Electronic Communications of the EASST
Volume 61 (2013)

Selected Revised Papers from the 4th International Workshop on Graph Computation Models (GCM 2012)

Derivation Languages of Graph Grammars

Nils Erik Flick

22 pages

Guest Editors: Rachid Echahed, Annegret Habel, Mohamed Mosbah
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Derivation Languages of Graph Grammars

Nils Erik Flick[1*]

[1] flick@informatik.uni-oldenburg.de
Carl von Ossietzky Universität, D-26111 Oldenburg

Abstract: We investigate sequential derivation languages associated with graph grammars, as a loose generalisation of free-labeled Petri nets and Szilard languages. The grammars are used to output strings of rule labels, and the applicability of a special rule determines the acceptance of a preceding derivation.

Due to the great power of such grammars, this family of languages is quite large and endowed with many closure properties. All derivation languages are decidable in nondeterministic polynomial time and space $O(n \log n)$, by simulation of the graph grammar on a Turing machine.

Keywords: derivation language, closure properties, graph grammars, double pushout

1 Introduction

Graph grammars are well known and well studied [Roz97]. This paper deals with their derivation languages: though a graph grammar is usually employed to generate graphs, we use it to output strings in an automaton-like fashion. The graph is thus the state of a nondeterministic machine and the labels of the rules used in the derivation are output symbols.

The original motivation for this work is the extension of notions already known in some restricted contexts. Petri nets are graph grammars with no edges [Cor95, KKK06], and their languages of labeled transition sequences are well known [Jan87]. Graph grammars can also simulate string grammars, and results about their derivation languages exist [Mäk98]. However, any description of the more derivation languages of graph grammars appears to be missing.

This, by itself, would perhaps not be motivation enough to consider such a vast generalisation; yet we have found the subject to be intricate enough to justify further investigation. The language class, we have named $\mathcal{L}_\$$, studied in this paper has a relatively concise description in terms of graph grammars, and it seems to be a fairly natural class with strong closure properties. Some new techniques and non-trivial results arise from the definitions.

Recasting other known formalisms as special cases of graph grammars after forging some generic tools for understanding their languages can also be a motivation to deal with graph grammars; Petri nets are sometimes too limiting from the modeling point of view. The dynamics of many kinds of systems, for instance modeled as high level Petri nets, can be understood in terms of local operations on graph-like system state. Edges are necessary, for instance, to ensure the coherent transport of a set of tokens which logically belong to a single modeled entity. We argue

* This work was supported by a University of Hamburg (HmbNFG) doctoral fellowship. This work is supported by the German Research Foundation (DFG), grant GRK 1765 (Research Training Group – System Correctness under Adverse Conditions).

that understanding their languages, as a simple form of semantics, can lead to steps towards analysis, verification and possibly synthesis from behavioural specifications, of complex systems.

The paper is structured as follows. Section 2 reviews the basic definitions and introduces derivation languages. Section 3 presents closure properties, constructively obtained via operations on graphs and grammars. We present a proof of $\mathcal{L}_\$$ under letter-to-letter homomorphisms, expressing (ambiguous) rule relabelings. We use the closure properties to establish a first hierarchy result. In Section 4 we present an upper bound on the nondeterministic complexity of the word problem for a derivation language. We show that a lot of questions about general derivation languages are undecidable, and application of deleting homomorphisms leads to the recursively enumerable languages. Section 5 concludes with an overview of our results and an outlook.

2 Graph Grammars

In this section, we introduce the notions of graphs, rules, grammars and derivation languages.

We write $-$ for set difference, $X - Y = \{x \in X \mid x \notin Y\}$, and $f|_Y$ for the restriction of the function $f : X \to Z$ to $Y \subseteq X$. $\circ$ is function or relation composition. $X + Y$ denotes the union of the sets X and Y when these are assumed to be disjoint. If $A \subseteq X \cap Y$, $X +_A Y$ also denotes a union of X and Y, with $X - A$ and $Y - A$ assumed disjoint.

Assumption 1 *Let Λ be a label alphabet.*

Definition 1 (Graphs and Morphisms) A *graph* is a tuple $G = (V_G, E_G, s_G, t_G, \lambda_{GV}, \lambda_{GE})$ of two disjoint finite sets V_G of nodes, E_G of edges and four total functions, $s_G, t_G : E \to V$, $\lambda_{GV} : V \to \Lambda$ $\lambda_{GE} : E \to \Lambda$ called the source and target mappings and the label functions, respectively.

A *morphism* $f : G \to H$ from a graph G to a graph H is a pair $f = (f_V, f_E)$ of total functions $f_V : V_G \to V_H$ and $f_E : E_G \to E_H$ which preserves the connectivity and the labels: $s_H \circ f_E = f_V \circ s_G$, $t_H \circ f_E = f_V \circ t_G$, $\lambda_{HE} \circ f_E = \lambda_{GE}$ and $\lambda_{HV} \circ f_V = \lambda_{GV}$. f is called *injective* or an *inclusion* (usually denoted by $f : G \hookrightarrow H$) iff both f_V and f_E are injective. An *isomorphism* is a morphism with bijective components. Being related by isomorphisms is an equivalence relation denoted $\cong$.

Nodes and edges are collectively called *items*. When the meaning can be inferred from the context, we will denote both components f_V and f_E of a morphism f just by the symbol f. We write λ_G for both λ_{GV} and λ_{GE} when it is convenient to do so. Graphs is the set of all graphs.

Definition 2 (Rule) A *rule* is a tuple $\varrho = (L \hookleftarrow K \hookrightarrow R)$ of graphs, where K is called the *interface* and is contained both in the *left hand side* graph L and in the *right hand side* graph R. A rule ϱ is called *identical* and denoted by id_K if $L = K = R$.

Definition 3 (Direct Derivation Step) Let ϱ be a rule, G a graph and $g : L \hookrightarrow G$ an injective morphism. The *dangling condition* requires that for every node $n \in V_G$ in the range of g_{GV} that is *not* the image of a node in K, every edge $e \in E_G$ with $s_G(e) = n$ or $t_G(e) = n$ is in the range of g_{GE}. If g satisfies the dangling condition, then g is called a *match* of L in G. A *direct derivation step* $G \overset{\varrho,g}{\Rightarrow} H$ from a graph G to a graph H with a rule ϱ via a match g consists of a *double pushout*

diagram

$$\begin{array}{ccccc} L & \hookleftarrow & K & \hookrightarrow & R \\ {\scriptstyle g}\downarrow & (1) & \downarrow & (2) & \downarrow{\scriptstyle h} \\ G & \hookleftarrow & D & \hookrightarrow & H \end{array}$$

where

$$(1)\ D = G - g(L-K), \qquad (2)\ H = D + h(R-K).$$

The equation (1) stands for $V_D = V_G - g_V(V_L - V_K)$ and $E_D = E_G - g_E(E_L - E_K)$, (2) means $V_H = V_D + h_V(V_R - K_R)$ and $E_H = E_D + h_E(E_R - K_R)$.

We may omit the match in the notation: $G \overset{\varrho}{\Rightarrow} H$. Given a set $\mathcal{R}$ of rules, a *derivation* $G_0 \overset{v}{\underset{\mathcal{R}}{\Rightarrow}}{}^* G_n$ is a sequence of derivation steps $G_0 \overset{\varrho_1}{\Longrightarrow} \ldots \overset{\varrho_n}{\Longrightarrow} G_n$ with $v = \varrho_1 \ldots \varrho_n$.

Definition 4 (Graph Grammars and Derivation Languages) A *graph grammar*, short *grammar*, is a tuple $GG = (T, \mathcal{R}, \mathfrak{r}, S)$ of an alphabet T of (rule) labels, a finite set of rules $\mathcal{R}$, a surjective function $\mathfrak{r} : T \to \mathcal{R}$ and a start graph S. We write $G \overset{w}{\underset{GG}{\Rightarrow}}{}^* H$ for $G \overset{\mathfrak{r}(w)}{\underset{\mathcal{R}}{\Rightarrow}}{}^* H$, where $w = w_1 \ldots w_n$ and $\mathfrak{r} : T^* \to \mathcal{R}^*$ denotes the extension of $\mathfrak{r} : T \to R$ with $\mathfrak{r}(w_1 \ldots w_n) = \mathfrak{r}(w_1) \ldots \mathfrak{r}(w_n)$ for $w = w_1 \ldots w_n \in T^*$. The subscript may be omitted when GG can be inferred from the context. If $G \overset{w}{\Rightarrow}{}^* H$ and $H \overset{\mathfrak{r}(t)}{\Rightarrow} J$, we say that w *may enable* t from G. We will also use the terms *rule application* and *applicability of a rule* for a derivation step and the existence of a derivation step, respectively.

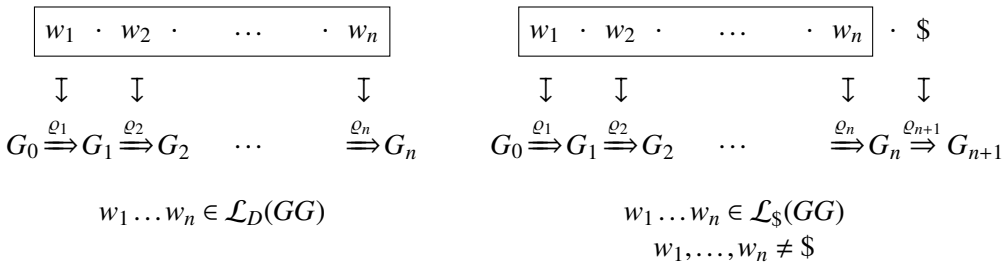
The prefix-closed *derivation language* of GG is the set

$$\mathcal{L}_D(GG) := \{w \in T^* \mid S \overset{w}{\underset{GG}{\Rightarrow}}{}^*\}$$

of all sequences w of rule labels such that a derivation starting from S with the corresponding rules exists. The $\$$-*terminated derivation language* of GG and $\$ \in T$ is the set of all strings over $T - \{\$\}$ in $\mathcal{L}_D(GG)$ which may enable $\$$:

$$\mathcal{L}_\$(GG, \$) := \{\, w \in (T - \{\$\})^* \mid w\$ \in \mathcal{L}_D(GG) \,\}$$

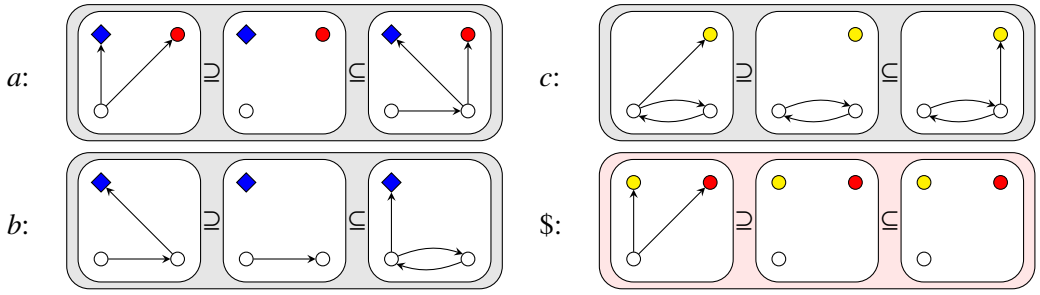
Unless otherwise stated, the terminating character is $\$$, and we simply write $\mathcal{L}_\$(GG)$.
$\mathcal{L}_D$ denotes the class of all languages $\mathcal{L}_D(GG)$ for some grammar GG, $\mathcal{L}_\$$ analogously.



