A New Notion of Regularity: Finite State Automata Accepting Graphs

Yvo Ad Meeres
Department of Theoretical Computer Science
University of Bremen
Bremen, Germany
yvo.meeres@mailbox.org

Analogous to regular string and tree languages, regular languages of directed acyclic graphs (DAGs) are defined in the literature. Although called regular, those DAG-languages are more powerful and, consequently, standard problems have a higher complexity than in the string case. Top-down as well as bottom-up deterministic DAG languages are subclasses of the regular DAG languages. We refine this hierarchy by providing a weaker subclass of the deterministic DAG languages. For a DAG grammar generating a language in this new DAG language class, or, equivalently, a DAG-automaton recognizing it, a classical deterministic finite state automaton (DFA) can be constructed. As the main result, we provide a characterization of this class.

The motivation behind this is the transfer of techniques for regular string languages to graphs. Trivially, our restricted DAG language class is closed under union and intersection. This permits the application of minimization and hyper-minimization algorithms known for DFAs. This alternative notion of regularity coins at the existence of a DFA for recognizing a DAG language.

1 Introduction

Many research fields either struggle with the complexity of processing graphs – for example fields like high performance computing [19] or neurocomputing [6], to mention just a few – or by encoding their graph problems as strings, see e.g. [16, 11]. The well-researched class of regular string languages, recognized by finite state automata (FSAs), exhibits a fruitful balance between expressiveness and efficiency concerning standard algorithmic problems. The problem is, that these algorithms are only applicable to strings. One approach would be to provide efficient graph algorithms for specific problems, as in [25] for the membership problem circumventing Braess's Paradox [7] or in [4] providing a faster algorithm for the very specific problem of the maximum independent set on interval filament graphs. Instead of the cumbersome approach to tackle all these specific problems one by one, our meta-approach suggests porting all efficient algorithms known for string processing to graph processing in one sweep. To port a wide range of well-known efficient algorithms (based on FSAs) from strings to graphs, this article introduces FSAs recognizing sets of directed acyclic graphs (DAGs) instead of sets of strings. Such sets are called DAG languages. We consider *vertex-labeled DAGs* with unlabeled edges but label those edges for accepting a DAG. A classical FSA accepts a string by reading it symbol by symbol. Our proposed FSA accepts a DAG by reading top-down vertex by vertex instead. A symbol read by the automaton encodes a whole vertex, consisting of its vertex label and its ordered and labeled in- and outgoing edges. While reading the vertices top-down, the outgoing edges in a DAG are labeled according to the automaton's specification while the ingoing edges, labeled beforehand, have to match. The FSA's states keep track of those ingoing edge labels whose target vertices are not yet read, thus whose outgoing edges are unlabeled. The class of DAG languages accepted by such an FSA is a proper subset of the top-down

Giovanni Pighizzini, Florin Manea (Eds.): NCMA 2024
EPTCS 407, 2024, pp. 5–26, doi:10.4204/EPTCS.407.2

© Yvo Ad Meeres
This work is licensed under the
Creative Commons Attribution License.

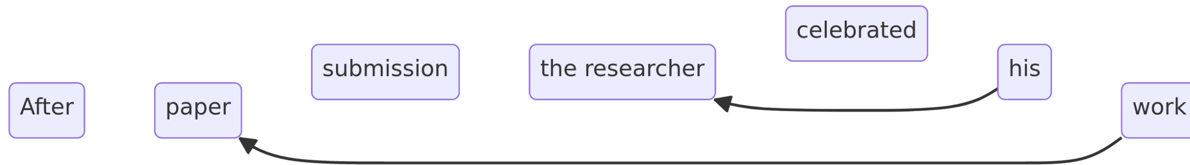


Figure 1: Classical NLP parsing is blind to coreferences within sentences, since trees cannot represent these edges within the parse tree. Graphs, on the contrary, are capable of showing coreferences between e.g. words of parsed sentences. For the above sentence, its parse trees could neither model the obvious possessive relation between the possessive pronoun *his* and *the researcher* nor the semantic kind of equivalence relation requiring world knowledge between the *paper* and the *work*. But, a semantic graph like an `AMR` DAG [29] could. The capabilities of semantic graphs are illustrated in [13] as well as for a complex sentence in [12] by means of the representation of a sentence as an `AMR` DAG.

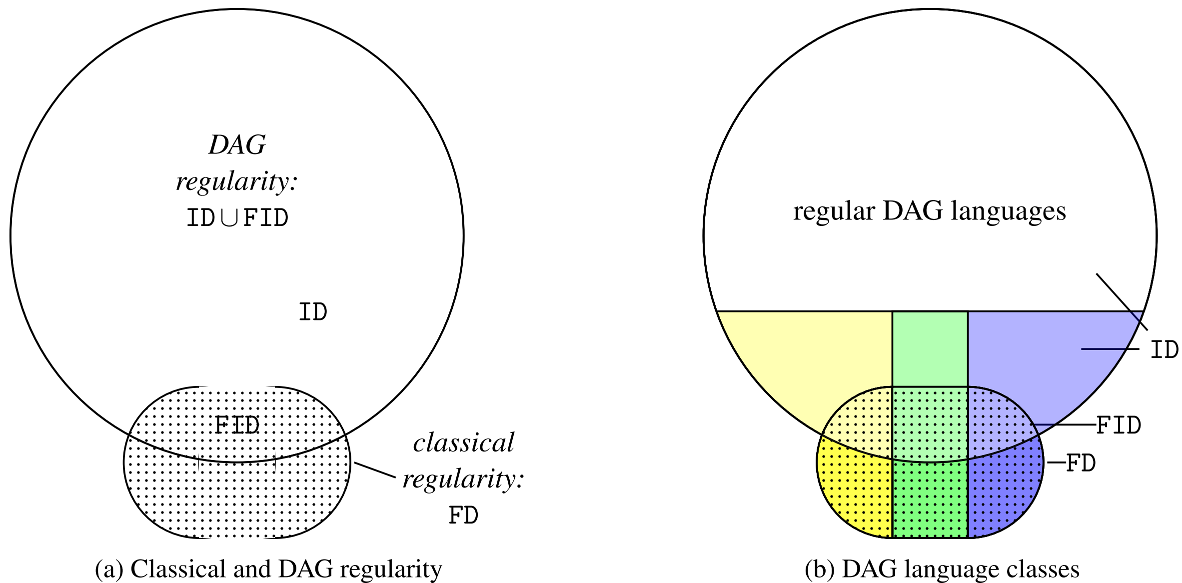
deterministic regular DAG languages defined in the literature [5]. This class, in turn, is a proper subset of the regular DAG languages [5].

In literature, the notion of regularity concerning DAG languages differs from that applied to string languages. Regular DAG automata recognize regular DAG languages [5]. These automata are one of the formalisms [12] proposed in the literature to model semantics by using Abstract Meaning Representation (`AMR`) [3]. DAG automata were originally introduced by Kamimura and Slutzki [21, 22]. A promising alternative formalism, not considered in this paper, is the hyperedge replacement graph grammar [20]. Classical natural language parsing turns a sentence into a parse tree while semantic parsing, such as `AMR` parsers [9, 10, 30], can model coreferences between e.g. the words of a sentence. Fig. 1 shows two such relations which turn a parse tree into a DAG. Such semantic relations, expressible in `AMR`, specify that the work was conducted by the mentioned researcher (*researcher* $\leftarrow$ *his*) and writing the paper is the researcher's daily work (*work* $\rightarrow$ *paper*). Semantic parsing provides vital contributions to improve natural language processing (NLP). The anecdotes about AI chat bots inventing facts feed wishes for NLP improvements by ensuring semantic consistency. This consistency is desirable also for sophisticated spell and grammar checking and machine translations.

