Context-Free Categorical Grammars

Michel Bauderon, Rui Chen, and Olivier Ly

Université de Bordeaux - LaBRI CNRS,
351 cours de la Libération Talence 33405 France
Zhongnan University of Economics and Law,
1 Nanhunan Dadao, Hongshan Wuhan 430073 China
{bauderon,ly}@labri.fr,
rui.chen.bordeaux1@gmail.com

Abstract. We define generic categorical notions of rewriting and grammar, using two basic operations, pullback and pushout, and show that these categorical grammars are intrinsically context-free in the sense of Courcelle. We then specialise to various settings, including classical word grammars, hyperedge replacement grammars or node-replacement grammars. We show that some languages which are classical counter-example to context-freeness become context-free within this new framework[1].

Keywords: Category, rewriting system, grammar, context-freeness.

1 Introduction

This works stems from research in the more specific area of graph rewriting where two main directions have been explored, which correspond to two distinct (somehow dual) approaches to the structure of a graph, either as nodes linked by arrows (vertex rewriting) or as arrows glued by nodes (edge or hyperedge rewriting).

In both directions, four levels of description have been explored : set theoretic, algebraic (namely using universal algebra), logical or categorical (using category theory as a basic tool). In this last setting - using category theory - the main effort has been devoted to edge (and hyperedge) replacement - using pushout as a basic operation to generalize the usual substitution, leading to the development of a large theoretical body, via the double and single pushout approach to graph rewriting (the so-called algebraic approach) and their extensions (the reader may refer to [9] for an extensive descriptions of formalisms and results).

In earlier works (such as [2,3,4,5]), we have shown how a dual approach - using pullback in place of pushout as the basic rewriting operation - could provide a sound categorical approach to node rewriting in graphs (and hypergraphs [4]). We have shown in [3] that pullback graph grammars are context-free.

In this paper, we generalize this approach by defining a generic categorical treatment of substitution, rewriting and grammars, abstracting as much as possible and lifting main notions and results to their proper level of abstraction. We

[1] This work was been completed while the first author was on a CNRS leave at LIAMA, Chinese Academy of Sciences, Institute of Automation, Beijing.

thus reach a necessary but sufficient level of abstraction to ensure the context-freeness property (in the sense of [6]) and it is our main result that categorical grammars[2] are intrinsically context-free. The main ideas of the proof remain those which have been presented in [3,5], although some useless conditions have been removed here.

This allows us to describe in this new setting several standard examples (such as words and graphs), showing that the mere reversing of arrows leads to very different situations and that the projective grammars are much more powerful than the inductive ones.

We show for instance in section 3, that inductive word grammars are exactly classical context free word grammars, while projective word grammars can generate some context-sensitive languages such as the well known $a^n b^n c^n$, which then becomes context-free in the categorical setting.

This work relies on elementary category theory whose basic definitions will be taken for granted (but the reader may refer e.g. to [1] available on line). Due to space limitations, proofs have been omitted.

2 Rewriting in a Category

There are (at least) two possible ways to rewrite objects in a category, by using the two simplest standard binary operations available in this framework, namely pullback and pushout (product and coproduct are much simpler, but much less flexible). Although pushout directly generalizes classical substitution, we follow the traditional approach where products, pullbacks and projective limits are put first, pushout and inductive limits being left to a duality argument ([1]).

In this paper, we shall not consider double-pushout or double-pullback rewritings, which introduce some pattern matching in the computation process. Our rules will always be directly applicable.

2.1 Basic Definitions

As usual, we let $\#S$ denote the cardinality of a set S and $\mathbb{N}$ be the set of non negative integers. In a category $\mathcal{C}$, a *span* (resp. a *cospan*) in $\mathcal{C}$ is a pair of arrows with same codomain (resp. domain). A family of arrows F has domain G (resp. codomain G) if the domain (resp. codomain) of each arrow of F is G. Let $G \xrightarrow{u} X \xleftarrow{p} H$ be a span. If it exists, let us denote by $G[p/u]$ the pullback object of this span and let $G \xleftarrow{a} G[p/u] \xrightarrow{b} H$ be the associated co-span. Let us note that it is symmetric: $G[p/u] = H[u/p]$. For any arrow f with domain G, we define a new arrow $f[p/u] = f \circ a$ with domain $G[p/u]$ and by extension, for any family F of arrows with domain G, we define $F[p/u] = \{f[p/u] \mid f \in F\}$.