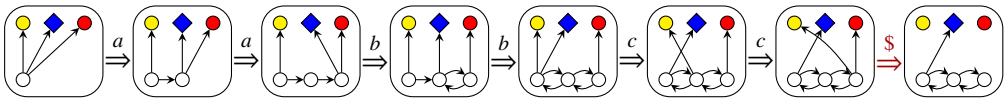
For every derivation sequence there exist in general one or several rule label sequences. For every rule label sequence in T^* there exist in general zero, one or several derivations.

Let us fix conventions for the graphical representation of graphs, rules and grammars. Node colours and shapes shall represent labels. Node inscriptions represent node names or labels, according to the legend given in the text. Edges are also decorated (dashed or thick lines) according to their label. Rules are represented with the left hand side, interface, right hand side in that order from left to right; the common items are placed at the same location relative to the bounding box in all three graphs of the rule. A special representation is introduced later to abstractly represent subgraphs. In two instances, a subgraph will be highlighted by rendering nodes and edges in bold and in bright colours.

Example 1 Let the rules $\mathcal{R}$ of a grammar GG be



Then there is a derivation with the given start graph,



and the word *aabbcc* is a word of $\mathcal{L}_\$(GG)$.

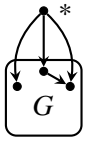
3 Closure Properties and Relation to Other Families

This section presents several features of $\mathcal{L}_\$$ in two subsections. In Subsection 3.1, we constructively prove that $\mathcal{L}_\$$ is closed under homomorphisms which assign only single letters (several letters may have the same image). For the purpose of language generation, it is possible to simulate such a non-injective relabeling of the rules with a modified grammar. The modification of the start graph and rules to this effect will be explained in detail and proved correct. The resulting theorem is a property of $\mathcal{L}_\$$ not shared by the similar but less general mechanisms that had so far been investigated in the literature. It has non-trivial applications to the further development of the derivation language theory of graph grammars: in Subsection 3.2, we use it as an ingredient for the proof of the inclusion of the context free languages CF in $\mathcal{L}_\$$. From that proof the stronger statement of the proper inclusion $CF \subset \mathcal{L}_\$$ already follows, because $\mathcal{L}_\$$ is closed under intersections and CF is not. We then demonstrate that all Szilard languages are in $\mathcal{L}_\$$, and finally provide another valuable example of a non-context-free language in $\mathcal{L}_\$$.

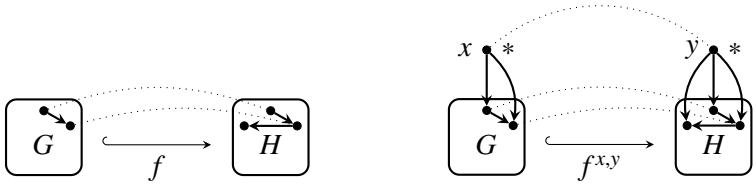
3.1 Closure under Rule Relabelings

We start with a construction that serves to extend a graph with an extra node, which is connected to all nodes of the original graph. The construction is then extended to rules. Because the extra node is given a special label, it cannot be confused with any other node, and the matches of the modified rule in the modified graph will correspond exactly to the matches of the original rule in the original graph. This construction, which we call *suspension*, may appear almost trivial but is essential in the sequel: the extra node can serve as a single gluing point to attach the whole graph to a specific place in a much larger graph. This, in turn, is essential for our approach to constructing a grammar that runs a simulation of another grammar such that the language of the simulating grammar is a homomorphic image of the language of the simulated grammar.

Definition 5 (Suspension) Given a graph G and a new label $*$ not in the range of λ_{GV} nor λ_{GE}, the *suspended* graph G^x is obtained from G by adding a new node x with label $*$, and edges from x to each node of V, also with label $*$. We will usually represent G graphically as a box,



Given a morphism $f : G \to H$, $f^{x,y}$ denotes the morphism from G^x to H^y, such that $f^{x,y}(u) = f(u)$ if u is an item of G, $f^{x,y}(x) = y$ and each new edge with target in G^x is mapped to the corresponding new edge in H^y.



We fix a label $*$ to be implicitly used as *the* new label for all suspensions in the rest of the paper, and assume it does not occur in any graph subjected to the construction.

Let ϱ^x denote the rule obtained by suspending the graphs of ϱ on x. Given a grammar GG, GG^x denotes the grammar obtained by applying suspension to the start graph and the interfaces, left hand and right hand sides of the rules of GG.

Lemma 1 (Suspension Lemma) *For every match $g : L \to G$ of a rule ϱ in G, there is a match $g^{x,y} : L^x \to L^y$ of ϱ^y in G^y such that $G^y \Rightarrow^{\varrho^x, g^{x,y}} H^y$ if $G \Rightarrow^{\varrho,g} H$.*

Every match $q : L^x \to G^y$ of a rule ϱ^x in G^y is of the form $q = g^{x,y}$ for some morphism $g : L \to G$, and g is a match of ϱ such that $G \Rightarrow^{\varrho,g} H$ if $G^y \Rightarrow^{\varrho^x, g^{x,y}} H^y$.

Proof. $G \overset{\varrho,g}{\Rightarrow} H$ implies $G^y \overset{\varrho^x, g^{x,y}}{\Longrightarrow} H^y$: Extend the match g, and h, to x to $g^{x,y}$ resp. $h^{x,y}$. This results in a derivation because $g^{x,y}$ is again a match: the dangling condition is satisfied because nodes n to be deleted have a supplementary edge attached to them but that edge from G^y to n is also present in the modified left hand side graph of the rule ρ^x.

$G^y \overset{\rho^x, g^{x,y}}{\Longrightarrow} H^y$ implies $G \overset{\rho,g}{\Rightarrow} H$: Since there is exactly one node labeled $*$ in G^y and one edge from it to every node from G if there are any nodes in G, the images of the $*$ labeled node and edges of D in H are determined by the source and target consistency requirements for a morphism. The removal of all such $*$ labeled items from all graphs involved in the derivation step does not impose any new restrictions. □

It is expedient to define gluing constructions for graphs and rules and to prove several properties that will allow us to use those definitions later in this section for the proof of the main theorem.

Definition 6 (Gluing) The gluing of two graphs G and H along the graph $A \subseteq G, H$ is the graph denoted $G +_A H$ with node set $(V_G +_{V_A} V_H)$ and edge set $(E_G +_{E_A} E_H)$, $V_G - V_A$ and $V_H - V_A$ assumed disjoint, similarly for the edges.

If A has no edges, we simply write $G +_{V_A} H$ for $G +_A H$, also omitting the set brackets if no confusion is possible, and $G + H$ if V_A is empty. If A is a set of nodes which is not a subset of $V_G \cap V_H$, $G +_A H$ is an abbreviation for $G +_{A \cap V_G \cap V_H} H$.

Definition 7 (Parallel Composition) Given two rules ϱ_1 and ϱ_2 and a set $A \subseteq V_{K_1} \cap V_{K_2}$ of common nodes, the *parallel composition* of ϱ_1 and ϱ_2 as the rule $\varrho_1 +_A \varrho_2 := (L_1 +_A L_2 \hookleftarrow K_1 +_A K_2 \hookrightarrow R_1 +_A R_2)$. If $\varrho_2 = (L_2 \hookleftarrow K_2 \hookrightarrow R_2)$ is identical ($L_2 = K_2 = R_2$), then $\varrho_1 +_A \varrho_2$ is called an *enlargement* of ϱ_1.

We recall two properties of the parallel composition collectively known as the Parallelism Theorem (Theorem 4.7 in [Ehr79]).

Lemma 2 (Parallel Step Lemma [Ehr79]) *If* $G_1 \overset{\varrho_1}{\Longrightarrow} H_1$ *and* $G_2 \overset{\varrho_2}{\Longrightarrow} H_2$, *then* $G_1 +_A G_2 \overset{\varrho_1 +_{A'} \varrho_2}{\Longrightarrow} H_1 +_A H_2$ *where* A' *is the common preimage of* A *via both matches.*

Lemma 3 (Sequential Decomposition [Ehr79]) *A derivation step* $G \overset{\varrho}{\Rightarrow} H$ *of the parallel composition of two rules* $\varrho = \varrho_1 +_A \varrho_2$ *can be decomposed* sequentially *into two derivation steps* $G \overset{\varrho_1}{\Rightarrow} J \overset{\varrho_2}{\Rightarrow} H$, *assigning the same image to the nodes of* A *in both matches.*

It will frequently become necessary in the subsequent constructions to transfer a rule application to a larger graph, or to restrict attention to a certain part of a derivation step. The next lemma allows under certain conditions to conclude that the graphs involved in derivation steps may be enlarged or shrunk without interfering with the derivation step. It is partly similar to the Embedding and Restriction Theorems (Theorems 6.14 and 6.18 in [EEPT06]) but additionally allows to enlarge and shrink rules, not just their contexts of application.