The membership problem for regular DAG automata surprisingly being NP-complete [8, 18], the uniform and even the non-uniform one[1], finding strategies for identifying efficient semantic parsing algorithms remains the core problem. By determinizing, either top-down or bottom-up, the membership problem becomes tractable. However, even with restrictions like determinism or planarity [21, 32], parsing problems can easily become too complex. For instance, Vasiljeva et al. were surprised that for certain probabilistic DAG automata non-trivial probability distributions are necessary to assign weights [32]. Since the notions of regularity differ for the string and DAG case, we propose the *new notion of regularity*: use the string case regularity for DAGs in order to obtain better algorithmic properties. The regularity notion for DAGs, presented in literature, seems to be too powerful to provide efficient algorithms. Viewing DAG languages only then as regular when they can be recognized by an FSA, provides deep insight into the structural properties of DAG languages.

Although the mildly context sensitive upper bound for natural language parsing classifies parsing as lying between context-free and context-sensitive formalisms, finite state descriptions of languages are of

[1]The uniform membership problem asks for a given automaton and a given graph whether the graph is an element of the given language; the non-uniform membership problem asks for a fixed automaton and a given graph whether the automaton accepts the graph, making the membership problem potentially easier.



(a) Classical and DAG regularity

(b) DAG language classes

Figure 2: *Overview over the language classes*
In both Venn diagrams, the circle denotes the the regular DAG languages whereas the oval denotes the language class FD. *The intersection between the two is the class* FID *which is both closed under edge swap as well as under DFA-construction. The dotted part, the oval, corresponds to* FD. *The non-dotted part corresponds to* ID.
(a) Classically, the term regularity *refers to FSAs and thus to string languages. This does not match the notion of regularity for DAG languages. The two notions match only for languages in* FID.
(b) Top-down determinism and bottom-up determinism are colored in yellow and blue. Consequently, green stands for languages which are both top-down as well as bottom-up deterministic. In the right Venn diagram, whereas all colored fields are deterministic, the nondeterministic part corresponds to the white part The class ID *comprises those regular DAG languages which are not in* FID *(and consequently not in* FD*), and which are either (top-down / bottom-up) deterministic or non-deterministic.*

major importance [1]. For the sake of efficiency, this field often seeks to digest also mildly context sensitive structures with finite state methods [27, 23, 28], an approach conceivable also for DAG digestion. The conference *Finite State Methods in Natural Language Processing* (FSMNLP) concentrates on this lowest level of the Chomsky hierarchy. Many NLP tools, like `apertium`, `HFST` or `GiellaLT` [17, 31, 26] operate with finite state descriptions.

All this said, our contributions, see them illustrated in Fig. 2, can be stated as

- the idea of using a classical FSA to recognize a DAG language
- the notion of a *meta-state*, a multiset of edge labels, serving as the states of the FSA
- separation of the regular top-down deterministic DAG languages into those recognizable (FID) and those not recognizable (ID) via an FSA by means of the meta-state technique
- restricting a DAG automaton by meta-states, resulting in the class FD comprising FID but not being a subset of the regular DAG languages
- characterization of the newly defined classes (*main result*).

Providing an FSA for a DAG language immediately opens up a wide range of results for DAGs formerly applicable only to strings. The folklore algorithm of FSA minimization can be applied, just as algorithms for lossy FSA compression, called hyper-minimization [2, 24], where *hyper* stands for the tolerance of finitely many errors. Morphological transducers already prove hyper-minimizations being useful for NLP [14].

Also from a structural point of view a deeper understanding of DAG language classes and a suitable overall hierarchy would be a beautiful result for theoretical computer science. As mentioned in the beginning, many research fields require efficient graph algorithms and therefore could potentially profit from these very limited DAG languages since they are parsable as efficiently as regular string languages. Even though their expressiveness is quite limited, implying that we cannot encode arbitrary graph languages, for a variety of important problems, the limited expressiveness will suffice, and algorithms can be ported directly from the string case.

2 Preliminaries

The set of non-negative integers is denoted by $\mathbb{N}$. For $n \in \mathbb{N}$ we define $[n] = \{1,\dots,n\}$. For a set A, we denote its cardinality by $|A|$. A finite set A is called an *alphabet*, an element $a \in A$ is a *symbol*, a *string* is the concatenation of symbols, the set of all finite strings over A is denoted by A^* and a, not necessarily proper, subset of A^* is called a *language*. The empty string of length 0 is denoted by λ. The length of a string $w \in A^*$ is denoted by $|w|$ and $[w]$ denotes the smallest set A such that $w \in A^*$. The concatenation of two strings a and b is written as its juxtaposition ab. For a string $w_1w_2\dots w_i\dots w_n$ of length n over A, the *position* of a symbol $w_i \in A$ is $i \in [n]$. The canonical extensions of a function $f\colon X \to Y$ to the power set of X and to X^* are denoted by f as well. Thus, $f(\{x_1,\dots,x_n\}) = \{f(x_1),\dots,f(x_n)\}$ and $f(x_1\cdots x_n) = f(x_1)\cdots f(x_n)$ for all $x_1,\dots,x_n \in X$. For a set Y a (locally finite) multiset over Y is a function $\mu\colon Y \to \mathbb{N}$. For brevity, we give a specific multiset by a string notation $y_1^{\mu(y_1)}\cdots y_n^{\mu(y_n)}$ for $y_1\dots y_n \in Y$. The size of μ is $|\mu| = \sum_{y\in Y}\mu(y)$. (Formally, $|\mu| = \infty$ if $\mu(y) \geq 1$ for infinitely many y, but this case will not be relevant for this paper, i.e., all multisets appearing here will be finite.) We denote the set of all multisets over Y by $\mathbb{N}^Y$. For a function $f\colon X \to Y$, we let $f_M\colon 2^X \to \mathbb{N}^Y$ be the mapping such that, for every $X' \subseteq X$, $f_M(X')$ is the multiset of images of elements of X' under f. Thus, formally, $f_M(X')(y) = |\{x \in X' \mid f(x) = y\}|$ for every $y \in Y$.

This article studies languages of vertex-labeled, directed multigraphs without loops and with ordered unlabeled edges (called *graphs*, see Def. 2.1) which are acyclic (called *DAGs*, see Def. 2.4). Edges will be labeled only temporarily by a grammar (c.f. Def. 2.5), its equivalent automaton or classical finite state automaton (see Section 7).

Definition 2.1 (Graph). *A* graph *over* Γ *is a tuple* $G = (V,E,\ell,in,out)$ *with* Γ, V *and* E *being disjoint finite sets, the sets of* vertex labels, vertices *and* edges, *respectively. The vertices are labeled by* $\ell\colon V \to \Gamma$. *For an* edge $e \in E$ *between the vertices* $(v,w) \in V \times V$, directed *from* v *to* w, *with* $v \neq w$, *the* source v *is referenced by* $src(e)$ *and the* target w *by* $tar(e)$. *By* $in,out\colon V \to E^*$ *we assign to each vertex* $v \in V$ *its* incoming *and* outgoing *edges such that* $src(e) = v \Leftrightarrow e \in [out(v)]$ *and* $tar(e) = v \Leftrightarrow e \in [in(v)]$. *These edges are ordered as specified by the strings* $in(v)$ *and* $out(v)$. *The* empty graph $\varnothing$ *is the graph whose set of vertices is empty. A vertex is called a* root *or a* leaf *if* $in(v)$ *or* $out(v)$ *are empty, respectively. The disjoint union of graphs, meaning disjoint sets of vertices and edges, is denoted by the operator* &.

