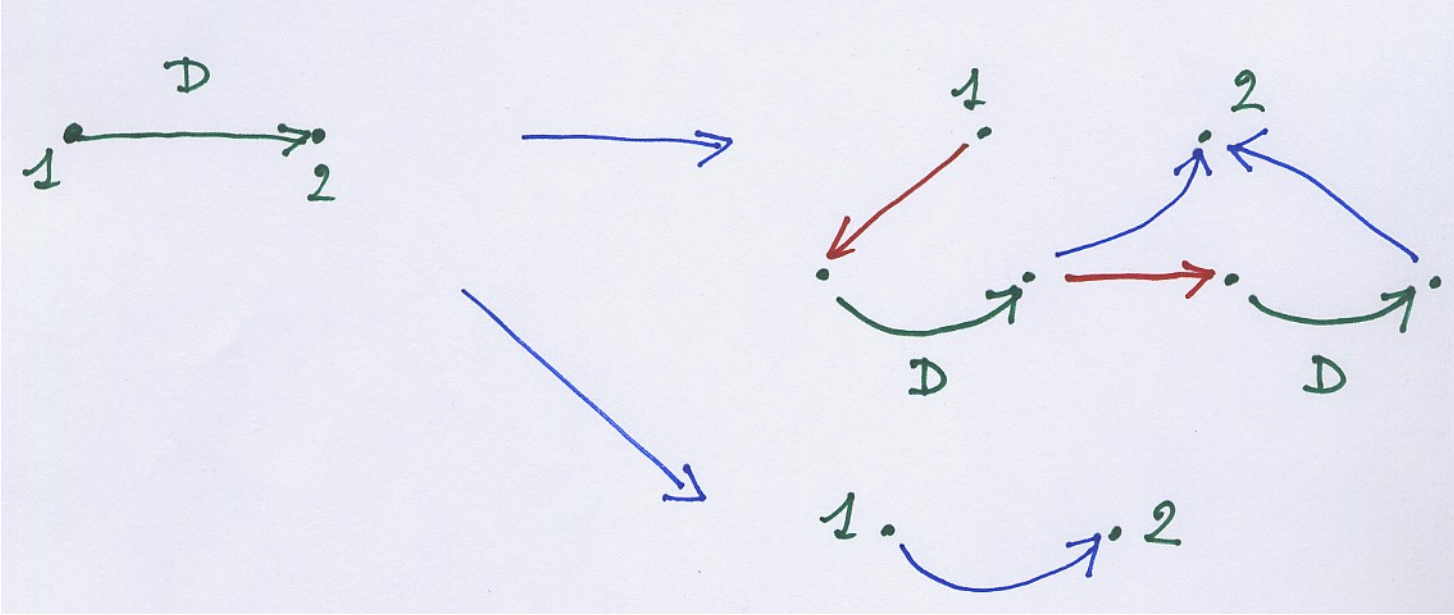
The dependency graph for the expression :

x + y + z

(x,y,z are variables).



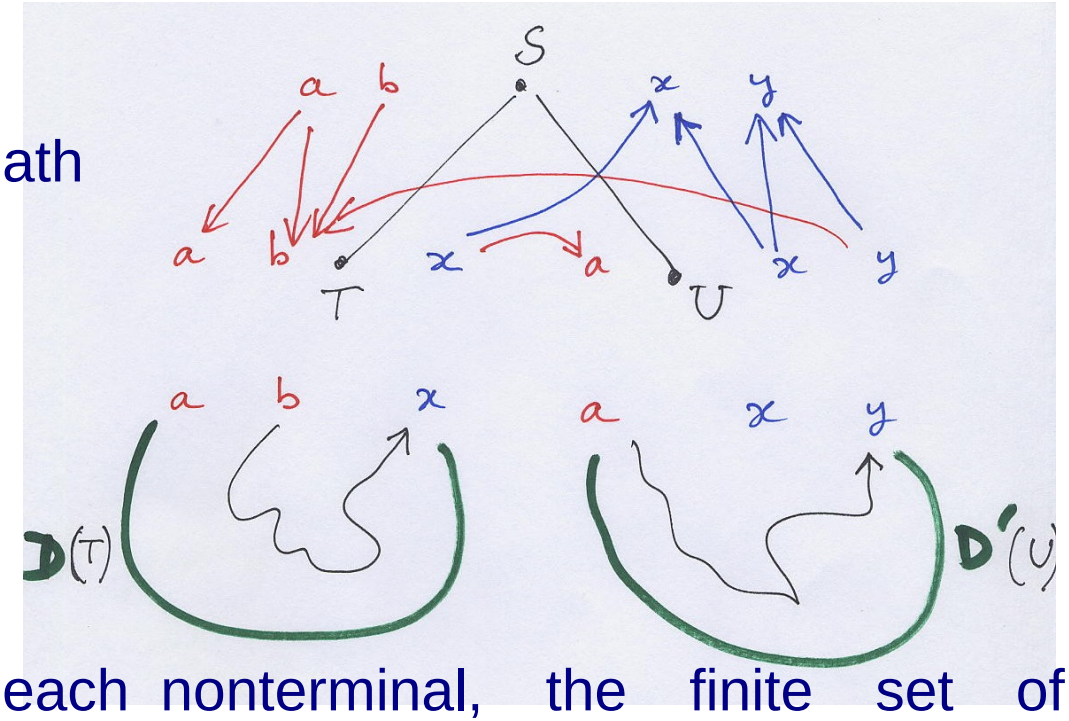
The Hyperedge Replacement grammar generating the dependencies for all words generated by S.