Source. From now on, we shall consider a distinguished subset $\mathcal{N}$ of objects in $\mathcal{C}$ whose elements will be called *non-terminals*.

[2] We are aware that, by its simplicity, the expressions "categorical grammars" has been extensively used, e.g. in the area of computational linguistics, but we did not find any more accurate expression to designate our grammars.

Definition 1. *An $\mathcal{N}$-*source *(or simply a* source*) is a triple $\overline{G} = (G, U_G, P_G)$ consisting of an object G in $\mathcal{C}$ and two families U_G and P_G of arrows which share the same domain G, whose codomains are non-terminal objects and such that for any non-terminal X, there is at most one arrow in P_G with codomain X. Elements of U are called the* unknowns *occuring in $\overline{G}$. An element $p : G \to X$ of P_G is called the* replacement scheme *for the non-terminal X in $\overline{G}$. A source is said to be* terminal *if it has no unknown, i.e., if $\#U_G = 0$.*

Substitution. Substitutions operate on sources: substituting an unknown in a source by an other source will rise to a new source, in the following way:

Definition 2. *Let $\overline{G} = (G, U_G, P_G)$ and $\overline{H} = (H, U_H, P_H)$ be two sources. Let $u : G \to X$ be an unknown of $\overline{G}$. If $\overline{H}$ has a replacement scheme $p : H \to X$ for the non-terminal X, the* substitution *(or* replacement*) of u by $\overline{H}$ in $\overline{G}$ is the source, denoted by $\overline{G}[\overline{H}/u]$, and defined by the triple:*

$$(G[p/u],\ (U_G \backslash \{u\})[p/u] \cup U_H[u/p],\ P_G[p/u])$$

Let us note that the notation $\overline{G}[\overline{H}/u]$ is no longer symmetric.

Rewriting. We can now define a rewriting rule (together with the rewriting mechanism):

Definition 3. *A* rewriting rule *is a pair denoted by $X \to \overline{H}$ where X is a non-terminal object and $\overline{H}$ is a source provided with a replacement scheme for X. Applying a rule r to a source $\overline{G}$ consists in selecting an unknown $u : G \to X$ of $\overline{G}$ (if any) and substituting $\overline{H}$ to u, giving rise to $\overline{G}[\overline{H}/u]$. This will be denoted by $\overline{G} \overset{r,u}{\Longrightarrow} \overline{G}[\overline{H}/u]$, or simply $\overline{G} \overset{r}{\Longrightarrow} \overline{G}[\overline{H}/u]$.*

As usual, one-step derivation $\overset{r}{\Rightarrow}$ defines a binary relation on sources whose reflexive and transitive closure is denoted by $\overset{*}{\Rightarrow}$.

2.2 Dual Approach

As already mentionned, similar definitions can of course be given in a dual way, by defining a *sink* (or co-source) with families of morphisms of the form $((f_i : X_i \to G)_{i \in I})$. G is the codomain of the sink and its domains are in $\mathcal{N}$. The definition of substitution will be dual, making use of pushout instead of pullback as a basic operation to produce a sink out of two sinks.

In the sequel, we shall try to simplify by using the expressions *production rules* (resp. *occurrence*) to designate both sources or sinks (resp. both sources or sinks with $\#P_G = 0$).

2.3 Rewriting Structures

Definition 4. *A* categorical rewriting system *in a category $\mathcal{C}$, over a family of objects $\mathcal{N}$ called non-terminals is given by a family of productions. The system is* admissible *when all productions may be applied at any stage. A* categorical grammar *will be given by a categorical rewriting system and a specific occurrence called the* axiom.

We shall in the sequel use the prefix *projective* (resp. *inductive*) to denote systems relying on pullback (resp. pushout) as their rewriting mechanism.

If the category is complete (projective systems) or co-complete (inductive systems), any categorical rewriting system is admissible. If not, we shall have to give conditions for such a system to be admissible.

The *language* generated by the grammar is the family of terminal occurences derived through the relation $\overset{*}{\Rightarrow}$. The *extended language* is the family of not necessarily terminal occurrences derived through the relation $\overset{*}{\Rightarrow}$.

2.4 Context-Freeness

We shall follow Courcelle [6], and use his axiomatic definition of the notion of context-freeness as the conjunction of three properties: preservation, confluence and associativity. Let us first describe categorical (co)rewriting system as a *substitution system* in the sense of [6].