Definition 2.2 (Path). *A* path *in a graph* $G = (V,E,\ell,in,out)$ *is a nonempty sequence of edges* $e_1,\dots,e_n$, $e_i \in E$ *for* $i \in [n]$, *yielding a unique alternating sequence* $v_0e_1v_1\cdots e_nv_n$ *with vertices* $v_0,\dots,v_n \in V$ *such that* $\{src(e_i), tar(e_i)\} = \{v_{i-1}, v_i\}$ *for all* $i \in [n]$. *Such a path is a* cycle *if* $v_0 = v_n$. *A* path between s *and*

t is a path with $s \in \{v_0, e_1\}$ and $t \in \{v_n, e_n\}$. A path is directed *if for $i \in [n]$ either $\forall i : tar(e_i) = v_i$ or $\forall i : tar(e_i) = v_{i-1}$ and we call it a* path from s to t *if it is a directed path between s and t with $\forall i : tar(e_i) = v_i$. The* length *of a path is the number of its edges, written as $|e_1, \dots, e_n| = n$. The graph G is said to be* connected *if there is a path between each pair of vertices. In a path specification, we may denote the vertices and edges $a \in V \cup E$ by their respective label $\ell(a)$.*

Definition 2.3 (Chord Path)**.** *A* chord path *of a cycle shares its end vertices with its corresponding cycle, but none of its edges [33, 15]. Given a graph $G = (V, E, \ell, in, out)$, let the path $c = e_1, \dots, e_n$ with $e_i \in E$ for $i \in [n]$, be a cycle yielding $v_n e_1 \cdots e_n v_n$. A* chord path *of the cycle c is a path $e'_1, \dots, e'_m$ with $e'_j \in E$ for $j \in [m]$ yielding $v'_0 e'_1 v'_1 \cdots e'_n v'_m$ with vertices $v'_0, \dots, v'_m \in V$ such that $v'_0 \neq v'_m$ and $\{v'_0, v'_m\} = \{v_0, \dots, v_n\} \cap \{v'_0, \dots, v'_m\}$ but $\{e_i \mid i \in [n]\}$ and $\{e'_j \mid j \in [m]\}$ being disjoint sets.*

Definition 2.4 (DAG, complete DAG, prefix-DAG)**.** *A* directed acyclic graph (over Γ)*, abbreviated as* DAG*, is a graph over Γ that does not contain any directed cycle. The set of all connected and nonempty DAGs over Σ is denoted by $\mathscr{D}_\Sigma$. A connected DAG $G = (V, E, \ell, in, out)$ is called a* string DAG *iff $|in(v)| \leq 1$ and $|out(v)| \leq 1$ for all vertices $v \in V$. Throughout this paper, Σ and N being disjoint sets, Σ will denote an* alphabet of terminals*, namely, the* vertex labels*, whereas N is our* alphabet of nonterminals *used for labeling vertices and edges temporarily. We call a DAG over Σ a* complete DAG*. A DAG over $\Sigma \cup N$ is called a* prefix-DAG*, a* proper prefix-DAG *if at least one vertex is labeled by a nonterminal $n \in N$.*

Definition 2.5 (Regular DAG grammar, $L(\mathscr{G})$, $L(\mathscr{G})^{\&}$ [13])**.** *A* regular DAG grammar[2] *is a triple $\mathscr{G} = (N, \Sigma, R)$. Each* rule *$r \in R$ is of the form $\alpha \twoheadrightarrow\!\!(\sigma)\!\!\twoheadrightarrow \beta$ where $\sigma \in \Sigma$ while the* head *α and the* tail *β are elements of N^*. For the prefix DAGs G and G', there exists a* derivation step *$G \Rightarrow_r G'$ using a rule r if G contains pairwise distinct vertices $v_1, \dots, v_k$ such that $\ell(v_1 \cdots v_k) = \alpha$. In that case, G' is obtained from G by*

- *adding the vertex v with its label $\ell(v) = \sigma$*
- *by letting the edges, formerly pointing to $v_1, \dots, v_k$, now point to v, thus $tar(in(v_i)) = v$,*
- *deleting the temporary vertices $v_1, \dots, v_k$ and, in turn,*
- *adding the temporary vertices $w_1, \dots, w_j$ labeled by their nonterminals $\ell(w_1 \cdots w_j) = \beta$*
- *by connecting them to the graph with the edges $(v, w_1), \dots (v, w_j)$.*

A derivation *is a sequence of prefix DAGs[3] $G_0 \Rightarrow_{r_1} G_1 \Rightarrow_{r_2} \cdots \Rightarrow_{r_n} G_n$, also denoted by $G_0 \Rightarrow_{r_1 \cdots r_n} G_n$. The set of all these G_n that are complete is denoted by $L(\mathscr{G})^{\&}$. The* DAG language generated by *$\mathscr{G}$ is $L(\mathscr{G}) = \{G \in \mathscr{D}_\Sigma \mid \varnothing \Rightarrow_R^* G\}$, the set of connected and complete DAGs which the grammar can derive, where $\Rightarrow_R^*$ denotes the transitive and reflexive closure of $\Rightarrow_R = \bigcup_{r \in R} \Rightarrow_r$. As usual, a rule is said to be* useless *if none of the derivations for DAGs in $L(\mathscr{G})$ comprises this rule and* useful *if it does. The DAG grammar $\mathscr{G}$ is* deterministic *if, for every pair $\alpha \twoheadrightarrow\!\!(\sigma)\!\!\twoheadrightarrow$ in $N^* \times \Sigma$, there exists at most one $\beta \in N^*$ such that $(\alpha \twoheadrightarrow\!\!(\sigma)\!\!\twoheadrightarrow \beta) \in R$. The DAG language generated by a deterministic DAG grammar, as well as a DAG automaton recognizing it, is called* top-down deterministic*. By reversing the orientation of the edges, we obtain its* dual language*. A language $L(\mathscr{G})$ of a deterministic DAG grammar is called* bottom-up deterministic *if its dual language is generated by $\mathscr{G}$.*

The class RDL of regular DAG languages *consists of all DAG languages generated by a regular DAG grammar (equivalently recognized by a regular DAG automaton). The class RDL^{det} of* regular

[2] Since regular DAG grammars are equivalent to regular DAG automata, an illustrative example of how the DAGs are handeled can be found in [12].

[3] We extend the notation $\Rightarrow_{r_1 \cdots r_n}$ to $\Rightarrow_E$, where E is an (extended) regular expression over rules: if $L(E)$ denotes the language of sequences of rules denoted by E, then $\Rightarrow_E = \bigcup_{r_1 \cdots r_n \in L(E)} \Rightarrow_{r_1 \cdots r_n}$.

deterministic DAG languages *consists of all DAG languages and dual languages generated by regular deterministic DAG grammars (equivalently recognized by a regular deterministic DAG automaton).*

In a derivation of a DAG G, at the time a new edge e is generated, its newly created target vertex v is labeled by a nonterminal, say q. At that time v "dangles" at the end of e without further incoming or outgoing edges. Later rule applications will take v, merge it with other vertices and label the resulting vertex with its final symbol taken from Σ according to the rule used. The edge e, however, remains untouched. We may represent a derivation of G up to reordering of derivation steps by the DAG G itself together with a labeling of edges by nonterminals. Then, e would be labeled with q. We call this the derivation DAG of G.