The non-circularity checking algorithm

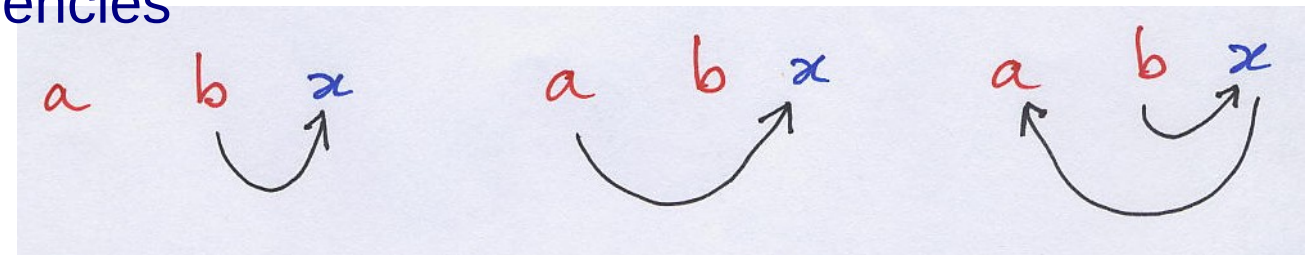
(exponential in extreme cases but practically usable)

There is a circularity if some dependency graph of T has a path from b to x (“root attributes”) and some dependency graph of U has one from a to y.



The algorithm constructs, for each nonterminal, the finite set of possible “types of dependencies” between its root attributes.

For T we may have :



Generalization

To all HR and VR graph grammars,

To all properties expressed in monadic second-order logic

(extending the non-circularity question),

Auxiliary properties (extending the possible “types” of dependencies)

i.e., “stronger inductive assertions” can be generated by an algorithm,

(no need to “guess” the right inductive property).

Consequence : linear time verification from the “derivation tree”.

Difficulties : 1) Huge numbers of auxiliary properties.

2) Parsing is sometimes NP-complete, anyway difficult.

Other examples of inductive proofs

Example : *Series-parallel graphs*

1) G, H connected implies : G // H and G • H are connected, (induction)

   e is connected (basis) : $\Rightarrow$ All series-parallel graphs are connected.

2) It is not true that :

   G and H planar implies : G//H is planar (K_5 = H//e).

   A stronger property for induction :

   G has a planar embedding with the sources in the same “face”

   $\Rightarrow$ All series-parallel graphs are planar.

Inductive computation : Test for 2-colorability of series-parallel graphs

Not all series-parallel graphs are 2-colorable. Example : K_3

G, H 2-colorable does not imply that G//H is 2-colorable (because $K_3 = P_3//e$).