The *alphabet* is simply $\mathcal{N}$. To completely fall within the framework described by [6], we need an indexing of the non-terminal objects as $\mathcal{N} = \{X_i / i \in [1, n]\}$.

The *objects* are the sources over $\mathcal{N}$, whose set is denoted by $\mathcal{C}_{\mathcal{N}}$. If $\overline{G} = (G, U_G, P_G)$ is such an object, the *arity function* α sends it to the ordered list $\alpha(\overline{G}) = (X_1 X_2 \ldots X_m)$ consisting of the elements of the (co)domain of $U_G = \{(u_j : G \to X_j)_{j \in [1,m]}\}$.

According to definition 2, the *substitution* operator [ ] defines a partial mapping from $\mathcal{C}_{\mathcal{N}} \times \mathcal{N} \times \mathcal{C}_{\mathcal{N}}$ to $\mathcal{C}_{\mathcal{N}}$ by $(\overline{G}, X, \overline{R}) \to \overline{G}[\overline{R}/X]$ whenever this makes sense. Using the previous indexing, it may also be described as a partial mapping $\mathcal{C}_{\mathcal{N}} \times \mathbb{N} \times \mathcal{C}_{\mathcal{N}}$ to $\mathcal{C}_{\mathcal{N}}$ defined by $(\overline{G}, i, \overline{R}) \to \overline{G}[\overline{R}/X_i]$.

Preservation. The first condition for context-freeness is to satisfy the *preservation axiom*: for all $(\overline{G}, i, \overline{R})$ in the domain of [ ], one must have

$$\alpha(\overline{G}[\overline{R}/X_i]) = X_1 X_2 \ldots X_{i-1} \alpha(\overline{R}) X_{i+1} \ldots X_n$$

where $X_1 X_2 \ldots X_n = \alpha(\overline{G})$, for $X_1, \ldots, X_n \in X$. It follows from definition 2 that:

Proposition 1. *The substitution mapping satisfies the preservation axiom.*

Hence, $\mathbf{G} = \langle \mathcal{C}_{\mathcal{N}}, \mathcal{N}, \alpha, [\,] \rangle$ is a *substitution system* in the sense of [6]. We may now study its properties.

Associativity. The first property expresses the fact that two consecutive steps of rewriting can be condensed into one.

Proposition 2. $\overline{G}[\overline{R}/X][\overline{S}/(Y[\overline{R}/X])]$ *and* $\overline{G}[\overline{R}[\overline{S}/Y]/(X[\overline{S}/X])]$ *are isomorphic (whenever they can both be computed), hence categorical rewriting systems are* associative.

Confluence. This second property expresses the commutativity of the substitution operation, or the fact that rewriting steps can be applied in any order, giving the same result.

Proposition 3. *$\overline{G}[\overline{R}/X][\overline{S}/Y]$ and $\overline{G}[\overline{S}/Y][\overline{R}/X]$ are isomorphic whenever they can both be computed hence categorical rewriting systems are* confluent.

These results show that categorical rewriting systems satify Courcelle's conditions for context-freeness ([6]) hence that :

Theorem 1. *Categorical rewriting systems are context-free.*

In the sequel, we shall use the expression *context-free* (abbreviated as CF) to denote the notion of context-freeness in the sense of [6] that we have briefly recalled in this section. When comparing with other definitions, we shall use a prefix such as w-CF to denote the standard definition of context-freeness in the classical theory of word languages.

3 Word Grammars

To describe word (or tree) languages in this setting, we need to put them in a categorical framework which is as close as possible to the usual definitions where for instance words are mappings from $[1, n] \subset \mathbb{N}$ to an alphabet A.

We shall consider as a base category the category **Pos** of partially ordered sets (with order-preserving mappings) which is well known to be both complete and cocomplete (see for instance [1]) and more precisely, categories $\mathbf{Pos}_A$ of posets labeled over an alphabet $A = \{a, b, c, \ldots\}$.

To model words or trees, we have to consider subcategories of **Pos** which are neither complete nor cocomplete, meaning that the pushout and pullback objects (which always exists in the enclosing category) may fail to be elements of the category of interest, hence that the rewriting systems mail fail to be admissible. Ensuring that they are will put stringent conditions of the type of rewriting systems that we can build. In each case we must identify conditions under which a pushout or a pullback object belongs to this subcategory as well as coherence conditions to be fulfilled by the labelings. Due to space limitations, we shall deal here only with words.