Lemma 4 (Embedding Lemma) *Let* X *be a graph,* ϱ *a rule,* A *a subgraph of* K, *and* ϱ' *the enlargement* $\varrho +_A id_X$. *Then, for any match* $g : L \to G$, *writing* g' *for the enlarged version of* g, *and* L' *for* $L +_A X$ *and so on,*

$$(1)\ G \overset{\varrho,g}{\Rightarrow} H \ \textit{implies}\ G' \overset{\varrho, g' \circ l'}{\Rightarrow} H'. \qquad (2)\ G \overset{\varrho,g}{\Rightarrow} H \ \textit{implies}\ G' \overset{\varrho', g'}{\Rightarrow} H'.$$

Conversely,

$$(3)\ G' \overset{\varrho,g'\circ l'}{\Rightarrow} H' \ \textit{implies}\ G \overset{\varrho,g}{\Rightarrow} H. \qquad (4)\ G' \overset{\varrho',g'}{\Rightarrow} H' \ \textit{implies}\ G \overset{\varrho,g}{\Rightarrow} H.$$

$$\begin{array}{ccccc} L & \hookleftarrow & K & \hookrightarrow & R \\ {\scriptstyle l'}\downarrow & & \downarrow & & \downarrow{\scriptstyle r'} \\ L' & \hookleftarrow & K' & \hookrightarrow & R' \\ {\scriptstyle g'}\downarrow & & \downarrow & & \downarrow{\scriptstyle h'} \\ G' & \hookleftarrow & D' & \hookrightarrow & H' \end{array}$$

Proof. We notice that the presence of items of X in the graph to which the rule is applied does not influence the satisfaction of the dangling condition because all edges from X are attached to nodes of K, and those are not deleted.

$G \overset{\varrho,g}{\Rightarrow} H$ implies $G' \overset{\varrho,g'\circ l'}{\Rightarrow} H'$: the intermediary graph is constructed as $D = G - g(L-K)$. Since X is glued to nodes of K, the dangling condition is respected and the outer difference in $D' = (G - g(L-K))' = (G' - g(L-K))$ is a graph. Completing the derivation step results in H'.

$G \overset{\varrho,g}{\Rightarrow} H$ implies $G' \overset{\varrho',g'}{\Rightarrow} H'$: the only difference to the first case is that now $g(L-K)$ is replaced with $g(L'-K')$ and analogously $R-K$ with $R'-K'$ when completing the derivation step. But $L'-K' = L-K$ and $R'-K' = R-K$, so the enlarged rule has the same effect as the original rule.

$G' \overset{\varrho,g'\circ l'}{\Rightarrow} H'$ implies $G \overset{\varrho,g}{\Rightarrow} H$: as argued before, gluing in X along A does not interfere with the dangling condition. One weakens the conclusion of the derivation step to obtain a proposition about $G'-(X-A)$, $D'-(X-A)$, $H'-(X-A)$ only, which are indeed the graphs G, D and H.

$G' \overset{\varrho',g'}{\Rightarrow} H'$ implies $G \overset{\varrho,g}{\Rightarrow} H$: one proceeds as in the third case and ignores what is known about the nodes and edges of X from the definition of the derivation step, to obtain a statement about a derivation step with $(L'-(X-A) \hookleftarrow K'-(X-A) \hookrightarrow R'-(X-A)) = \varrho$ from $G'-(X-A) = G$. The result of the first half of the derivation step is $D'-(X-A) = D$ since $(G'-g(L-K))-(X-A)$ is equal to $(G'-(X-A)-g(L-K)) = D$, and this is also a graph (no dangling edges) for the reason stated in the introductory remark. H is obtained as before, as $H = (D + h(R-K))$. □

A comparison with Petri net and Szilard languages suggests the question whether the image of a derivation language under a non-injective relabeling of the rules (assigning ambiguous labels to some rules) is still a derivation language. In the cases of Petri net [Jan87] and Szilard [Hö75] languages, this increases the generative power. We devote the next pages to showing how a grammar GG can be modified such that the derivation language of the modified grammar GG_h indeed simulates two distinct rules with the same label. Two letters a_1 and a_2 used for rules in GG would be merged to a common image $h(a_1) = h(a_2) = a$, and $\mathcal{L}_D(GG_h) = h(\mathcal{L}_D(GG))$ and $\mathcal{L}_\$(GG_h) = h(\mathcal{L}_\$(GG))$. Closure under arbitrary letter-to-letter homomorphisms then follows by repeated application of such mergers.

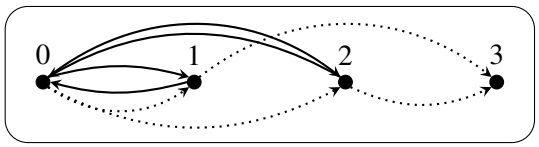
Since the effect of a rule is always the same, when simulating the effects of two different rules originally labeled a_1, a_2 with a single compound rule ϱ_a, it is impossible to choose between the two in a derivation step. Both will necessarily be applied. But since one of the effects

is unwanted, the idea is to neutralise it by extending the graph of the simulated grammar with a specially constructed sandbox which contains the left hand side of the unwanted rule to guarantee its applicability, and then forcing the match of the unwanted rule into the sandbox, where it cannot interfere with the simulation. Since the sandbox will be depleted after each usage, it must also be replenished by the compound rule.

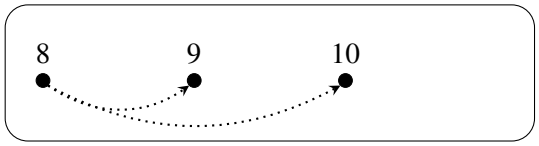
We now introduce three control graphs (Control, Control′ and Control″) that guide the application of the compound rule and the other rules in the simulation. Roughly, a suspended copy of the simulated grammar's current graph G will be glued to a node of Control. The other nodes of Control will serve as a sandbox, initialised by gluing to them the suspended left hand sides of a_1 and a_2 (as one does not know in advance which of a_1 or a_2 will be unwanted, both left hand sides must be available in the sandbox). We call the result of this construction an *augmented* graph $\widetilde{G}$. It is the graph on which the modified grammar operates to perform the simulation.

In the first control graph, the role of node 0 is to provide a handle for suspending the simulation state. Control has no symmetry exchanging 0 with another node. The nodes 1 and 2 will serve as the sandbox to execute the unwanted rule, while 3 only collects garbage. Control′, the control graph for unambiguously relabeled rules, has a node that can only match to 0 and this ensures the correct simulation of these rules. On the other hand, Control″ will be used for compound rules. This graph has two nodes (4 and 5) that can match to 0. The special construction of Control″ directs the match of the a_1 part of the compound rule to the sandbox when the a_2 part is matched on the graph of the simulated grammar and vice versa.

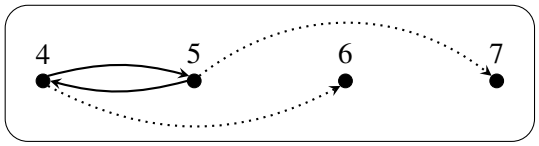
Construction 1 (Control Graphs for the Simulation) The basic control graph Control is defined as the graph with node set $\{0,1,2,3\}$, all nodes labeled $*$ and the edges shown: edges represented in dotted lines bear the label *new* and edges represented in thick lines have the label *link*.



The control graph Control′ for unambiguously relabeled rules is defined as the graph with node set $\{8,9,10\}$, label $*$ for all three nodes and just two *new* edges:



The control graph Control″ for ambiguously relabeled rules is defined as the graph with node set $\{4,5,6,7\}$, label $*$ for all four nodes, and the edges shown.

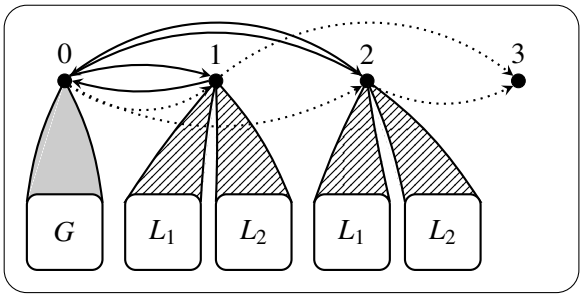


We also assume that in the subsequent constructions, no other *new* and *link* labeled edges occur except those introduced with these three graphs.

Construction 2 (Augmented Graphs) Given a grammar $GG = (T, \mathcal{R}, \mathfrak{r}, S)$, two letters $a_1, a_2 \in T$, $\mathfrak{r}(a_1) = \varrho_1$, $\mathfrak{r}(a_2) = \varrho_2$, and a label $*$ not occurring in any rule of $\mathcal{R}$ nor in S, for $i, j \in \{1,2\}$, let

$$\widetilde{G}^{(i,j)} := \text{Control} \underset{0,1,2,3}{+} \left(G^0 + \left({L_1}^i \underset{1,2}{+} {L_2}^j \right) \right),$$

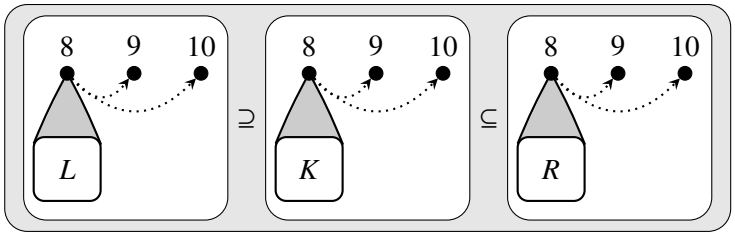
be the graph consisting of one copy of G suspended to the node 0 of Control, and one supplementary copy of the left hand side of each one of the rules ϱ_1, ϱ_2 suspended either on 1 or on 2. All four possibilities $(i, j) = (1,1), (1,2), (2,1)$ or $(2,2)$ are represented in a schematic drawing; an actual $\widetilde{G}^{(i,j)}$ graph is obtained by choosing one L_1 and one L_2 copy and striking the other.