Definition 2.6 (Derivation DAG, $\lfloor D \rfloor$)**.** *Let $G_0 \Rightarrow_{r_1 \cdots r_n} G_n$ with $r_1, \ldots r_n \in R$ be a derivation of a DAG $G_n = G = (V, E, \ell, in, out)$ generated by a DAG grammar $\mathscr{G} = (N, \Sigma, R)$. Then, the corresponding* derivation DAG *of G is the tuple $D = (V, E, \ell, in, out)$, where $\ell \colon E \cup V \to \Sigma \cup N$ is extended to edges by: for every edge $e \in E$, $\ell(e)$ is the unique nonterminal $q \in N$ such that, for some $i \in [n]$, e is an edge of G_i with $\ell(tar_{G_i}(e)) = q$. Let $\lfloor D \rfloor$ denote the DAG G_n, obtained from D by restricting ℓ to V, denoted by $\ell|_V$.*

A derivation DAG is not necessarily connected, thus $\lfloor D \rfloor \in L(\mathscr{G})^{\&}$ if $\varnothing \Rightarrow_R^* D$ but $\lfloor D \rfloor \in L(\mathscr{G})$ only if D, or equivalently $\lfloor D \rfloor$, is connected. It should be noted that the set of all derivation DAGs of (DAGs in) $L(\mathscr{G})$ is easily characterized: For every such derivation DAG $D = (V, E, \ell, in, out)$ the DAG $G = (V, E, \ell|_V, in, out)$ is an element of $\mathscr{D}_\Sigma$) and for every vertex $v \in V$ there is a rule $\alpha \twoheadrightarrow \textcircled{\sigma} \twoheadrightarrow \beta$ such that $\alpha = \ell(in(v))$ and $\beta = \ell(out(v))$. Thus, the derivation DAGs of $\mathscr{G}$ coincide with the runs of the DAG automata in [5], and $L(\mathscr{G})$ is the set of all DAGs $\lfloor D \rfloor$ such that D is a derivation DAG of a DAG generated by $\mathscr{G}$. Moreover, a regular DAG grammar $\mathscr{G}$ without useless rules is deterministic if and only if every DAG in $L(\mathscr{G})$ has exactly one derivation DAG.

Definition 2.7 (Rule Path and Cycle)**.** Marking *a symbol q (at a position $i \in [n]$ of a string of length n) by a* mark $\bar{\ }$ *means replacing it with $\bar{q}$. We mark rules with the* entry mark $\check{\ }$ *and the* exit mark $\hat{\ }$*; if it is not specified which of those two marks is used, we use $\bar{\ }$. A* marked rule *$\bar{r} = (\bar{\alpha} \twoheadrightarrow \textcircled{\sigma} \twoheadrightarrow \bar{\beta})$ is obtained from a rule $r = (\alpha \twoheadrightarrow \textcircled{\sigma} \twoheadrightarrow \beta)$ by marking two nonterminals at two distinct positions $i, j \in [|\alpha\beta|]$ in $\alpha\beta$, one with the entry, one with the exit mark; in a* weakly *marked rule only at one position with either entry or exit mark. Such a* marked nonterminal *is referenced by its tuple $(\bar{q}, \bar{r})$ where $\bar{r} = (\bar{\alpha} \twoheadrightarrow \textcircled{\sigma} \twoheadrightarrow \bar{\beta})$ is the (weakly) marked rule in which q is replaced with $\bar{q}$. A rule pair $(\bar{r}_i, \bar{r}_j)$ for the (weakly) marked rules $\bar{r}_i$ and $\bar{r}_j$* agrees on *the marked nonterminals $(\hat{q}, \bar{r}_i)$ and $(\check{q}, \bar{r}_j)$ if q is marked once in a head and once in a tail in order to obtain $(\hat{q}, \bar{r}_i)$ and $(\check{q}, \bar{r}_j)$. Two weakly marked rules always agree – regardless of their marked nonterminals. A* rule sequence *is a nonempty sequence of (weakly) marked rules $\bar{r}_1, \ldots, \bar{r}_n$ for which all rule pairs $(r_i, r_{i+1 \bmod n})$ with $i \in [n]$ agree and every marked nonterminal in the rule sequence is exactly once agreed on. A rule sequence of marked rules $\bar{r}_1, \ldots, \bar{r}_n$ is a* rule cycle*, with $\bar{r}_1$ and $\bar{r}_n$ being weakly marked it is a* rule path*. A rule path* between *s and t is a rule path with marked nonterminals $\bar{q}_1, \ldots, \bar{q}_{n-1}$ which yields a path $\sigma_1, q_1, \ldots, q_{n-1}, \sigma_n$ in a derivation DAG between s and t for $s \in \{q_1, \sigma_1\}$ and $t \in \{q_n, \sigma_n\}$.*

Figures 5 and 6a show examples of rule sequences. Observe that these definitions permit two types of rule sequences. A rule sequence yields a directed path in a graph if and only if both mark types, exit and entry, do not occur both in heads and tails. We call this a *directed* rule sequence. If heads (and consequently tails) comprise both types of marks, the resulting path in the graph will be undirected. This is called an *undirected* rule sequence.

Observation 2.8 (Yielding Cycle)**.** *Obviously, a directed rule cycle cannot yield a directed cycle in a DAG, since DAGs are acyclic. Therefore, only undirected rule cycles can yield a cycle in a DAG. Directed rule cycles yield paths only, no cycles, in a DAG, undirected ones yield both paths and undirected cycles.*

Theorem 2.9 (Infinite Language [5]). *The DAG language generated by a DAG grammar $\mathscr{G} = (N, \Sigma, R)$ without useless rules is infinite iff R contains a rule cycle.*

Definition 2.10 (Swap). *Let $G = (V, E, \ell, in, out)$ be a DAG. Two edges $e_0, e_1 \in E$ are* independent *if there is no directed path between e_0 and e_1. In this case, the* edge swap of e_0 and e_1 *is defined and yields the DAG $G[e_0 \bowtie e_1] = (V, E, \ell, swap \circ in, out)$ given by the bijection $swap\colon E \to E$ defined as $swap(e_i) = e_{1-i}$ for $i \in \{0,1\}$ and $swap(e) = e$ for $e \notin \{e_0, e_1\}$. For $k \in \mathbb{N}$, let $G_0, G_1, \ldots, G_k$ be $k+1$ disjoint isomorphic copies of G, and for $i \in \{0,1,\ldots,k\}$, let e_i and e'_i be the copies of e and e' in G_i, respectively. Then the graph $G(e \bowtie e')^k$ for $k \in \mathbb{N}$ is defined as*

$$G(e \bowtie e')^0 = G_0 \quad \textit{and} \quad G(e \bowtie e')^k = (G(e \bowtie e')^{k-1} \,\&\, G_k)[e'_{k-1} \bowtie e_k].$$

Swapping two edges means that the tips of the arrows, the edge targets, are exchanged with one another. This swapping operation is, of course only allowed, if the result of the swap is still a DAG. Swapping edges yielding a directed cycle is not defined. The operation is central for regular DAG languages since, after swapping two edges in a DAG that have the same label in one of its derivation DAGs, the swapped result is still part of the language.

Lemma 2.11 (Swap Preserves Generation [5]). *Let $\mathscr{G} = (N, \Sigma, R)$ be a regular DAG grammar and $D = (V, E, \ell, in, out)$ a derivation DAG with $\lfloor D \rfloor \in L(\mathscr{G})^{\&}$. Then, if $\ell(e_0) = \ell(e_1)$ for $e_0, e_1 \in E$, the edge swap of e_0 and e_1 in D, in case it is defined, yields a DAG generated by $\mathscr{G}$, thus $\lfloor D[e_0 \bowtie e_1] \rfloor \in L(\mathscr{G})^{\&}$.*

Deterministic DAG grammars as defined above are equivalent to the top-down deterministic DAG automata in [5] and every derivation DAG of a grammar corresponds one-to-one to a run of the corresponding DAG automaton. Therefore, all results for top-down deterministic DAG automata carry over to deterministic DAG grammars. Refer to [5] for the notation concerning DAG automata. In particular, this holds for the following theorem, in which a regular DAG grammar is called minimal if it does not contain useless rules and there is no regular DAG grammar with fewer nonterminals generating the same language.