One can check 2-colorability with 2 auxiliary properties :

Same(G) = G is 2-colorable with sources of the same color,
Diff(G) = G is 2-colorable with sources of different colors

by using rules :

Diff(e) = True ; Same(e) = False

$\text{Same}(G//H) \Leftrightarrow \text{Same}(G) \wedge \text{Same}(H)$
$\text{Diff}(G//H) \Leftrightarrow \text{Diff}(G) \wedge \text{Diff}(H)$

$\text{Same}(G \bullet H) \Leftrightarrow (\text{Same}(G) \wedge \text{Same}(H)) \vee (\text{Diff}(G) \wedge \text{Diff}(H))$
$\text{Diff}(G \bullet H) \Leftrightarrow (\text{Same}(G) \wedge \text{Diff}(H)) \vee (\text{Diff}(G) \wedge \text{Same}(H))$

We can compute for every SP-term t, by induction on the structure of t the pair of Boolean values (Same(Val(t)) , Diff(Val(t))).

We get the answer for G = Val(t) (the graph that is the *value* of t) regarding 2-colorability.

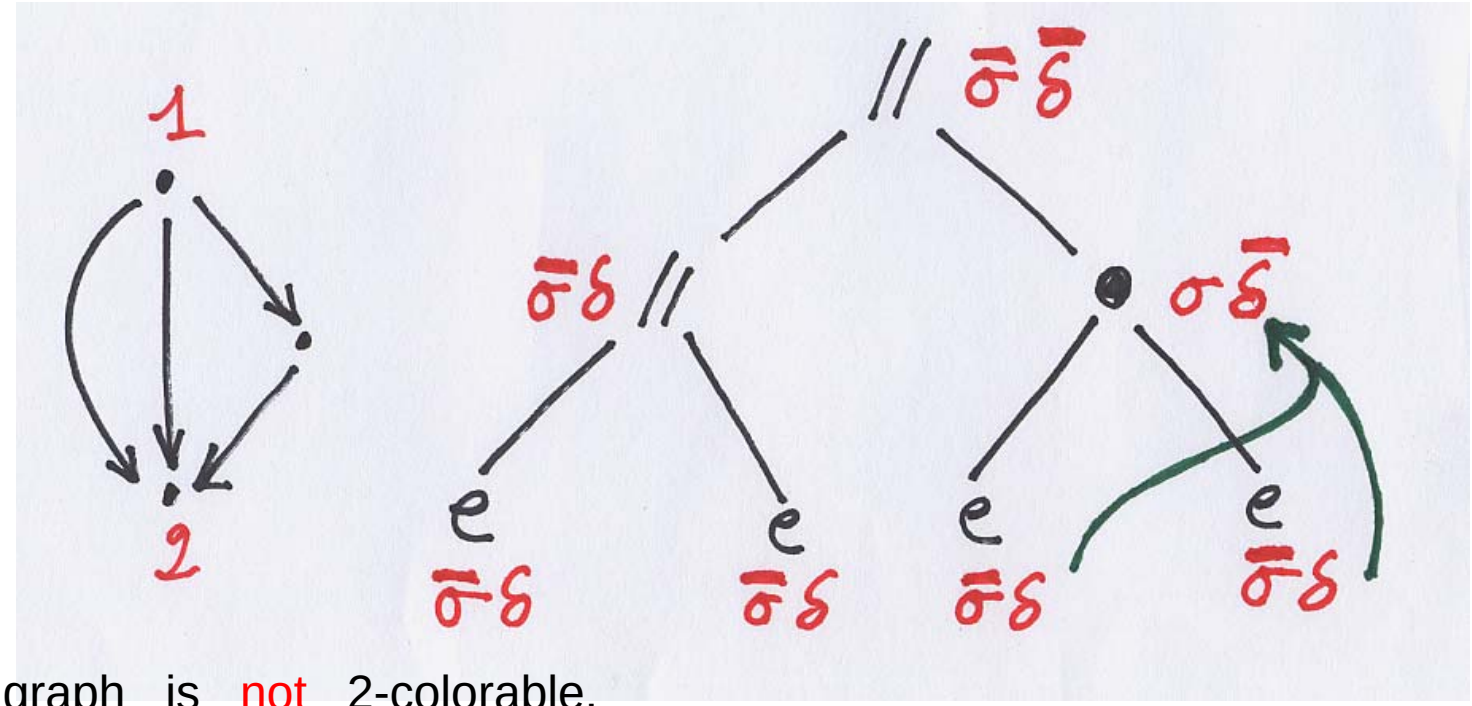
Application 1 : Linear algorithm

For every SP-term t, we can compute, by running a finite deterministic bottom-automaton on t, the pair of Boolean values (Same(Val(t)) , Diff(Val(t))).