3.1 Words

To model words, we shall consider the subcategory $A^* = \mathbf{Tos}_A$ of totally ordered finite sets labeled over A.

Let us first give a few definitions. If $m \in A^*$, with $\#m = n$, we shall write $m = m^1m^2m^3 \ldots m^n$ (subscripts will be used to denote elements in a family of words). If its order relation is denoted by $\leq$, we shall say that two elements of $m^1 \leq m^2$ in m are *adjacent* if there is no m^3 such that $m^1 \leq m^3 \leq m^2$; m^2 is then the *successor* of m^1, while m^1 is the *predecessor* of m^2.

Totally Ordered Sets. The following two lemmata give a characterization of those pushouts and pullbacks which build total orders out of total orders.

Starting with the more intuitive case, let us consider the following pushout diagram in **Pos**, where x, u and v are total orders:

$$\begin{tikzcd} x \arrow[r, "xv"] \arrow[d, "xu"] & v \arrow[d, "vm"] \\ u \arrow[r, "um"] & m \end{tikzcd}$$

The pushout object m is easily constructed: its carrier is the pushout object in **Set** (the category of sets) while the order relation is induced by those on u and v : $m^1 \leq m^2$ if and only if $v^1 \leq v^2$ or $u^1 \leq u^2$, where $m^i = um(u^i)$ and/or $m^i = vm(v^i)$, $i = 1, 2$.

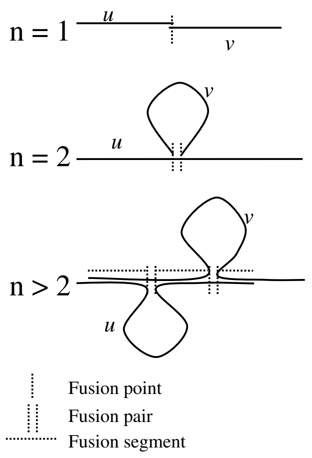

Fig. 1. Pushouts of total orders

Intuitively, if we represent u and v by linear segments, m is obtained by gluing these two segments at some fusion points (or along fusion segments) which are images of elements of x in u and v. The main possibilities to build total orders out of total orders are described in figure 1.

Lemma 1. *Let $n = \#x$.*

1. *if $n = 1$, m is a total order if and only if $xv(x)$ is the maximal point of v and $xu(x)$ is the minimal point of u (or conversely).*
2. *if $n = 2$, m is a total order if and only if $xv(x^1)$ and $xv(x^2)$ are adjacent in v, while $xu(x^1)$ and $xu(x^2)$ are the extremal points of u (or conversely)*
3. *if $n > 2$, m is a total order if and only if the three following conditions hold*
   (a) *either $xv(x^1)$ is the minimum of v or $xu(x^1)$ is the minimum of u,*
   (b) *either $xv(x^n)$ is the maximum of v or $xu(x^n)$ is the maximum of u,*
   (c) *if x^i and x^j are adjacent in x, then at least one of the pairs $xv(x^i)$ and $xv(x^j)$ or $xu(x^i)$ and $xu(x^j)$ is adjacent (in v or u).*

Remark 1. The first two cases are easily interpreted. The case where $n = 1$ clearly corresponds to the concatenation vu (or conversely uv) and could therefore be used to model *regular word grammars* (which we will not do), while the case $n = 2$ can be seen as the substitution of x by u in v (or conversely x by v in u) and will be used later to model *context-free word grammars.* The third case identifies and mixes parts of the two words according to the definition of the mappings o_x and r_x.

Let us now consider the pullback case, by considering the following pullback diagram in **Pos**:

$$\begin{tikzcd} m \arrow[r, "mv"] \arrow[d, "mu"] & v \arrow[d, "vx"] \\ u \arrow[r, "ux"] & x \end{tikzcd}$$

The carrier of the pullback object m is computed as the pullback object in **Set**, ie the set of those elements (u_1, v_1) in the cartesian product $u \times v$ such that $ux(u_1) = vx(v_1)$. The order on m is the order induced by the product order, namely $(u_1, v_1) \leq (u_2, v_2)$ if and only if $u_1 \leq u_2$ and $v_1 \leq v_2$.