$\widetilde{G}^{(1,2)}$ and $\widetilde{G}^{(2,1)}$ are isomorphic, and so are $\widetilde{G}^{(1,1)}$ and $\widetilde{G}^{(2,2)}$, but since we have chosen to consistently track the nodes of the Control graph through a derivation sequence by their identities, there are four variants, not two. The role of i and j is as follows: since one does not know in advance whether the simulation requires a copy of L_1 or a copy of L_2 at a given step to neutralise the collateral effect, both must be "on stock"; however, just placing a new left hand side where the current one was consumed is not a solution, because *one* of the rule effects will be applied "for real" on the graph suspended on node 0. Placing a new left hand side on 0, however, could interfere with the simulation. Therefore, a more complicated scheme is necessary. Our solution requires, for compound rules, an alternation between the cases $i = j$ and $i \neq j$. Unambiguously relabeled rules ignore the sandbox part and leave i and j unchanged after the derivation step.

Construction 3 (Augmented Rules, First Case) A rule $\varrho = (L \hookleftarrow K \hookrightarrow R)$ is changed into

$$\varrho' := \left(\text{Control}' \underset{8}{+} L^8 \hookleftarrow \text{Control}' \underset{8}{+} K^8 \hookrightarrow \text{Control}' \underset{8}{+} R^8 \right).$$



Lemma 5 (Unique Simulation Step in the First Case) *If a rule ϱ is transformed to ϱ' according to Construction 3, and* Rest *contains no $*$ labeled nodes or edges besides $1,2,3$, there is a ϱ derivation from G to H iff there is a corresponding ϱ' derivation between the augmented graphs:*

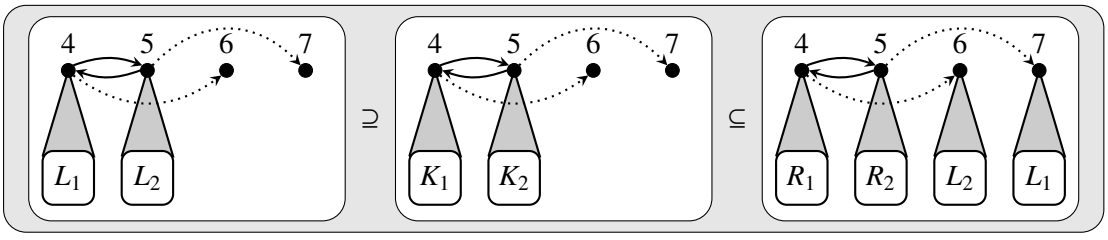
$$G \overset{\varrho}{\Rightarrow} H \quad \textit{iff } \exists \text{Rest} \in \text{Graphs } \textit{and } i,j \in \{1,2\} \textit{ such that } \left(\widetilde{G}^{(ij)} \underset{1,2,3}{+} \text{Rest} \overset{\varrho'}{\Rightarrow} \widetilde{H}^{(ij)} \underset{1,2,3}{+} \text{Rest}\right).$$

Proof. $G \overset{\varrho}{\Rightarrow} H$ implies $\widetilde{G}^{(ij)} \underset{1,2,3}{+} \text{Rest} \overset{\varrho'}{\Rightarrow} \widetilde{H}^{(ij)} \underset{1,2,3}{+} \text{Rest}$: By Lemma 1, $G \overset{\varrho,g}{\Rightarrow} H$ implies $G^0 \overset{\varrho^8,g^{8,0}}{\Longrightarrow} H^0$, which in turn implies $\widetilde{G}^{(ij)} \overset{\varrho^8,g^{8,0}}{\Longrightarrow} \widetilde{H}^{(ij)}$ by Lemma 4 (1), and $\widetilde{G}^{(ij)} +_{1,2,3} \text{Rest} \overset{\varrho',g'^{8,0}}{\Longrightarrow} \widetilde{H}^{(ij)} +_{1,2,3} \text{Rest}$ by Lemma 4 (2), where ϱ' is obtained from rule ϱ as in Construction 3, and g' is the enlargement of the match (as in Lemma 4) by the graph Control′, glued in along 8.

$\widetilde{G}^{(ij)} +_{1,2,3} \text{Rest} \overset{\varrho'}{\Rightarrow} \widetilde{H}^{(ij)} +_{1,2,3} \text{Rest}$ implies $G \overset{\varrho}{\Rightarrow} H$: let q be the match used in the derivation step in the antecedent. The only possible image for the node 8 is the node 0 (no new $*$ labeled nodes nor control edges have been added, and there is only one morphism from Control′ to Control). This constrains the images of the nodes from L_ϱ to lie within G, because all edges e of $L_{\varrho'}$ with $q(s(e)) = 0$ in $\widetilde{G}^{(ij)} +_{1,2,3} \text{Rest}$ not labeled *new* are suspension edges from G^0, by construction. $\text{Rest} + \widetilde{\emptyset}^{(ij)}$ is linked only to nodes of $g(K)$, therefore an application of Lemma 4 (3) with $X = \text{Rest} + \widetilde{\emptyset}^{(ij)}$ glued in along $\{1,2,3\}$, followed by the second proposition of Lemma 1 to reverse the suspension, concludes the proof. □

Construction 4 (Augmented Rules, Second Case) From two rules $\mathfrak{r}(a_1) = \varrho_1$ and $\mathfrak{r}(a_2) = \varrho_2$, a rule ϱ_a is constructed by gluing along the $*$ nodes the following rules, in the sequel referred to as the sub-rules of ϱ_a, ${\varrho_1}^4$, ${\varrho_2}^5$, $\varrho_{\text{add}_1} := (\emptyset \hookleftarrow \emptyset \hookrightarrow L_1)^7$, $\varrho_{\text{add}_2} := (\emptyset \hookleftarrow \emptyset \hookrightarrow L_2)^6$, $id_{\text{Control}''}$:

$$\varrho_a := \left({\varrho_1}^4 + {\varrho_2}^5 + \varrho_{\text{add}_1} + \varrho_{\text{add}_2}\right) \underset{4,5,6,7}{+} id_{\text{Control}''}.$$



Lemma 6 (Essentially Unique Simulation Step in the Second Case) *If the rules ϱ_1 and ϱ_2 are transformed into ϱ_a according to Construction 4, then if $\lambda^{-1}_{\text{Rest}}(*) \subseteq \{1,2,3\}$ and $\exists i,j,k,l \in \{1,2\}$,*

$$(G \overset{\varrho_1}{\Rightarrow} H \vee G \overset{\varrho_2}{\Rightarrow} H) \quad \textit{iff} \quad \left(\exists \text{Rest}, \text{Rest}' \in \text{Graphs} : \widetilde{G}^{(ij)} \underset{1,2,3}{+} \text{Rest} \overset{\varrho_a}{\Rightarrow} \widetilde{H}^{(kl)} \underset{1,2,3}{+} \text{Rest}'\right).$$

Proof. $G \overset{\varrho_1}{\Rightarrow} H \vee G \overset{\varrho_2}{\Rightarrow} H \;\Rightarrow\; \exists \text{Rest}, \text{Rest}' \in \text{Graphs} : \widetilde{G}^{(ij)} +_{1,2,3} \text{Rest} \overset{\varrho_a}{\Rightarrow} \widetilde{H}^{(kl)} +_{1,2,3} \text{Rest}'$:
Assume first that $G \overset{\varrho_1}{\Rightarrow} H$. Due to the symmetrical construction of ϱ_a, the case of $G \overset{\varrho_2}{\Rightarrow} H$ is completely analogous.

There is a match g such that $G \overset{\varrho_1,g}{\Longrightarrow} H$. We apply the Suspension Lemma (Lemma 1) to suspend ϱ_1 as ${\varrho_1}^4$ and the match as $g^{4,0}$, $G^0 \overset{{\varrho_1}^4,g^{4,0}}{\Longrightarrow} H^0$, and repeat the same with the derivation step $L_2 \overset{\varrho 2,f}{\Longrightarrow} R_2$ to obtain ${L_2}^j \overset{{\varrho_2}^5,f^{5,j}}{\Longrightarrow} {R_2}^j$.

Fix matches of the sub-rules ϱ_{add_1} and ϱ_{add_2} which assign 7 to 3 and 6 to $3-j$ (6 is mapped one of 1 and 2, but not the same as j). Applying proposition (1) of the Embedding Lemma to each of the sub-rule derivations in turn results in a derivation of four steps. Leaving the matches implicit,

$$
\begin{aligned}
\widetilde{G}^{(ij)} +_{1,2,3} \mathrm{Rest} \quad &= \quad \left(\mathrm{Control} +_{0,1,2,3} \left(G^0 + ({L_1}^i +_{1,2} {L_2}^j)\right)\right) +_{1,2,3} \mathrm{Rest} \\
&\overset{{\varrho_1}^4}{\Longrightarrow} \quad \left(\mathrm{Control} +_{0,1,2,3} \left(H^0 + ({L_1}^i +_{1,2} {L_2}^j)\right)\right) +_{1,2,3} \mathrm{Rest} \\
&\overset{{\varrho_2}^5}{\Longrightarrow} \quad \left(\mathrm{Control} +_{0,1,2,3} \left(H^0 + ({L_1}^i +_{1,2} {R_2}^j)\right)\right) +_{1,2,3} \mathrm{Rest} \\
&\overset{(\emptyset \hookleftarrow \emptyset \hookrightarrow L_1)^7}{\Longrightarrow} \left(\mathrm{Control} +_{0,1,2,3} \left(H^0 + ({L_1}^i +_{1,2} {R_2}^j)\right)\right) +_{1,2,3} (\mathrm{Rest} +_3 {L_1}^3) \\
&\overset{(\emptyset \hookleftarrow \emptyset \hookrightarrow L_2)^6}{\Longrightarrow} \left(\mathrm{Control} +_{0,1,2,3} \left(H^0 + ({L_1}^i +_{1,2} {R_2}^j)\right)\right) +_{1,2,3} \left(\mathrm{Rest} +_{i,3} \left({L_1}^3 + {L_2}^{3-j}\right)\right) \\
&= \quad \left(\mathrm{Control} +_{0,1,2,3} \left(H^0 + ({L_1}^i +_{1,2} {L_2}^{3-j})\right)\right) +_{1,2,3} \underbrace{\left(\mathrm{Rest} +_{i,3} \left({L_1}^3 + {R_2}^j\right)\right)}_{\mathrm{Rest}'}
\end{aligned}
$$

Here, $k = i$ because the subgraph ${L_1}^i$ is not used up and can be re-used as a constituent of the augmented graph $\widetilde{H}^{(kl)}$. The new copy of L_1 which has been suspended to 3 by the application of the sub-rule ϱ_{add_1} becomes part of $Rest'$.