We get the answer for G = Val(t) (the graph that is the *value* of t) regarding 2-colorability.

Example : σ at node u means that Same(Val(t/u)) is true, $\overline{\sigma}$ that it is false,

δ that Diff (Val(t/u)) is true, etc… Computation is **done bottom-up** with the rules :



The graph is not 2-colorable.

Application 2 : Equation system for 2-colorable series-parallel graphs

We let $S_{\sigma,\delta}$ be the set of series-parallel graphs that satisfy Same (σ) and Diff (δ)
$S_{\sigma,\bar{\delta}}$ be the set of those that satisfy Same and not Diff , etc …

From the equation : $S = S // S \cup S \bullet S \cup e$ we get the equation system :

$$(a)\quad S_{\sigma,\delta} = S_{\sigma,\delta}//S_{\sigma,\delta} \cup S_{\sigma,\delta} \bullet S_{\sigma,\delta} \cup S_{\sigma,\delta} \bullet S_{\sigma,\bar{\delta}} \cup S_{\sigma,\delta} \bullet S_{\bar{\sigma},\delta} \cup S_{\sigma,\bar{\delta}} \bullet S_{\sigma,\delta} \cup S_{\bar{\sigma},\delta} \bullet S_{\sigma,\delta}$$

$$(b)\quad S_{\bar{\sigma},\delta} = e \cup S_{\bar{\sigma},\delta}//S_{\bar{\sigma},\delta} \cup S_{\sigma,\delta}//S_{\bar{\sigma},\delta} \cup S_{\bar{\sigma},\delta}//S_{\sigma,\delta} \cup S_{\sigma,\bar{\delta}} \bullet S_{\bar{\sigma},\delta} \cup S_{\bar{\sigma},\delta} \bullet S_{\sigma,\bar{\delta}}$$

$$(c)\quad S_{\sigma,\bar{\delta}} = S_{\sigma,\delta}//S_{\sigma,\bar{\delta}} \cup S_{\sigma,\bar{\delta}}//S_{\sigma,\delta} \cup S_{\sigma,\bar{\delta}}//S_{\sigma,\bar{\delta}} \cup S_{\sigma,\bar{\delta}} \bullet S_{\sigma,\bar{\delta}} \cup S_{\bar{\sigma},\delta} \bullet S_{\bar{\sigma},\delta}$$

$$\begin{aligned}(d)\quad S_{\bar{\sigma},\bar{\delta}} = {} & S_{\bar{\sigma},\bar{\delta}}//S_{\bar{\sigma},\bar{\delta}} \cup S_{\bar{\sigma},\bar{\delta}}//S_{\sigma,\delta} \cup S_{\bar{\sigma},\bar{\delta}}//S_{\bar{\sigma},\delta} \cup S_{\bar{\sigma},\bar{\delta}}//S_{\sigma,\bar{\delta}} \cup \\ & S_{\sigma,\delta}//S_{\bar{\sigma},\bar{\delta}} \cup S_{\bar{\sigma},\delta}//S_{\bar{\sigma},\bar{\delta}} \cup S_{\sigma,\bar{\delta}}//S_{\bar{\sigma},\bar{\delta}} \cup S_{\bar{\sigma},\delta}//S_{\sigma,\bar{\delta}} \cup \\ & S_{\sigma,\bar{\delta}}//S_{\bar{\sigma},\delta} \cup S_{\bar{\sigma},\bar{\delta}} \bullet S_{\sigma,\delta} \cup S_{\bar{\sigma},\bar{\delta}} \bullet S_{\bar{\sigma},\delta} \cup S_{\bar{\sigma},\bar{\delta}} \bullet S_{\sigma,\bar{\delta}} \cup \\ & S_{\bar{\sigma},\bar{\delta}} \bullet S_{\bar{\sigma},\bar{\delta}} \cup S_{\sigma,\delta} \bullet S_{\bar{\sigma},\bar{\delta}} \cup S_{\bar{\sigma},\delta} \bullet S_{\bar{\sigma},\bar{\delta}} \cup S_{\sigma,\bar{\delta}} \bullet S_{\bar{\sigma},\bar{\delta}}\end{aligned}$$