Theorem 2.12 (Minimal Grammar [5]). *For every deterministic DAG grammar $\mathscr{G}$, a minimal deterministic DAG grammar $\mathscr{G}'$ with $L(\mathscr{G}') = L(\mathscr{G})$ can be computed in polynomial time. This DAG grammar is unique: every minimal deterministic DAG grammar that accepts $L(\mathscr{G})$ is identical to $\mathscr{G}'$ up to a bijective renaming of its nonterminals.*

3 Meta-State

Classical finite state automata recognize string languages by a finite memory, its set of states. In every step the automaton memorizes exactly one state. On the contrary, while generating a DAG, every derivation step of a DAG grammar has to recall several nonterminals and not just one. If we summarize those nonterminals to one meta-state per derivation step, can we then accept DAGs with a finite state automaton? If so, which kind of DAG languages can the finite state automaton recognize?

Definition 3.1 (Meta-state). *A multiset over nonterminals, i.e. an element of $\mathbb{N}^N$, is called a* meta-state. *The symbol $\mathscr{Q}$ is used for sets of meta-states, thus $\mathscr{Q} \in 2^{\mathbb{N}^N}$. For a DAG $G = (V, E, \ell, in, out)$ over $\Sigma \cup N$ we let $\underline{G}$ denote the meta-state $\ell_M(\{v \in V \mid \ell(v) \in N\})$, the multiset of all labels which are nonterminals.*[4]

Observe that it only depends on the meta-state of a graph, not on the graph as a whole, how the derivation can proceed. For this we define the notion of a graph being useful.

[4]Note that, according to the *Notation* section, the function ℓ_M returns a *multiset* of labels.

Definition 3.2 (Useful)**.** *We say that a prefix DAG G is* useful *with respect to a given grammar $\mathscr{G}$, if this grammar $\mathscr{G}$ derives $\varnothing \Rightarrow_R^* G \Rightarrow_R^* G'$ with the DAG G' being complete and* language-useful *if G' is complete and connected, thus equivalently, if G occurs in a derivation of a DAG $G' \in L(\mathscr{G})$.*

Note that a prefix DAG with respect to a grammar is useful if and only if all its connected components are language-useful. Which properties depend on the meta-state only?

Lemma 3.3 (Meta-state dependent)**.** *Let G, G' be prefix DAGs with $\underline{G} = \underline{G'}$ derived by a DAG grammar $\mathscr{G} = (N, \Sigma, R)$. Then, $\mathscr{G}$ can apply the rule $r \in R$ as the next derivation step $G \Rightarrow_r H$ iff $G' \Rightarrow_r H'$. Similarly, G is useful if and only if G' is useful. However, if G is language-useful this only implies that G' is useful.*

Interestingly, a minimal grammar can finalize every derivation to a complete DAG.

Lemma 3.4 (Useful Prefix DAG)**.** *Let G be a prefix DAG and $\mathscr{G} = (N, \Sigma, R)$ be a DAG grammar without useless rules. If $\varnothing \Rightarrow_R^* G$, then G is* useful with respect to *$\mathscr{G}$, i.e. there exists a derivation $\varnothing \Rightarrow_R^* G \Rightarrow_R^* G'$ for a complete DAG G'. If G' is connected it is* language-useful.

After having defined the notion of a meta-state, let us use them for derivations. The first naive idea is to use all meta-states which occur in all the derivations for complete graphs. We call that set $\mathscr{Q}_\mathbf{0}$ because a derivation of a DAG in the language starts and ends with zero states. Apart from $\mathscr{Q}_\mathbf{0}$ also another set of meta-states is of interest. While $\mathscr{Q}_\mathbf{0}(\mathscr{G})$ incorporates all meta-states occurring during all derivations of DAGs in $L(\mathscr{G})$, not all of these meta-states may be needed to generate the language $L(\mathscr{G})$. This observation gives rise to a smaller set of meta-states $\mathscr{Q}_{\min}$.

Definition 3.5 ($\mathscr{Q}_\mathbf{0}$ and $\mathscr{Q}_{\min}$)**.** *Let $\mathscr{G}$ be DAG grammar. The set of all meta-states that occur in derivations of DAGs in $L(\mathscr{G})$ is denoted by $\mathscr{Q}_\mathbf{0}(\mathscr{G})$ with*

$$\mathscr{Q}_\mathbf{0}(\mathscr{G}) = \{\underline{G} \mid G \text{ is a DAG which is language-useful with respect to } \mathscr{G}.\}.$$

A minimal set of meta-states is denoted by $\mathscr{Q}_{min}$. And, $\mathscr{Q}_{min}(\mathscr{G})$ denotes any set of meta-states such that

1. *every DAG $G_n \in L(\mathscr{G})$ has a derivation $\varnothing \Rightarrow_{r_1} G_1 \Rightarrow_{r_2} \ldots \Rightarrow_{r_n} G_n$ such that $\underline{G_1}, \ldots, \underline{G_n} \in \mathscr{Q}_{min}(\mathscr{G})$, and*
2. *there is no set of meta-states of smaller cardinality with this property.*

Thus, $\mathscr{Q}_{\min}(\mathscr{G})$ is a minimal set of meta-states sufficient to generate a DAG language, while $\mathscr{Q}_\mathbf{0}(\mathscr{G})$ incorporates also meta-states that could be dispensed. The set $\mathscr{Q}_{\min}(\mathscr{G})$ is not necessarily unique since often several derivations exist for one DAG, and furthermore a permutation of derivation steps may result in different meta-states. In general, we are interested in $|\mathscr{Q}_{\min}(\mathscr{G})|$, and in particular its finiteness, rather than in the set itself.

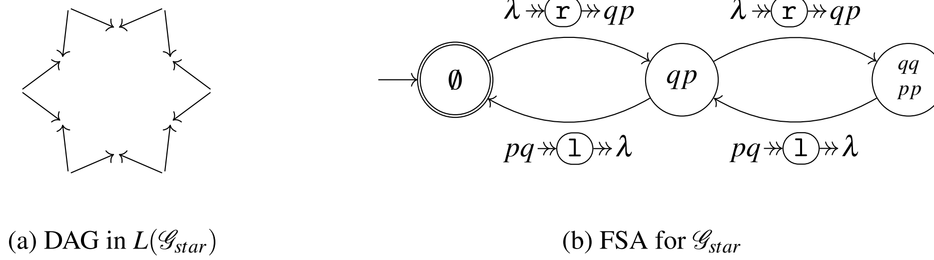
The subsequent example illustrates the existence of DAG grammars for which $\mathscr{Q}_{\min}(\mathscr{G})$ is finite while $\mathscr{Q}_\mathbf{0}(\mathscr{G})$ is not.

Example 3.6 (DAG language of stars)**.** *Consider a DAG grammar $\mathscr{G}_{star}$ with the rules $r = \lambda \twoheadrightarrow$* (r) *$\twoheadrightarrow qp$ and $l = pq \twoheadrightarrow$* (l) *$\twoheadrightarrow \lambda$. Let $G \in L(\mathscr{G}_{star})$, cf. Fig. 3a. First, $\varnothing \Rightarrow_{r^n} G_{root} \Rightarrow_{l^n} G$ for $n \geq 1$ generates a graph G_{root} consisting of n roots labeled* `r`*. Subsequently, l fuses pairs of nonterminal vertices into a single leaf labeled* `l`*. Collecting the meta-states that occur in these derivations or in an arbitrary derivation $\varnothing \Rightarrow_R^* G$ both result in $\mathscr{Q}_\mathbf{0}(\mathscr{G}_{star}) = \{p^n q^n \mid n \in \mathbb{N}\}$, since every rule either consumes or produces both a pair of nonterminals q and p. However, by first generating a single root, then alternating between r and l, and finally applying l once more, $\mathscr{G}_{star}$ offers the derivations $\varnothing \Rightarrow_{r(rl)^*l} G$ whose largest meta-state is p^2q^2. Hence, $\mathscr{Q}_{min}(\mathscr{G}_{star}) = \{pq, p^2q^2\}$.*

The previous example gives rise to the following observation.