Lemma 2. *The pullback object is a total order if and only if for each x_i in x, either $\#ux^{-1} \leq 1$ or $\#vx^{-1} \leq 1$.*

Words. After describing conditions for pushout and pullback of total orders to be total orders, we must describe the action of these operations on the labeling which we need in order to turn total orders into words.

Let $A = \{a, b, c, \ldots\} \cup \{\odot, \top, \bot\}$ be a set of terminal letters with three special symbols $\odot$, $\top$ and $\bot$. We shall let the alphabet A be partially ordered by $\bot, \top \leq a \leq \odot, \forall a \in A$ (letters in A are not comparable, $\bot$ and $\top$ need not be either).

In $\mathbf{Tos}_A$, an object m of length n (a word in A^*) is a mapping $[1, n] \xrightarrow{m} A$, that we may write in the form $\{m_1 \leq m_2 \leq \ldots \leq m_n\}$ where $m_i \in A$.

Since the set A of labels (terminal letters) is ordered, we can set :

Definition 5. *A morphims of words $f : m \to m'$ is an order-preserving mapping (i.e. an arrow in **Tos**) such that $m_i \leq m'_{f(i)}$.*

One may check that the composition of two such morphism is still a morphism hence that $A^* = \mathbf{Tos}_A$ is a category, and that if a diagram is a pushout (resp. a pullback) in **Tos**, then it is a pushout (resp. a pullback) in $\mathbf{Tos}_A$. It is enough for that to set that whenever an element m_i in m has preimages (resp. is image of) both u_i and v_i in u and v, the label of m_i will be the maximum (resp. the minimum) of the labels of u_i and v_i. It will be shown in the two following sections that whenever the result of the computation is in **Tos**, then these labels are comparable.

3.2 Inductive Grammar

It follows from lemma 1, that the usual context-free substitution of a letter by a word can be modelled in the category $A^* = \mathbf{Tos}_A$.

A non terminal letter x will be modeled by a 2-elements total order $x = \{x^1 \leq x^2\}$ (we shall use the same symbol to denote the letter and the order), which we shall label $\bot\top$. An occurence of x in a word w is an arrow $x \overset{o_x}{\to} w$, such that $o_x(x^1)$ and $o_x(x^2)$ are adjacent. This means that w is of the form $w^1 \ldots w^k o_x(x^1) o_x(x^2) w^{k+2} \ldots w^p$. We shall write as usual $w = txu$, with $t = w^1 \ldots w^k$ and $u = w^{k+2} \ldots w^p$.

Let us be given a context-free word rewriting rule of the form $x \to m$, where $m \in A^*$ is a word of length n and some other non-terminals y_i occuring in m.

This rule may be modelled by a sink $((y_i \overset{o_{y_i}}{\to} m), x \overset{r_x}{\to} m)$ where $r_x(x^1) = m^1, r_x(x^2) = m^n, \#m = n$, and for each i, $y_i \overset{o_{y_i}}{\to} m$ is an occurence[3] of y_i in m.

The substitution is given by computing the pushout corresponding to the following diagram (which is well defined in $\mathbf{Pos}_A$):

$$\begin{tikzcd}
x \arrow[r, "r_x"] \arrow[d, "o_x"] & m \arrow[d, "\alpha"] \\
txu \arrow[r, "\beta"] & tmu
\end{tikzcd}$$

From Lemma 1, it follows that tmu is a total order. The labelling on tmu is uniquely defined everywhere except on the images of x^1 and x^2 where there are two possibilities, one coming from x in txu, the other from m. Since x is labelled by $\bot\top$, the labelling on m is well defined in each case as the maximum of the two possible labels.

Each arrow $(y_i \overset{r_{y_i}}{\to} m)$ defines by composition with α an occurence $(y_i \overset{\alpha \circ r_{y_i}}{\to} tmu)$ and the computation can be continued by further application of rules of the form $(y_i \to m_i)$.

This construction shows that inductive rewriting models the substitution mechanism used in the standard theory of words languages.

The details of the encoding are omitted due to the lack of space.

Conversely, it may be shown that any pushout rule can be interpreted as a classical context-free rewriting rule, hence:

Lemma 3. *Context-free word languages are exactly inductive word languages, which we summarize as w-CF = po.*

3.3 Projective Grammar

We shall now make use of the symbol $\odot$ which we added to A to label a single element order which will thus be turned into a terminal object in the category A^* and a neutral element for the categorical product. For the sake of clarity in the following diagrams defining morphisms, we shall also use the letters $x, y, z, \ldots$ to denote the same neutral element $\odot$.