In equation

$$(a)\quad S_{\sigma,\delta} = S_{\sigma,\delta}//S_{\sigma,\delta} \cup S_{\sigma,\delta} \bullet S_{\sigma,\delta} \cup S_{\sigma,\delta} \bullet S_{\sigma,\bar{\delta}} \cup S_{\sigma,\delta} \bullet S_{\bar{\sigma},\delta} \cup S_{\sigma,\bar{\delta}} \bullet S_{\sigma,\delta} \cup S_{\bar{\sigma},\delta} \bullet S_{\sigma,\delta}$$

$S_{\sigma,\delta}$ is in all terms of the righthand side. Hence, it defines (least solution) the empty set. This proves (a small theorem) :

Fact : No series-parallel graph satisfies Same and Diff.

We can simplify the system {(a), (b), (c), (d)} into :

$$(b')\quad S_{\bar{\sigma},\delta} = e \cup S_{\bar{\sigma},\delta}//S_{\bar{\sigma},\delta} \cup S_{\sigma,\bar{\delta}} \bullet S_{\bar{\sigma},\delta} \cup S_{\bar{\sigma},\delta} \bullet S_{\sigma,\bar{\delta}}$$

$$(c')\quad S_{\sigma,\bar{\delta}} = S_{\sigma,\bar{\delta}}//S_{\sigma,\bar{\delta}} \cup S_{\sigma,\bar{\delta}} \bullet S_{\sigma,\bar{\delta}} \cup S_{\bar{\sigma},\delta} \bullet S_{\bar{\sigma},\delta}$$

$$(d')\quad S_{\bar{\sigma},\bar{\delta}} = S_{\bar{\sigma},\bar{\delta}}//S_{\bar{\sigma},\bar{\delta}} \cup S_{\bar{\sigma},\bar{\delta}}//S_{\bar{\sigma},\delta} \cup S_{\bar{\sigma},\bar{\delta}}//S_{\sigma,\bar{\delta}} \cup S_{\bar{\sigma},\delta}//S_{\sigma,\bar{\delta}} \cup S_{\bar{\sigma},\bar{\delta}} \bullet S_{\bar{\sigma},\delta} \cup S_{\bar{\sigma},\bar{\delta}} \bullet S_{\sigma,\bar{\delta}} \cup S_{\bar{\sigma},\bar{\delta}} \bullet S_{\bar{\sigma},\bar{\delta}} \cup S_{\bar{\sigma},\delta} \bullet S_{\bar{\sigma},\bar{\delta}} \cup S_{\sigma,\bar{\delta}} \bullet S_{\bar{\sigma},\bar{\delta}}$$

By replacing $S_{\sigma,\bar{\delta}}$ by T_σ, $S_{\bar{\sigma},\delta}$ by T_δ, by using commutativity of // , we get the system

$$\begin{cases} T = T_\sigma \cup T_\delta & \text{(defining 2-colorable series-parallel graphs)} \\ T_\sigma = T_\sigma//T_\sigma \cup T_\sigma \bullet T_\sigma \cup T_\delta \bullet T_\delta \\ T_\delta = e \cup T_\delta//T_\delta \cup T_\sigma \bullet T_\delta \cup T_\delta \bullet T_\sigma \end{cases}$$

Recognizability and inductive properties

Definitions : A set P of properties on an F-algebra **M** is F-inductive if, for every $p \in P$ and $f \in F$, there exists a (known) Boolean formula B such that :

$$p(f_{\mathbf{M}}(a,b)) = B[\ldots,q(a),\ldots,q'(b),\ldots.] \text{ for all } a \text{ and } b \text{ in } \mathbf{M}$$

(here $q, q' \in P$, $q(a),\ldots, q(b) \in \{True, False\}$) .

A subset L of **M** is recognizable if and only if it is the set of elements that satisfy a property belonging to a finite inductive set P of properties.

This generalizes the characterization of regular languages in terms of finite congruences (or of their finite syntactical monoid).

Inductive properties and automata on terms

The simultaneous computation of m inductive properties can be implemented by a *finite deterministic bottom-up automaton* with 2^m states, running on terms t.