Observation 3.7. *DAG Grammars $\mathscr{G}$ with finite $\mathscr{Q}_{min}(\mathscr{G})$ but infinite $\mathscr{Q}_\mathbf{0}(\mathscr{G})$ exist.*

This brings us to a further investigation of the finiteness of $\mathscr{Q}_\mathbf{0}$ and $\mathscr{Q}_{\min}$.



(a) DAG in $L(\mathscr{G}_{star})$

(b) FSA for $\mathscr{G}_{star}$

Figure 3: The grammar $\mathscr{G}_{star}$ gives rise to an FSA (b) that accepts DAGs like (a) (*labels are omitted*).

4 Finite Number of Meta-States

The previous section showed that languages generated with finite $\mathscr{Q}_{\min}$ indeed exist. This section investigates which types of DAG languages can be generated with a finite number of meta-states induced by the rules of a minimal deterministic grammar. First, the newly identified language class deserves a name. We call it *finite induced meta-state DAG language*. The term *induced* is chosen since the grammar $\mathscr{G}$ induces this set $\mathscr{Q}$ by a suitable (c.f. Lemma 4.4) or all (c.f. Lemma 4.3) derivation DAGs.

Definition 4.1 (Finite induced meta-state DAG language (`FID`))**.** *A language recognized by a minimal deterministic grammar $\mathscr{G}$ with finite $\mathscr{Q}_{min}(\mathscr{G})$ is called a* finite induced meta-state DAG language. *The language class comprising finite induced meta-state DAG languages is denoted by* `FID`*.*

In the following, we look into different categories for finite sets of meta-states. For each category we check three types, first languages, second paths and finally rule cycles.

Lemma 4.2 (Finite $L(\mathscr{G})$ – finite $\mathscr{Q}_{\mathbf{0}}$ and $\mathscr{Q}_{\min}$)**.** *Let $\mathscr{G} = (N, \Sigma, R)$ be a DAG grammar.*

1. *If $L(\mathscr{G})$ is finite, the sets $\mathscr{Q}_{\mathbf{0}}(\mathscr{G})$ and $\mathscr{Q}_{min}(\mathscr{G})$ are finite as well.*
2. *For a rule path Π of finite length within R, whose marked rules comprise only marked nonterminals except for the weak marked rules, finitely many meta-states suffice for all derivation sequences which generate the corresponding path π of Π in a DAG.*

Proof. Since N is finite and $L(\mathscr{G})$ is finite, only a finite number of derivations for all DAGs in L exists. Obviously, this combination yields a finite number of meta-states for $\mathscr{Q}_{\mathbf{0}}(\mathscr{G})$ and consequently also for $\mathscr{Q}_{\min}(\mathscr{G})$. In the second statement, the finite rule path Π gives rise to a finite derivation $G \Rightarrow_{\Pi} G'$ for all possible prefix DAGs G of G', since, due to all nonterminals marked except start and end, Π does not need to be interleaved with rules. A finite derivation yields finitely many meta-states, both for $\mathscr{Q}_{\min}$ as well as for $\mathscr{Q}_{\mathbf{0}}$. □

Infinite languages $L(\mathscr{G})$ do not necessarily induce an infinite $\mathscr{Q}_{\min}$ and not even an infinite $\mathscr{Q}_{\mathbf{0}}$. First, we look at those cases, where they indeed induce finite sets of meta-states only.

Lemma 4.3 (Strings – finite $\mathscr{Q}_{\mathbf{0}}$ and $\mathscr{Q}_{\min}$)**.** *Let $\mathscr{G} = (N, \Sigma, R)$ be a DAG grammar.*

1. *If $L(\mathscr{G})$ is a string DAG language the sets $\mathscr{Q}_{\mathbf{0}}(\mathscr{G})$ and $\mathscr{Q}_{min}(\mathscr{G})$ are finite*
2. *For a, possibly arbitrary long, directed rule path, Π in R, whose marked rules comprise only marked nonterminals except for the weak marked rules, finitely many meta-states suffice for all derivation sequences which generate the corresponding path π by Π in a DAG.*

3. *For a directed rule cycle c whose marked rules comprise only marked nonterminals, finitely many meta-states suffice for all derivation sequences which generate the corresponding path π by c in a DAG.*

Proof. Since string languages consume and produce exactly one nonterminal in every derivation step except for the first and last steps producing and consuming one nonterminal only, respectively, N equals $\mathscr{Q}_{\mathbf{0}}(\mathscr{G})$.[5] Obviously, this means that $\mathscr{Q}_{\mathbf{0}}(\mathscr{G})$ is finite, since N is finite.

This carries over to subgraphs which are string DAGs. Strings as subgraphs are derived in the statements (2) and (3). Let $\bar{r}$ denote a marked rule with only marked nonterminals of the form $\check{q} \twoheadrightarrow (\sigma) \twoheadrightarrow \hat{p}$. Such a rule $\bar{r}$ does not alter the size of the meta-states, thus $|\underline{G}| = |\underline{G}'|$, in its derivation step $G \Rightarrow_r G'$. For a directed rule cycle c with only marked nonterminals holds the same just as for an infinite rule path Π in R, since they consist of a rule sequence with rules of type $\bar{r}$. Such a rule path Π comprises only finitely many marked rules since R is finite. Thus, an arbitrary long rule path corresponds to a rule cycle. Consequently, $\mathscr{Q}_{\mathbf{0}}$ is finite for both (2) and (3).

In all three cases $\mathscr{Q}_{\mathbf{0}}$ is finite, yielding a finite $\mathscr{Q}_{\min}$. □

In summary, for strings of any kind, string languages or strings as subgraphs, also of unbounded length, both $\mathscr{Q}_{\min}$ and $\mathscr{Q}_{\mathbf{0}}$ stay finite. This was to be expected since regular string languages are accepted by finite state automata with a finite set of states. What happens beyond string DAGs?

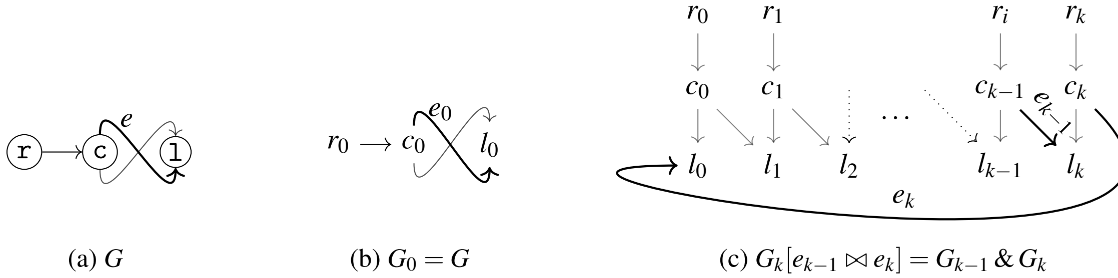
Lemma 4.4 (Finite $\mathscr{Q}_{\min}$ – infinite $\mathscr{Q}_{\mathbf{0}}$)**.** *Let $\mathscr{G} = (N, \Sigma, R)$ be a minimal deterministic DAG grammar*