[3] Of course m must be of the form $m = a_1 y_1 \ldots a_p y_p a_{p+1}$ for some (possibly empty) words $a_1, \ldots a_{p+1}$, meaning that the images $o_{y_i}(y_i)$ in m must not overlap. This corresponds to the fact that there can not be two distinct letters at the same place in a word and is necessary to ensure that all computed pushout objects are actually words.

Let us first note that the following diagram, where the definition of the arrows r_x and o_x should be clear from the drawing (modulo the use of the letter x to denote an occurence of $\odot$), models in a projective way the application of the rule $x \to m$ to the word axb (the relabelling of the pullback object is being defined as earlier):

$$\begin{tikzcd} amb \arrow[r] \arrow[d] & axb \arrow[d, "r_x"] \\ \odot m \odot \arrow[r, "o_x"] & \odot x \odot \end{tikzcd}$$

Hence the following lemma:

Lemma 4. *Every word context-free rule can be encoded as a projective rule, hence: w-CF = po ⊂ pb = CF*

Let us consider a slightly more complex rule, by considering the production defined by the pair of arrows

$$(\odot ax \odot by \odot cz \overset{o_y}{\to} \odot x \odot y \odot z, \odot ax \odot by \odot cz \overset{r_x}{\to} \odot x \odot y \odot z)$$

where $r_x(\odot) = \odot$, $r_x(ax) = x$ and so on (o_y being defined in the same way). Let o_x be the occurence $axbycz \to \odot x \odot y \odot z$.

Then the following pullback diagram :

$$\begin{tikzcd} aaxbbyccz \arrow[r, "\alpha"] \arrow[d] & \odot ax \odot by \odot cz \arrow[d, "r_x"] \\ axbycz \arrow[r, "o_x"] & \odot x \odot y \odot z \end{tikzcd}$$

computes a new word $aaxbbyccz$ and generates a new occurence $aaxbbyccz \xrightarrow{o_y \circ \alpha} \odot x \odot y \odot z$ where the production rule may be applied once more.

Lemma 5. *This rewriting system (together with an unknown erasing rule) generates* in a context free way [4] *the language $a^n b^n c^n$.*

This shows that:

Theorem 2. *w-CF= po $\subsetneq$ CF = pb $\subsetneq$ CS*

The last inequality is quite clear: projective grammars can not generate all context-sensitive grammars (CS): the rewriting mechanism does not provide any sort of pattern matching as needed for the most general CS grammars, since all new occurences are built by composition of functions and can not appear through mere juxtaposition of letters.

The question remains of the exact expressive power of projective word grammars, although a careful examination of the possible codomains for productions suggests that they can not generate anything really more complex than $a^n b^n c^n$.

[4] It is not new that the language $a^n b^n c^n$ can be generated in a context-free way, but this needs an encoding of words as string graphs (see [8]) hence needs going out of word-rewriting to use techniques from hyperedge replacement grammars within the category of hypergraphs.

4 Graphs and Hypergraphs

It is well known that categories of graphs or hypergraphs are both complete and cocomplete. Pushout and pullback can always be computed, and we therefore simply need to interpret the nature of inductive and projective graph grammars.

4.1 Inductive Hypergraph Grammars

Let $\mathcal{H}$ be the category of hypergraphs and $\mathcal{X}$ be a set of non-terminal hypergraphs. Inductive hypergraph grammars can be defined with no restrictions along the lines of section 2.3.

Theorem 3. *Inductive hypergraph grammars are exactly hyperedge replacement grammars in the sense of [8].*

4.2 Projective Graph Grammars

Projective graph grammars have not so far been studied in general.

We shall recall here the basic definitions of *pullback graph grammars*, which, while being a very restricted case, are sufficient to describe e.g. node replacement systems and to provide already intersting results (details and proofs may be found in [2,3,4,5]).

Definition 6. *A graph G is a* 4*-tuple $G = \langle V_G, E_G, s_G, t_G\rangle$ where the sets V_G of vertices and E_G of edges are two finite disjoint sets and s_G and t_G are mappings from E_G to V_G. For every element $e \in E_G$, $s_G(e)$ and $t_G(e)$ are called source vertex and target vertex of the edge e respectively.*

A vertex $v \in V_G$ is *reflexive* if there exists an edge $e \in E_G$ such that $s_G(e) = t_G(e) = v$. A graph G is reflexive if all its vertices are reflexive, it is said to be simple if for any pair x, y of vertices of G, there is at most one edge from x to y.