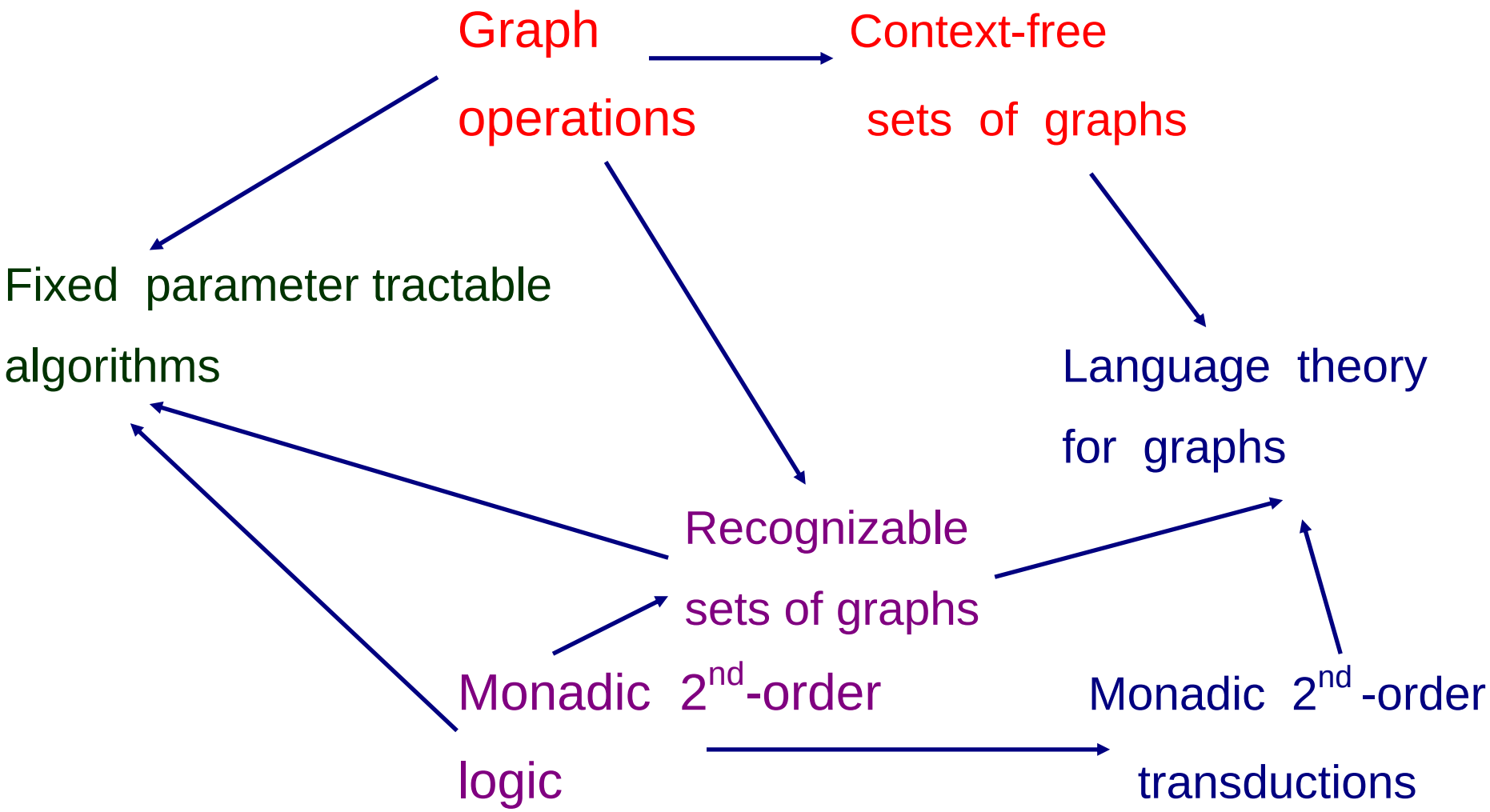
This computation takes time $O(|t|)$: the key to *fixed-parameter tractable* algorithms

An inductive set of properties can be effectively constructed (at least theoretically) from every monadic-second order formula.

Open Problem : How to make this technique usable ?

One idea is to design logical languages with “strong primitives” in order to express useful graph properties with few quantifications.

Conclusion


Graph operations
Context-free sets of graphs
Fixed parameter tractable algorithms
Language theory for graphs
Recognizable sets of graphs
Monadic 2nd-order logic
Monadic 2nd-order transductions