1. *In the case where $\mathscr{Q}_{min}(\mathscr{G})$ is finite, it is possible that $\mathscr{Q}_{\mathbf{0}}(\mathscr{G})$ is infinite.*
2. *For a possibly arbitrary long, undirected rule path, Π in R, whose marked rules comprise only marked nonterminals except for the weak marked rules, finitely many meta-states suffice for deriving all complete DAGs incorporating the corresponding path π generated by Π, however, some derivations for each such complete DAG ask for an infinite set of meta-states.*
3. *For an undirected rule cycle c in R, whose marked rules comprise only marked nonterminals, finitely many meta-states suffice for deriving all complete DAGs incorporating the corresponding path π of unbounded length generated by c. However, some derivations for each such complete DAG ask for an infinite set of meta-states.*

Proof. The first statement is equivalent to Observation 3.7. Both Statements (2) and (3) may yield a path π of unbounded length. Such a path π gives rise to a derivation which does not increase the cardinality of the meta-state, such that $\mathscr{Q}_{\min}$ with respect to Π and c is finite. This is possible by using rules that add nonterminals to the current meta-state only when they are needed, just like for the DAG language of stars in Example 3.6. But for undirected rule paths and cycles, we can rearrange the rules in the derivations in order to first generate all roots (again, c.f. Example 3.6). Since both Π and c can make the path π unbounded, the size of the meta-states will grow without bound, yielding an infinite $\mathscr{Q}_{\mathbf{0}}$. □

In the next section, we will turn to languages which cannot be generated with a finite set of meta-states. Prior to this, we look at the unexpected fact that although a label does not occur in any rule cycle, its number of occurrences could be unbounded.

[5] Since we use strings as a notation for multisets, this special set of multisets equals a normal set.



(a) G (b) $G_0 = G$ (c) $G_k[e_{k-1} \bowtie e_k] = G_{k-1} \,\&\, G_k$

Figure 4: The decorated one-pointed star G is shown in (a). This equals the star G_0 in (b), where in addition to the vertex label, the label's index allows us to reference each vertex uniquely by its number of copies. Note thus, that in (b), as well as in (c), the index is not part of the vertex label. To draw the graph itself in those two pictures, the indices would be stripped off. Note, (c) illustrates the swapping of $k+1$ disjoint isomorphic copies to a $k+1$-pointed star $G(e \bowtie e)^k$.

Lemma 4.5 (Unbounded without Rule Cycle)**.** *There exists a DAG grammar $\mathscr{G} = (N, \Sigma, R)$ and a vertex or edge label* u $\in \Sigma \cup N$ *such that the number of occurrences of* u *in a (derivation) DAG generated by $\mathscr{G}$ is unbounded, although* u *does not occur in any rule cycle.*

The following lemma summarizes when the number of label occurrences is not bounded.

Lemma 4.6 (Unbounded Label Occurrence)**.** *Let $\mathscr{G} = (N, \Sigma, R)$ be a minimal deterministic DAG grammar and* u $\in \Sigma \cup N$ *a label of a vertex or an edge. The number of occurrences of* u *in graphs $G \in L(\mathscr{G})$, or, for edge labels, in their corresponding derivation DAGs D, is unbounded, iff one of the two following conditions is fulfilled:*

a) Label u *occurs in some rule cycle of $\mathscr{G}$.*

b) There exist both a rule cycle c in which an unmarked $q \in N$ occurs as well as a rule path Π between this nonterminal q and the label u*.*

5 Infinite Number of Meta-States

Unfortunately, from the algorithmic viewpoint at least, there exist infinite DAG languages where a finite number of meta-states is not sufficient for generating them in a top-down deterministic manner. We call this language class the infinite meta-state DAG languages, abbreviated as `ID`.

Definition 5.1 (Infinite meta-state DAG language (`ID`))**.** *A language $L(\mathscr{G})$ of a minimal deterministic DAG grammar $\mathscr{G}$ is called an* infinite meta-state DAG language *if $\mathscr{Q}_{min}(\mathscr{G})$ is infinite. The language class comprised of all infinite meta-state DAG languages is denoted* `ID`*.*

Lemma 5.2 (Infinite $\mathscr{Q}_{\text{min}}$ – infinite $\mathscr{Q}_{\mathbf{0}}$)**.** *Infinite meta-state DAG languages exist.*

Proof. Let us consider the following DAG grammar for binary trees $\mathscr{G}_{tree} = (\{q\}, \{\texttt{r}, \texttt{m}, \texttt{L}\}, R)$. As a tree, every $G \in L(\mathscr{G}_{tree})$ has only one root. Therefore, every derivation for the tree grammar $\mathscr{G}_{tree}$ applies $r_r = \lambda \twoheadrightarrow \!\text{(r)}\! \twoheadrightarrow qq$ only once: $\varnothing \Rightarrow_{r_r} G_{\text{root}} \Rightarrow_{R \setminus \{r_r\}} G$. The leaf rule $r_l = q \twoheadrightarrow \!\text{(l)}\! \twoheadrightarrow \lambda$ cannot be part of a rule cycle since this requires a minimum of two nonterminals. The rule $r_m = q \twoheadrightarrow \!\text{(m)}\! \twoheadrightarrow qq$ is the only rule that occurs in a rule cycle.

Every derivation step using r_m increases the cardinality of the meta-state by one due to consuming one nonterminal and producing two nonterminals. If r_m is the only rule in a rule cycle and r_m increments

the meta-state in every derivation step $\Rightarrow_{r_m}$, do we end up with an infinite set $\mathscr{Q}_{\min}$? Luckily, Lemma 4.5 tells us that we can repeatedly apply derivation steps also with rules not being part of a rule cycle. Lemma 4.6 confirms that this is the case for the rule r_l, since we have an unmarked q in all marked rule combinations of r_m for a rule cycle and we have a rule path q to `l` only with the rule r_l. The number of leaves, labeled by `l`, is not bounded. As a sole rule in R the leaf rule decreases the cardinality of the meta-state. It is folklore that the amount of leaf vertices is one more than non-leaf vertices. This sounds thus promising to end up for a finite set $\mathscr{Q}_{\min}$. The derivations $\varnothing \Rightarrow_{r_r} G_{\text{root}} \Rightarrow_{r_l} G'_{\text{root}} \Rightarrow_{(r_m r_l)^*} G' \Rightarrow_{r_l} G$ generate DAGs in $L(\mathscr{G})$. Those derivations include only two meta-states $\underline{G}_{\text{root}} = q^2$ and $\underline{G}_{\text{root}} = \underline{G}' = q$. Unfortunately, this does not yield fully balanced trees.

If we consider the fully balanced binary trees of depth n, where every leaf is the nth vertex in a path from the root to the leaf, $\mathscr{G}$ has to generate at least $n-2$ vertices labeled `m` or $n-1$ vertices not labeled `l` until it can apply the rule r_l. This, in turn, means that $n-2$ derivation steps increment the meta-state. The generation of a complete binary tree results in a meta-state q^n. For generating the language of binary trees of all depths, $\mathscr{Q} = \{q^i \mid i \in [n]\}$. But this set is minimal, since no other derivations exist, and infinite which shows that $\mathscr{Q}_{\min}(\mathscr{G}_{tree})$ is infinite. Note, $\mathscr{G}_{tree}$ is a deterministic and minimal grammar. Thus, minimal deterministic DAG grammars with finite $\mathscr{Q}_{\min}$ exist. □

After proving the existence of languages in the language class `ID`, we present languages in this class. The following theorem about certain tree languages follows directly from Lemma 5.2.