$l = 2$ when $j = 1$ and $l = 1$ when $j = 2$, because the subgraph ${L_2}^j$ has been used up and transformed into ${R_2}^j$ by the application of the sub-rule ${\varrho_2}^5$. The subgraph ${R_2}^j$ thus created becomes part of Rest′. A new copy of L_2 has instead been suspended to the node l by the rule ϱ_{add_2}. Note that $k = l$ if $i \neq j$ and $k \neq l$ if $i = j$.

After applying the Embedding Lemma, forms (3) and (2), to enlarge the rules by the control graph Control″ (the image of Control″ in Control is determined by the node mapping), this yields the desired derivation step by Parallel Composition (Lemma 2), by the construction of ϱ_a.

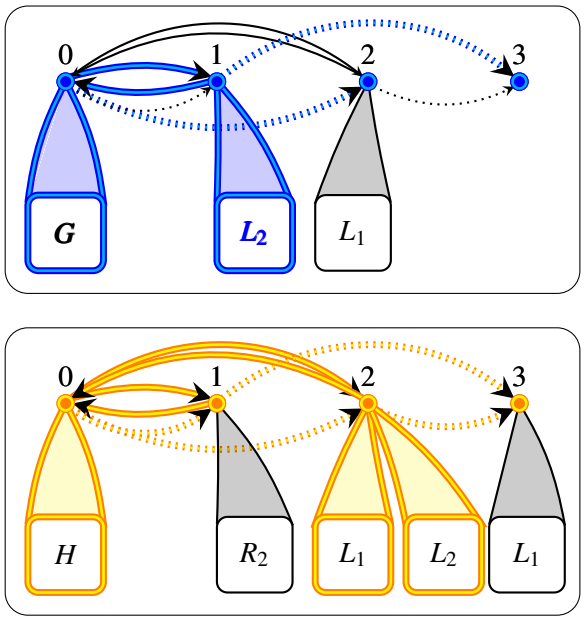
$\exists \mathrm{Rest}, \mathrm{Rest}' \in \mathrm{Graphs} : \widetilde{G}^{(ij)} +_{1,2,3} \mathrm{Rest} \overset{\varrho_a}{\Rightarrow} \widetilde{H}^{(kl)} +_{1,2,3} \mathrm{Rest}' \Rightarrow G \overset{\varrho_1}{\Rightarrow} H \vee G \overset{\varrho_2}{\Rightarrow} H$: We proceed again by case analysis. The following is true for any match q of ϱ_a in $\widetilde{G}^{(ij)}$: By construction of Control″ and Control, q must assign either 4 to 0 and 5 to j, or 5 to 0 and 4 to i. In the first case, we seek to arrive at the conclusion that $G \overset{\varrho_1}{\Rightarrow} H$, the second case is again completely analogous and will, mutandis mutatis, lead to $G \overset{\varrho_2}{\Rightarrow} H$.

The derivation step using ϱ_a is equivalent to a parallel composition of the following two derivation steps: the first step is an application of an embedded form of ϱ_1^0 with a match mapping 0 to 4. The other step leaves unchanged the graph suspended on 0, but performs the rest of ϱ_a. By Lemma 3, we can extract the first derivation step by sequential decomposition, with suitable matches both mapping $4,5,6,7$ to $0, j, i, 3$ in that order.

$$
\widetilde{G}^{(ij)} \underset{1,2,3}{+} \mathrm{Rest} \overset{{\varrho_1}^4 \underset{4,5,6,7}{+} id_{\mathrm{Control}''}}{\Longrightarrow} \widetilde{H}^{(ij)} \underset{1,2,3}{+} \mathrm{Rest} \overset{\varrho'''}{\Longrightarrow} \widetilde{H}^{(kl)} \underset{1,2,3}{+} \mathrm{Rest}',
$$

where $\text{Rest}' = \text{Rest} +_{1,2,3} \left(R_2{}^j + L_1{}^3\right)$, $k = i$, $l = 3 - j$, $\varrho''' = \left((\varrho_2{}^5 + \varrho_{\text{add}_1} + \varrho_{\text{add}_2}) \underset{4,5,6,7}{+} id_{\text{Control}''}\right)$, because from $\widetilde{H}^{(ij)} +_{1,2,3} \text{Rest} = (Control +_{1,2,3,4} (H^0 +_{1,2,3} (L_1{}^i +_{1,2} L_2{}^j))) +_{1,2,3} \text{Rest}$, with the given information about the match, a derivation step ϱ''' must yield $(Control +_{1,2,3,4} (H^0 +_{1,2,3} (L_1{}^i +_{1,2} R_2{}^j))) +_{1,2,3} (\text{Rest} +_{3,i} (L_1{}^3 + L_2{}^i))$. This can again be rearranged into the desired form $\widetilde{H}^{(kl)} +_{1,2,3} \text{Rest}'$. An application of the Embedding Lemma (4) and (3) to the first step, followed by the Suspension Lemma, allows us to arrive at the conclusion $G \overset{\varrho_1}{\Rightarrow} H$. □

We have represented the graph $\widetilde{G}^{(21)}$ schematically and highlighted a match of ϱ_a in blue, in the case $G \Rightarrow^{\varrho_1} H$. The second drawing represents the situation after the derivation step induced by this match. We have highlighted the resulting subgraph $\widetilde{H}^{(22)}$ and represented Rest' in gray.



Construction 5 (Merging Two Letters) Given a homomorphism $h : T^* \to T'^*$, $T' = (T - \{a\} + \{a_1, a_2\})$, and a grammar $GG = (T, \mathcal{R}, \mathfrak{r}, S)$, let $GG_h := (T', \mathcal{R}', \mathfrak{r}', \widetilde{S}^{(12)})$, $\mathfrak{r}'(a) := \varrho_a$ built from $\mathfrak{r}'(a_1)$, $\mathfrak{r}'(a_2)$ according to Construction 4, $\forall b \neq a$, $\mathfrak{r}'(b) := \mathfrak{r}(b)'$ according to Construction 3.

Closure under strictly letter-to-letter homomorphisms is an operation on classes of languages often considered in the literature on formal languages and sometimes denoted by $\mathcal{H}_1$.

Theorem 1 (Closure under Letter-to-Letter Homomorphisms) $\mathcal{L}_\$$ *is closed under* $\mathcal{H}_1$.

Proof. Assume, w.l.o.g., that the homomorphism h merges two letters a_1 and a_2 to a single image a and maps each other letter to itself. Any letter-to-letter homomorphism between finite alphabets can be factored into finitely many such elementary mergers. Let $L = \mathcal{L}_\$(GG)$. We shall show that Construction 5 results in a grammar GG_h such that $\mathcal{L}_\$(GG_h) = h(\mathcal{L}_\$(GG))$. We prove the proposition by induction over the length of a derivation, on the hypothesis that $G \Rightarrow^{*w}_{GG} H$ iff there are two graphs Rest, Rest' such that $\widetilde{G} +_{1,2,3} \text{Rest} \Rightarrow^{*h(w)}_{GG_h} \widetilde{H} +_{1,2,3} \text{Rest}'$, where $\widetilde{G}$ is one of

$\widetilde{G}^{(11)}$, $\widetilde{G}^{(12)}$, $\widetilde{G}^{(21)}$, $\widetilde{G}^{(22)}$ and $\widetilde{H}$ is one of $\widetilde{H}^{(11)}$, $\widetilde{H}^{(12)}$, $\widetilde{H}^{(21)}$, $\widetilde{H}^{(22)}$. The induction step is already proven in Lemmata 5 and 6, since the required property that the nodes labeled $*$ are precisely $0,1,2,3$ is fulfilled by all graphs reachable in GG as no derivation step can introduce a label occurring neither in the graph being transformed nor in the rule. □

3.2 Context Free Languages and Beyond

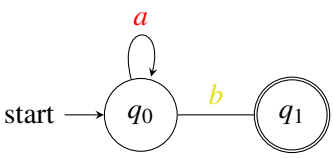
From formal language theory, we know that $REG \subset CF \subset CS \subset RE$, where REG stands for the regular languages, CF for the context free languages, CS for the context sensitive languages and RE for all recursively enumerable languages. To each of the four classes corresponds a type of grammar, known as type 3, 2, 1 and 0, respectively [HU79]. In this subsection, we locate $\mathcal{L}_\$$ above the context free languages in a first hierarchy result, $CF \subseteq \mathcal{L}_\$$. This result will immediately exposed as being quite weak, because our proof already shows the closure of $\mathcal{L}_\$$ under intersections, from which follows $CF \subset \mathcal{L}_\$$. We also establish that the Szilard languages are a subset of $\mathcal{L}_\$$, and provide a grammar for the language $\{a^{2^n} \mid n \in \mathbb{N}\}$. We shall start with the inclusion $REG \subseteq \mathcal{L}_\$$, which is also useful subsequently.

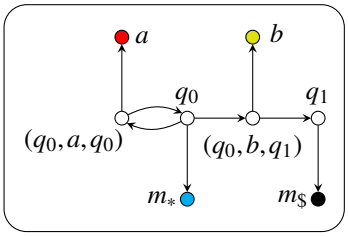
The construction for transforming a nondeterministic finite automaton A into a graph grammar GG whose derivation language $\mathcal{L}_\$(GG)$ is the language $L(A)$ accepted by A is straightforward.