Non Terminals. Non-terminals graphs have a quite specific form which allows them to distinguish between nodes to be transformed, nodes to be identically reproduced and an intermediate zone.

Definition 7. *A* non-terminal graph *X is a graph made out of two components: a complete reflexive graph K_{m+1}, and a reflexive subgraph U linked to only m of the vertices of K_{m+1}.*

Occurrences and Productions. For the sake of simplicity (and to keep some coherence with e.g. [3]), we shall simply describe the shapes of the morphisms involved in the definition of a source $\overline{G} = (G, U_G, P_G)$, calling occurence any morphism appearing in U_G and production the morphism involved as a replacement scheme in P_G. They both will have a very special structure.

Definition 8. *Let G be a directed simple graph and X a non terminal graph, an* occurrence *x on G is a graph morphism from G to X such that the pre-image $x^{-1}(U)$ is non empty.*

Definition 9. *A* production *r is a morphism $r : R \to X$ which is isomorphic on the inverse image of the subgraph of X generated by K_{m+1}.*

Theorem 4. *Pullback graph grammar are projective graph grammars hence are context-free. For every inductive hypergraph grammar, there is an equivalent pullback graph grammar.*

The converse of the second assertion is false, since hypergraph replacement grammars cannot generate square grids ([8]), which can be generated by a pullback graph grammar with only one rule (as shown in [3]).

In [6], Courcelle shows that vertex replacement grammars (VR-grammars) are context-free. The result from [3] shows that although it satisfies the same definition of context-freeness, pullback rewriting provides us with a strictly more powerful context-free mechanism.

The real expressive power of general projective graph grammars remain to be investigated.

5 Conclusion

In this paper, we have set a generic categorical framework for rewriting, with two distinct possibilities, inductive rewriting based on the pushout operation and projective rewriting based on pullback. We have then shown that both types of categorical rewriting, either inductive or projective, are intrinsically context-free (after the well accepted definition of [6]).

This general framework has then been instantiated to two specific cases. First of all, we have shown that inductive rewriting on words mimics the classical theory of context-free word languages (generating exactly the same languages), while projective rewriting gives a more powerful notion of context-freeness, where languages such that $a^n b^n c^n$ become context-free.

In a similar way, inductive rewriting of hypergraphs describes hyperedge replacement grammars (well known to be context-free), while projective graph rewriting yields a much more powerful context-free mechanism, in which graphs languages such as the language of all complete graphs or that of square grids become context-free (as already noticed, we actually used only a very specific case of projective graph rewriting by putting strong conditions on the rules and occurences).

While we have shown that inductive rewriting describes (at least in two cases) "classical" context-free rewriting, the exact power of projective rewriting, both in the case of words or graphs remains an open question. It is also open whether projective rewriting is *always* strictly more powerfull than inductive rewriting, as is the case for words or graphs.

References

1. Adamek, J., Herrlich, H., Strecker, G.E.: Abstract and Concrete Categories, `http://katmat.math.uni-bremen.de/acc/acc.pdf`
2. Bauderon, M.: A uniform approach to graph rewriting: the pullback approach. In: Nagl, M. (ed.) WG 1995. LNCS, vol. 1017, pp. 101–115. Springer, Heidelberg (1995)

3. Bauderon, M., Chen, R., Ly, O.: Pullback Grammars Are Context-Free. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 366–378. Springer, Heidelberg (2008)
4. Bauderon, M., Jacquet, H.: Node rewriting in graphs and hypergraphs: a categorical framework. Theoretical Computer Science 266(1-2), 463–487 (2001)
5. Chen, R.: Graph Transformation and Graph Grammar Based on Pullback Operation, PhD thesis, Université Bordeaux 1 (2007)
6. Courcelle, B.: An axiomatic approach to context-free rewriting and its application to NLC graph grammars. Theoretical Computer Science 55, 141–181 (1987)
7. Engelfriet, J., Rozenberg, G.: Node Replacement Graph Grammars. In: [9], pp. 1–94
8. Habel, A.: Hyperedge Replacement: Grammars and Languages. LNCS, vol. 643. Springer, Heidelberg (1992)
9. Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific Publishing, Singapore (1997)