Lemma 7 (Regular Languages) *Every regular language is in $\mathcal{L}_\$$:*

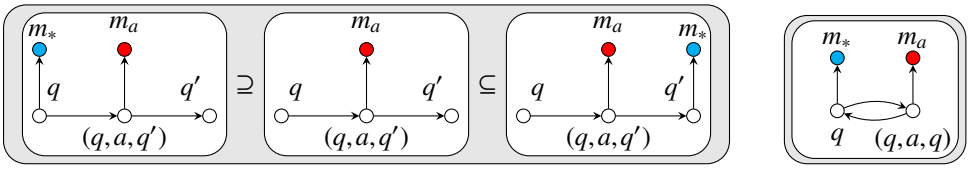
$$REG \subseteq \mathcal{L}_\$.$$

Proof. Given a finite automaton $A = (Q, T, \delta, q_0, Q_F)$ with state set Q, one initial state $q_0 \in Q$, alphabet T, transition relation $\delta \subseteq Q \times T \times Q$, final states $Q_F \subseteq Q$, let $s_A : Q \to$ Graphs be the function assigning to each state $\hat{q}$ a graph $s_A(\hat{q})$ with edge set $V = Q \cup \delta \cup \{m_a \mid a \in T\} \cup \{m_*\} \cup \{m_\$\}$, the union assumed disjoint. That is, every state of A, as well as every transition, becomes a node; besides, some auxiliary nodes are introduced. The edge set $E = \{e_{qm_a} \mid (q,a,q') \in \delta\} \cup \{e_{m_a q'} \mid (q,a,q') \in \delta\} \cup \{e_{(q,a,q')m_a} \mid (q,a,q') \in \delta\} \cup \{e_{qm_\$} \mid q \in Q_F\} \cup \{e_{\hat{q}m_*}\}$ with $s(e_{ab}) = a$ and $t(e_{ab}) = b$ attaches the state nodes to the transition nodes and to the special marker nodes m_*, $m_\$$ and the transition nodes to state nodes and alphabet nodes. The marker nodes are distinguishable: $\forall a \in T \cup \{*, \$\} : \lambda(m_a) = a$, all other nodes obtain a nondescript label $x \notin T$: $\forall v \in V - \{m_a \mid a \in T \cup \{*, \$\}\} : \lambda(v) = x$. All edges also have the same label, $\lambda(e) = x$. We have represented an automaton A and the graph $s_A(q_0)$.

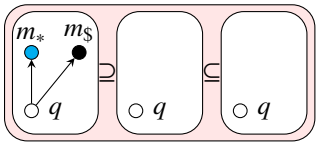




The rules are the same for every automaton. For each letter $a \in T$, let ϱ_a and $\varrho_{\hat{a}}$ be the rules (the latter being an identical rule):



The end rule $\varrho_\$$ removes the nodes labeled \$ and $*$ and the edges attached to them, when they are linked to a common node q.

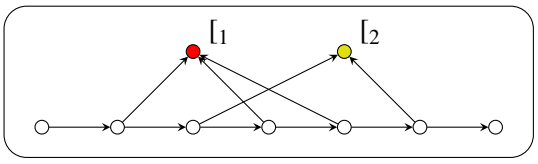


It is easy to see that $(q,a,q') \in \delta$ iff $s_A(q) \overset{\mathfrak{r}(a)}{\Rightarrow} s_A(q')$ or $s_A(q) \overset{\mathfrak{r}(\mathring{a})}{\Rightarrow} s_A(q')$. By induction over the length of an accepted or generated word, the fact that the end rule is applicable in $s_A(q)$ iff $q \in Q_F$, and an application of Theorem 1 with the homomorphism mapping $\mathring{a}$ and a to a for each $a \in \Sigma$, Hence the languages of the automaton A and the grammar are equal. □

The extended Dyck language [Okh12] over m pairs of parentheses $[_1]_1 \dots [_m]_m$ and n extra symbols $x_1 \dots x_n$, $\hat{D}_{m,n}$, is the language generated by the context free grammar with variables $V = \{S\}$, terminal alphabet $\bigcup_{i\in\{1,\dots,m\}}\{]_i,[_i\} \cup \{x_j \mid j \in \{1,\dots,n\}\}$ and productions $\{(S, SS), (S,\epsilon)\}$ $\cup \{(S, [_i S]_i) \mid i \in \{1,\dots,m\}\} \cup \{(S,x_j) \mid j \in \{1,\dots,n\}\}$. These are a building block for the context free languages. We show that every such language is the derivation language of a graph grammar.

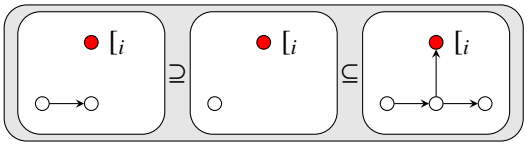
Lemma 8 (Extended Dyck languages) *The extended Dyck languages are in* $\mathcal{L}_\$$.

Proof. Let $X_n = \{x_j \mid j \in \{1\dots n\}\}$, $\Sigma_{m,n} = \{[_i]_i \mid i \in \{1\dots m\}\} \cup X_n$. Given an alphabet X, let $G(w)$ denote the graph encoding the word $w \in X^*$ in the following manner: there is one node m_a labeled a for each $a \in X$, one node v_{-1} and one node $v_{|w|}$, and one node v_i for each $i \in \{0,\dots,|w|-1\}$. Each node v_i, $i \in \{-1,\dots,|w|-1\}$ is linked with an edge to its successor v_{i+1}, and if $i > 0$ also to the node m_a such that the i-th letter $w_i = a$. The edges and the nodes v_i are all labeled $\bullet$. As an example, we have represented the graph $G([_1[_2[_1[_1[_2)$.



Let $GG_{m,n} = (\Sigma_{m,n}, \mathcal{R}, \mathfrak{r}, S)$ be a grammar where S is the graph with $m+n$ unconnected nodes m_a, one for each opening bracket $a \in \{[_i \mid i \in \{1\dots m\}\}$ and two nodes v_0 and v_1 linked by a single edge. The end rule $\mathfrak{r}(\$)$ is the rule $(S \hookleftarrow \emptyset \hookrightarrow \emptyset)$ that just deletes a connected component isomorphic to S. All rules $\mathfrak{r}(x_j)$ are identical rules that check for the existence of a single $\bullet$ labeled node.

All rules $\mathfrak{r}([_i)$ and $\mathfrak{r}(]_i)$ are mirror images of each other, $L_{[_i} = R_{]_i}$, $K_{[_i} = K_{]_i}$, $L_{]_i} = R_{[_i}$. This is $\mathfrak{r}([_i)$, which extends a chain of nodes by one node linked to $m_{[_i}$, using the graph as a stack. $\mathfrak{r}([_i)$ pushes a symbol on the stack while $\mathfrak{r}(]_i)$ removes the top stack symbol:



We use a well-known alternative characterisation of the Dyck languages: $D_m := \hat{D}_{m,0}$ is the equivalence class $[\epsilon]_{\sim_m}$ of ϵ under the congruence $\sim_m$ induced by $\{([_i]_i, \epsilon) \mid i \in \{1,\ldots,m\}\}$ (see Berstel [Ber79]). This is easily extended to the extended Dyck languages, the congruence $\sim_{m,n}$ now being induced by $\{([_i]_i, \epsilon) \mid i \in \{1,\ldots,m\}\} \cup \{(x_j, \epsilon) \mid j \in \{1,\ldots,n\}\}$. This relation can be used as a confluent rewriting system, and there is a unique reduced word $w_{red} \in [w]_{\sim_{m,n}}$ for any word $w \in \Sigma_{m,n}$ and $w \in \hat{D}_{m,n}$ iff $w_{red} = \epsilon$.

We show by induction over the length n of a word w that $w \in \mathcal{L}_D(GG_{m,n})$ implies $S \overset{w}{\underset{GG_{m,n}}{\Rightarrow}}{}^* G(w_{red})$, and the derivation is uniquely determined by the word w. The statement is obvious for $w = \epsilon$; it is also true for $w = ux_j$ for any $x_j \in X_n$ and $u \in \Sigma^*_{m,n}$ of length n since the rules $\mathfrak{r}(x_j)$ do not modify the graph. It is also true for any $u[_i$, $[_i \in \Sigma_{m,n}$ because an opening bracket does not create any new opportunity for applying a reduction to the string. It is also true for any $u]_i$, since $\mathfrak{r}(]_i)$ can *only* be applied when the last rule was $\mathfrak{r}([_i)$ or any $\mathfrak{r}(x_j)$, $j \in \mathbb{N}$ (which can again be shown by induction, examining the effects and applicability of the rules) and has a unique effect: $]_i$ takes back the effect of the last $[_i$, which is to extend the graph by one node encoding an opening bracket.

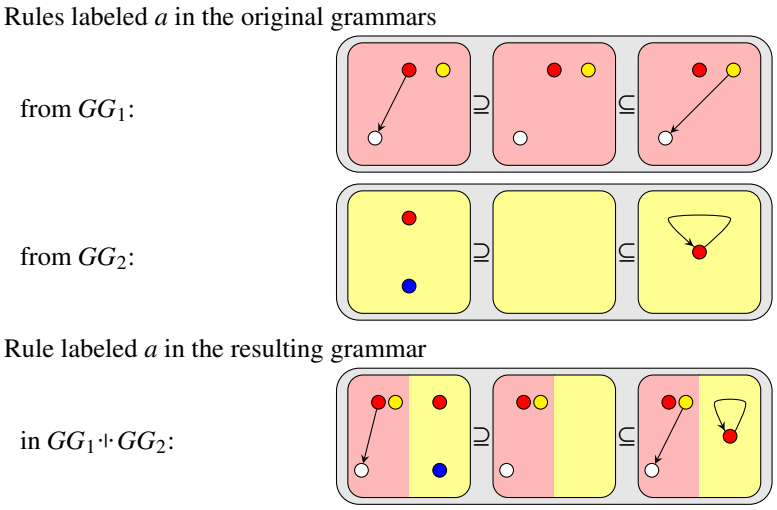
$\mathcal{L}_\$(GG_{m,n}) \subseteq \hat{D}_{m,n}$: We have just shown that any derivation labeled w ends in state $G(w_{red})$. The end rule can be applied only to a graph having $G(\lambda)$ as a connected component. This can clearly not occur until the stack is cleared again, therefore $w \in \hat{D}_{m,n}$.

$\hat{D}_{m,n} \subseteq \mathcal{L}_\$(GG_{m,n})$: Any word $w \in \hat{D}_{m,n}$ is related to λ by the relations introduced at the start of this proof. We show how to construct a derivation by applying these relations in any direction, by induction over the number of rules needed to go from λ to w. The induction step hinges on the fact that $\mathfrak{r}(x_j)$ rules are always applicable and have no effect on the graph, while any sequence $[_i]_i$ can also be inserted at any position in a derivation because starting at any reachable graph (induction) it can be applied, uniquely corresponding to a derivation with no effect. □

The next definition serves to show closure of $\mathcal{L}_\$$ under intersections, after which we will be able to prove the inclusion of the context free languages.

Definition 8 (Label-Disjoint Union) Given two graphs G and H, let $G \cdot|\cdot H := G_{(0)} + H_{(1)}$, where $X_{(i)}$ denotes the graph X with $\lambda_X(x) = (x, i)$. Let this operation be extended to morphisms and rules: if $f_1 : G_1 \to H_1$ and $f_2 : G_2 \to H_2$, let $f_1 \cdot|\cdot f_2$ be the morphism such that $(f_1 \cdot|\cdot f_2)_v(n, i) = ((f_1)_v(n), i)$ and likewise for edges. Let it be extended to grammars by removing all rules which are not present in one of the grammars: Given $GG_1 = (T_1, \mathcal{R}_1, \mathfrak{r}_1, S_1)$ and $GG_2 = (T_2, \mathcal{R}_2, \mathfrak{r}_2, S_2)$, $GG_1 \cdot|\cdot GG_2$ is the grammar with rule labels $T_1 \cap T_2$, rules $\varrho_1 \cdot|\cdot \varrho_2$ for any pair of rules ϱ_1 and ϱ_2 with the same label via $\mathfrak{r}_1$ and $\mathfrak{r}_2$, start graph $S_1 \cdot|\cdot S_2$.

In the illustration, the background colour is used to depict the alteration of the labels. It must be thought of as a contribution to the edge and node colours.



By forming such disjoint unions of graphs and rules, parallel steps combining the individual effects on the component graphs are possible. The disjointness of the label sets prevents spurious derivation steps.

Lemma 9 *The definition of* $\cdot|\cdot$ *for morphisms establishes a bijective correspondence between the morphisms from* $G_1 \cdot|\cdot G_2$ *to* $H_1 \cdot|\cdot H_2$ *and the pairs of morphisms from* G *to* H_1 *and* G_2 *to* H_2.

Proof. Every morphism from $G_1 \cdot|\cdot G_2$ to $H_1 \cdot|\cdot H_2$ assigns the items of $(G_1)_0$ to items of $(H_1)_0$ and those of $(G_2)_1$ to items of $(H_1)_1$, independently, determining two morphisms. Likewise, every pair of morphisms from G to H_1 and G_2 to H_2 determines one from $G_1 \cdot|\cdot G_2$ to $H_1 \cdot|\cdot H_2$. □

The $\cdot|\cdot$ constructions are now leveraged to build a grammar that simulates two grammars GG_1, GG_2 simultaneously. It can perform a parallel derivation step that corresponds to one step in GG_1 and one step with the same label in GG_2, whenever these are applicable individually.

Lemma 10 (Intersection Closure) $\mathcal{L}_\$$ *is closed under intersections*

$$K, L \in \mathcal{L}_\$ \quad \textit{implies} \quad K \cap L \in \mathcal{L}_\$,$$

and the operation $\cdot|\cdot$ *provides a construction for* $(\mathcal{L} \in \{\mathcal{L}_D, \mathcal{L}_\$\})$

$$\mathcal{L}(GG_1) \cap \mathcal{L}(GG_2) = \mathcal{L}(GG_1 \cdot|\cdot GG_2).$$

Proof. $\mathcal{L}_D(GG_1) \cap \mathcal{L}_D(GG_2) = \mathcal{L}_D(GG_1 \cdot|\cdot GG_2)$, because the existence of any morphism in the combined grammar is now tantamount by Lemma 9 to the conjunction of the existence of one morphism in the first grammar *and* one in the second grammar, rule applicability and effects clearly being preserved under bijective relabelings. The same holds for the end rule, therefore also $\mathcal{L}_\$(\$, GG_1) \cap \mathcal{L}_\$(\$, GG_2) = \mathcal{L}_\$(\$, GG_1 \cdot|\cdot GG_2)$. □

Closure under intersections with regular languages follows from Lemmata 7 and 10.

Corollary 1 (Regular Intersection Closure) *$\mathcal{L}_\$$ is closed under intersections with regular languages.*

Theorem 2 (Context Free Languages) *The context free languages are properly contained in $\mathcal{L}_\$$:*

$$CF \subset \mathcal{L}_\$.$$

Proof. The context free languages are characterised as the images under letter-to-letter homomorphisms of the intersection of an extended Dyck and a regular language (Theorem 3 in [Okh12]). $\mathcal{L}_\$$ contains the extended Dyck languages and is closed under the letter-to-letter homomorphism by Theorem 1 and regular intersection by Corollary 1. The inclusion is proper by the closure under all intersections (Lemma 10), which the context free languages lack. □

We now go from the languages defined by Chomsky grammars to the derivation languages of these grammars, the Szilard languages [MS97].

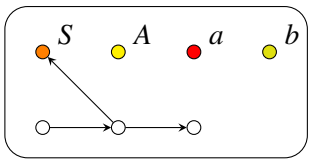
Definition 9 (Szilard Languages) The Szilard language of a Chomsky type 0 grammar G is the set of images of the terminal derivations of G under a bijective labeling of the productions.

By Mateescu and Salomaa [MS97], it is known that all Szilard languages are context sensitive.

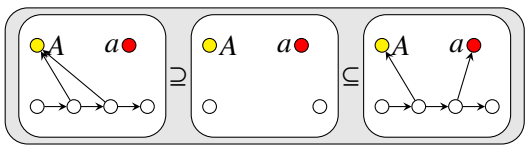
Proposition 1 (Szilard Languages) *The Szilard languages are in $\mathcal{L}_\$$:*

$$Sz \subset \mathcal{L}_\$.$$

Proof. This proposition can be seen by encoding each word composed of terminals and non-terminals as a linearly shaped graph with two dummy nodes at the ends, a chain of nodes representing the positions in the currently derived word, which are linked to alphabet nodes: for each terminal or non-terminal a one node m_a with $\lambda(m_a) = a$, the links representing the letters at each position. It is the same encoding as in the proof of Lemma 8. The start state for a grammar with non-terminals $N = \{S, A\}$ and terminals $T = \{a, b\}$ is shown in the following picture.



Each production as a rule replacing a fixed, known portion of the string by another sequence of terminals and non-terminals. A production $AA \to Aa$, for instance, gives the following rule:



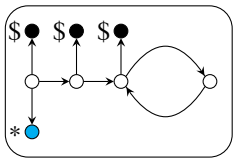
Termination, that is derivation of a terminal word, can be recognised by using as the left hand side of the end rule a graph consisting of $|N|$ nodes, one labeled with each non-terminal, and

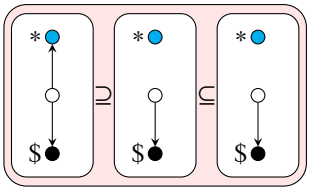
deleting these. When no more nodes use these labels, a terminal word has clearly been derived, and only then there are no more edges inhibiting the application of the end rule. Clearly, the Szilard language of any given type 0 (or type 1, type 2) grammar is obtained in this way. □

To finish the section, we demonstrate that another frequently seen non-context-free language is the derivation language of a graph grammar.

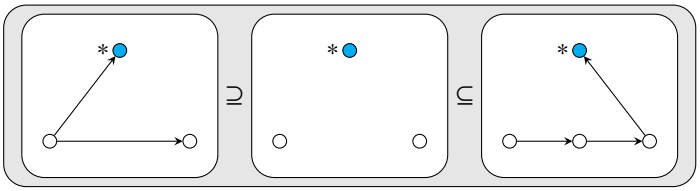
Proposition 2 (Powers of 2) *The graph grammar GG from the following Construction 6 has the derivation language* $\mathcal{L}_{\$}(GG) = \{a^{2^n} \mid n \in \mathbb{N}\}$.

Construction 6 (Powers of 2) Let GG be the grammar with rule alphabet $\{a, \$\}$, start graph S and end rule as follows (node labels are indicated, white nodes are labeled •, as are edges):





The rule $\mathfrak{r}(a)$ moves the $*$ labeled node along an edge, deletes the edge and replaces it with a new node labeled • and two edges:



Proof. By inspection, the word aa corresponds to a single possible derivation sequence and enables \$ at every step; it can be shown by induction over the length of a word that when the $*$ labeled marker has reached the cycle, it can only travel around the cycle, doubling its length every time it reaches the node linked to the \$ labeled marker. The induction hypothesis is:

Hypothesis After $n \geq 2$ derivation steps with the rule a, a unique graph is reached. In this state, the unique node m_* labeled $*$ is connected with an edge to the node which is on a cycle of length $n = 2^k + j$, $2^k > j$, and there are no outgoing edges from the cycle to • labeled nodes, and the node connected to m_* is $2^k - j \mod 2^k$ edges away in forward direction from the unique node on the cycle connected to a node labeled \$, and $2j$ edges in backward direction. Therefore if $j = 0$, then the word a^k has been read and the node connected with $*$ is also connected to a node labeled \$ and so the end rule can be applied. Otherwise, a^k does not enable the end rule.

The hypothesis is true for $n = 2$ by inspection; If it is true for $n = 2^k + j$, then also for $n+1$ because the cycle makes only one match of rule a possible, whose effect is to reduce the number of edges in the forward direction by one and increase the number of edges in the backward direction by two. The cycle structure is preserved, and the number of $*$ labeled nodes is not changed. The number of and edges connected to \$ labeled nodes are not changed. □

4 Limitations

Graph grammars provide a very powerful mechanism for the generation of string languages as derivation languages. In this section, we examine some of the limitations of this mechanism. We present an upper bound on the nondeterministic complexity for the recognition of a derivation language. We then use known results about the context free languages to show that many other questions about general derivation languages, such as the inclusion problem, are undecidable, and that applying deleting homomorphisms to them yields the recursively enumerable languages.

Proposition 3 (Space and Time Bound) *Polynomial time and $O(n \log n)$ space bounds for the problem $w \in^? \mathcal{L}_\$(GG)$ are attainable simultaneously with a nondeterministic Turing machine:*

$$\mathcal{L}_\$ \subseteq \mathrm{NP}, \quad \mathcal{L}_\$ \subseteq \mathrm{NSPACE}(n \log n)$$

Proof. A direct simulation of a derivation of length n of the grammar by a nondeterministic Turing machine is possible in $O(n \log n)$ space and polynomial time.

Each finite graph with v nodes and e edges can be represented as a list of node identifiers and labels, followed by an adjacency list carrying also the edge labels. If node identifiers are numbers not exceeding v (or at least bounded by v times a constant factor), this takes $O((v+e)\log v)$ space.

A potential match g of a rule ϱ is represented by flagging $|V_L|$ nodes and $|E_L|$ edges with an annotation: either a node or edge is not in $g(L)$, or it is to be deleted, or preserved. The preimage via g is also in the annotation. Since rules have constant size throughout the derivation, this takes only linearly more space over the bare graph representation. Rule applicability comes down to checking the dangling condition, which is possible in polynomial time and very little space by searching for unflagged edges attached to nodes flagged for deletion. Rule application involves adding at most $|V_R|$ new nodes (precisely $c_\varrho = |V_R| - |V_L|$ new nodes) and $|E_R|$ new edges, which may increase the space needed for node identifiers: therefore the Turing machine first deletes the entries for the obsolete nodes and edges in polynomial time and no excess space, then calls a routine to increase the space allocated per node identifier to $\lceil \log v + c_\varrho \rceil$, for example in linear space and quadratic time by using a work tape to perform the operation while copying. Then it adds the c_ϱ new nodes (for which space has been left blank at the end of the node list) and $|E_R| - |E_L|$ new edges and clears all flags. Looking up the source and target node labels for the new edges can be done in polynomial time by going through the annotations.

The number of nodes and edges involved (added) in a derivation step is constant. So during and after the application of n rules, the length of the representation is increased to at most $O((v+e)\log v)$ with v and e linear in n. □

Undecidability of the termination of graph rewriting is already well known (see Plump [Plu98]). Indeed, a graph grammar can simulate the workings of a Turing machine, but only with a derivation sequence of length proportional to the length of the computation. If unlabeled steps are allowed, then the whole set of recursively enumerable languages is obtained:

Closure under arbitrary homomorphisms, which may also delete letters, is an operation on language classes often considered in the literature on formal languages, sometimes denoted by $\widehat{\mathcal{H}}$.

Lemma 11 (The Effect of Deleting Homomorphisms) *The closure of the derivation languages $\mathcal{L}_\$$ under deleting homomorphisms is the family of recursively enumerable languages:*

$$RE = \widehat{\mathcal{H}}(\mathcal{L}_\$).$$

Proof. We slightly modify the usual proof [HU79, Kud04] of the fact that the emptiness problem for the intersection of two context free languages is undecidable, by simulating the elementary steps of any given Turing machine with two context free languages, one producing sequences of correct steps at even positions $(2i, 2i+1)$ and the other one doing so at odd positions $(2i+1, 2i+2)$. We refer to the literature and follow the proof in [HU79], modifying it to write out the input word in a special alphabet at the beginning of the computation. Upon reaching a configuration where the Turing machine halts in an accepting state, the derivation in the simulating grammar terminates with \$. Application of the deleting homomorphism to remove the whole subsequent calculation, however long it may be, then results in $L \in \hat{\mathcal{H}}(\mathcal{L}_\$)$. On the other hand, all images under deleting homomorphisms of $\mathcal{L}_\$$ languages are recursively enumerable: simulate the given graph grammar breadth-first, output the images (if any) under the homomorphism of the labels of the rules used, and accept once there is a valid match for the termination rule. □

Proposition 4 (Undecidability) *The following questions are undecidable for $\mathcal{L}_\$$ languages given by grammars GG, GG′ in general: totality, equivalence, inclusion, regularity, emptiness.*

Proof. Decidability of the word problem follows from Proposition 3. All questions undecidable [Kud04] for context free languages (given as push-down automata or equivalently as context free grammars) are undecidable for $\mathcal{L}_\$$ languages. Because of Lemma 10 there is furthermore an effective way of constructing a grammar outputting the intersection of two context free languages, but it is undecidable whether the intersection of two context free languages is empty. □

The following table recapitulates the decidability results.

Question	Formal Description	Decidable?
Word problem	$w \overset{?}{\in} \mathcal{L}_\$(GG)$	+
Totality	$\mathcal{L}_\$(GG) \overset{?}{=} T^*$	–
Equivalence	$\mathcal{L}_\$(GG) \overset{?}{=} \mathcal{L}_\$(GG')$	–
Inclusion	$\mathcal{L}_\$(GG) \overset{?}{\subseteq} \mathcal{L}_\$(GG')$	–
Regularity	$\mathcal{L}_\$(GG) \overset{?}{\subseteq} REG$	–
Emptiness	$\mathcal{L}_\$(GG) \overset{?}{=} \emptyset$	–

5 Conclusion and Outlook

These are the relations of $\mathcal{L}_\$$ and the Chomsky families and the Szilard languages:

$$\begin{array}{ccccccc} REG & \subset & CF & \subset & CS & \subset & RE \\ \cap & & \cap & & \cup & & \| \\ \mathcal{H}_1(\mathcal{L}_\$) & = & \mathcal{L}_\$ & \supset & Sz & \subset & \widehat{\mathcal{H}}(\mathcal{L}_\$) \end{array}$$

For easy reference, we tabulate the main results of the paper and their dependencies.

Reference	Result	Dependencies
Theorem 1	$\mathcal{L}_\$ = \mathcal{H}_1(\mathcal{L}_\$)$	
Lemma 7	$REG \subseteq \mathcal{L}_\$$	Theorem 1
Lemma 8	$\widehat{D}_{m,n} \in \mathcal{L}_\$$	-
Lemma 10	$K, L \in \mathcal{L}_\$ \Rightarrow K \cap L \in \mathcal{L}_\$$	-
Theorem 2	$CF \subset \mathcal{L}_\$$	Theorem 1, Lemmata 7, 8, 10
Proposition 1	$Sz \subset \mathcal{L}_\$$	-
Proposition 6	$\{a^{2^n} \mid n \in \mathbb{N}\} \in \mathcal{L}_\$$	-
Proposition 3	$\mathcal{L}_\$ \subseteq \mathrm{NP}, \mathrm{NSPACE}(n \log n)$	-
Lemma 11	$RE = \widehat{\mathcal{H}}(\mathcal{L}_\$)$	Theorem 2, Lemma 10
Proposition 4	Decidability Issues	Theorem 2, Lemma 10, Proposition 3

Further work in progress suggests a large array of closure properties, and the inclusion of several well studied supersets of the context free languages in $\mathcal{L}_\$$. The comparison with Petri net languages with various acceptance conditions has also been an object of our attention, since Petri nets are an important subclass of graph grammars.

Several questions remain open. First of all, we do not know of a criterion like a pumping lemma to place a language outside of $\mathcal{L}_\$$. Also, answers to these questions are still missing:

$L \in \mathcal{L}_\$ \Rightarrow (\Sigma^* - L) \in \mathcal{L}_\$$?	is $\mathcal{L}_\$$ closed under complementation?
$\mathcal{L}_\$ \subseteq CS$?	are all derivation languages context sensitive?
$CS \subseteq \mathcal{L}_\$$?	are all context sensitive languages derivation languages?

The second question would be answered in the affirmative if the space requirements for recognition from Proposition 3 could be reduced from $O(n \log n)$ to $O(n)$. The statement of the third question should not be expected to hold, since then there would be a graph grammar whose derivation language is the PSPACE complete language of true quantified Boolean formulae [HU79], highly unlikely in the light of Proposition 3. If the language $L_p \in CS$ of all words from $\{a\}^*$ of prime length could be shown not to be a derivation language, the first and third questions would have a negative answer, since its complement $\{a\}^* - L_p$ is indeed a derivation language (although we have not shown it here).

Acknowledgements The author gratefully acknowledges the anonymous reviewers, whose comments helped greatly towards improving this paper. Many thanks to Annegret Habel for providing valuable criticism at several readings.

Bibliography

[Ber79] J. Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.

[Cor95] A. Corradini. Concurrent Computing: from Petri Nets to Graph Grammars. Volume 2, pp. 56–70. Elsevier, 1995.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

[Ehr79] H. Ehrig. Introduction to the Algebraic Theory of Graph Grammars (a Survey). LNCS 73, pp. 1–69. Springer, 1979.

[HU79] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[Hö75] M. Höpner. Eine Charakterisierung der Szilardsprachen und ihre Verwendung als Steuersprachen. In *Gl-4.Jahrestagung*. LNCS 26, pp. 113–121. Springer, 1975.

[Jan87] M. Jantzen. Language Theory of Petri Nets. LNCS 254, pp. 397–412. Springer, 1987.

[KKK06] H.-J. Kreowski, R. Klempien-Hinrichs, S. Kuske. Some Essentials of Graph Transformation. In *Recent Advances in Formal Languages and Applications*. Studies in Computational Intelligence 25, pp. 229–254. Springer, 2006.

[Kud04] M. Kudlek. Context-Free Languages. In *Formal Languages and Applications*. Volume 148, pp. 97–116. Springer, 2004.

[Mäk98] E. Mäkinen. A Bibliography on Szilard Languages. *Bulletin of the EATCS* 65:143–148, 1998.

[MS97] A. Mateescu, A. Salomaa. Aspects of Classical Language Theory. In Rozenberg and Salomaa (eds.), *Handbook of formal languages, vol. 1: word, language, grammar*. Chapter 7. Springer, 1997.

[Okh12] A. Okhotin. Non-erasing Variants of the Chomsky–Schützenberger Theorem. In *Developments in Language Theory*. LNCS 7410, pp. 121–129. Springer, 2012.

[Plu98] D. Plump. Termination of Graph Rewriting is Undecidable. Volume 33(2), pp. 201–209. IOS Press, 1998.

[Roz97] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.