Theorem 5.3 (Trees in `ID`)**.** *A minimal deterministic DAG grammar $\mathscr{G} = (N, \Sigma, R)$ generating fully balanced trees of unbounded size, $L(\mathscr{G})$ is in* `ID`.

Proof. For $L(\mathscr{G})$ comprising fully balanced trees with one root, $L(\mathscr{G}) \in$ `ID` was shown to be true in Lemma 5.2. For trees "upside down", one leaf and many roots, the argumentation is similar. Instead every derivation step $\Rightarrow_{r_m}$ for a path from a root to a leaf adds a state to produce the ensuing meta-state. In doing so, it forces dependencies to be derived. The top-down generation of $\mathscr{G}$ prevents the isolated derivation of a root-to-leaf path. A vertex at depth n requires n roots. Therefore, the size of the meta-states depends on n, yielding infinitely many meta-states. Thus, the dependencies are responsible for the infinite class $\mathscr{Q}_{\min}(\mathscr{G})$. □

Trees are fine, but what about graphs? We observed (Obs. 2.8) that only undirected rule cycles can yield cycles in DAGs. Directed trees are generated by directed rule cycles. So let's consider languages with proper graphs, thus generated by undirected rule cycles.

Lemma 5.4 (DAG Cycles in `ID`)**.** *Let $\mathscr{G}$ be a minimal deterministic DAG grammar. Then $L(\mathscr{G}) \in$ `ID` if some DAG $G \in L(\mathscr{G})$ contains a cycle of arbitrary length that exhibits a chord path.*

Proof. As Lemma 5.2 tells us, generating a cycle of arbitrary length causes $\mathscr{Q}_{\mathbf{0}}(\mathscr{G})$ to be infinite, but not necessarily $\mathscr{Q}_{\min}(\mathscr{G})$. Hence, the infinite size of $\mathscr{Q}_{\min}(\mathscr{G})$ hides in the rule cycle's chord path.

In a DAG, a cycle of arbitrary length $m \in \mathbb{N}$ requires the repeated application of an undirected rule cycle (Obs. 2.8) in an unbounded number of iteration steps, say, in $n \in \mathbb{N}$ iteration steps, since the set of rules in $\mathscr{G}$ is finite and, in order to obtain a cycle of arbitrary length m, as stated, by the pigeonhole principle, we need an unbounded number n of applications of the required rule cycle.

Let c be this undirected rule cycle that is required. With the constant $|c|$ denoting the number of rules c consists of, the length of the DAG's undirected cycle is $m = |c| \cdot n$.

Consider the DAG G generated by $2n$ rule cycle iterations of c, yielding a cycle of length $2 \cdot |c| \cdot n$, as detailed in the following.

As stated, the cycle in G has a chord path. However, if one application of c generated the stated chord path, a graph generated by many applications of c, could, if 'wired' appropriately, contain many such chord paths instead of just one. The grammar $\mathscr{G}$ generated G by $2n$ applications of c. Consequently, $\mathscr{G}$ can generate G in a way that it exhibits $2n$ such chord paths, thus, with identical edge labels as the stated chord path exhibits. The identical labeling is key for the proof, since this will allow 'wiring' the chords differently.

By definition (Def. 2.7), a rule cycle includes marked rules with a minimum of two nonterminals with two of them marked. However, obviously, the end vertices of a chord path have a vertex degree of at least three [34]. Due to that, the marked rules generating the chord path need at least three nonterminals and that means one unmarked nonterminal. Thus, a chord path requires an unmarked nonterminal. An unmarked nonterminal needs to be *saved* in a meta-state. Let us call this nonterminal s. Assume that $\mathscr{G}$ opens and closes the chord paths one after another while generating G. Then $\mathscr{G}$ could possibly reuse the meta-state in which we stored s: when it opens the chord path, it stores s in the ensuing meta-state, when it closes the chord path, it does not need to save s anymore. Then, $\mathscr{G}$ repeats this step for the next chord path, reusing the meta-state in which it stored s.

Yet, Lemma 2.11 allows the swapping of the $2n$ chord paths. In order to obtain one of those graphs in $L(\mathscr{G})$ requiring an infinite $\mathscr{Q}_{\min}(\mathscr{G})$, let us swap the chord paths in G (technically more precise: certain edges of the chord paths). Let e_i, labeled with the afore mentioned $\ell(e_i) = s$, be one of the ends of the chord path number i for $i \in [2n]$. Swapping $G[e_1 \bowtie e_n]$, $G[e_2 \bowtie e_{n-1}]$, ..., thus $G[e_k \bowtie e_{n-k+1}]$ for $k \in [n/2]$ results in the first n chord paths bridging a distance of length $|c| \cdot n$ of its DAG cycle.

Please note: Swapping the second half of the chord paths is unnecessary, but can be performed in order to obtain a symmetric, beautiful graph. If we would, however, swap the $2n$'th chord path with the first, the distance would be one length of c, since the chords are arranged on a cycle. First and $2n$'th chord paths are neighbors, thus. That is the reason for choosing $2n$ instead of n cycle iterations of c. The first and the n'th chord paths are a maximum distance apart with respect to the cycle generated by $2n$ applications of c.

After these swapping operations, the chords are nested: $\mathscr{G}$ opens the first chord path before the second but closes the second before the first, and so on, according to the FILO (first in, last out) principle. This constitutes the proof: by opening $n/2$ chord paths, all identically labeled with s, there exists a meta-state in $\mathscr{Q}_{\min}(\mathscr{G})$ that has to store all those $s^{n/2}$ labels. Then, $\mathscr{G}$ has to remember that all these edges of the chord paths are dangling. Thus, the number of meta-states depends on the number of cycle iterations $2n$, which completes the proof, since $n \in \mathbb{N}$. □

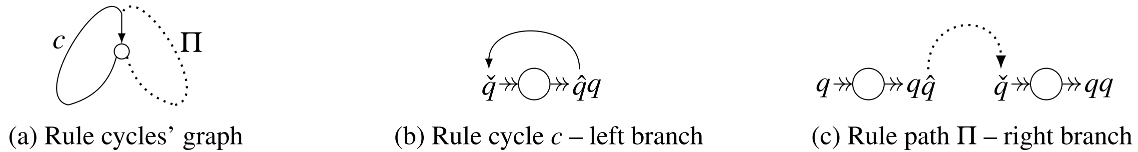
6 Characterization

Now we contemplate the newly found language classes which divide the class of top-down deterministic regular DAG languages RDL^{det}, whose languages are generated by deterministic DAG grammars, into disjoint subclasses.

Observation 6.1 (`FID` $\cup$ `ID` $= RDL^{\text{det}}$)**.** *By the definitions of* `FID` *and* `ID`*, it is true that* `FID` *and* `ID` *are disjoint and that* `FID` $\cup$ `ID` $= RDL^{det}$.

The following lemma characterizes the language class `ID` by the rules of a grammar. Since `ID` and `FID` are disjoint, indirectly it is also a characterization of `FID`.

Theorem 6.2 (Characterization of `ID`)**.** *For a minimal deterministic DAG grammar $\mathscr{G} = (N, \Sigma, R)$ its set of meta-states $\mathscr{Q}_{min}(\mathscr{G})$ is infinite iff there exist a rule cycle c, a rule path Π and not necessarily distinct nonterminals $q, p \in N$ satisfying the following conditions:*



(a) Rule cycles' graph (b) Rule cycle c – left branch (c) Rule path Π – right branch

Figure 5: Rule cycles and their graphs for the binary tree language given by a cycle and its chord

- *The nonterminals q and p occur in c as unmarked $q \in N$ and marked $\bar{p} \in \bar{N}$.* (c)
- *The rule path Π lies between q and p.* (Π)
- *The nonterminal q is in c in a head iff it is in Π's weak rule in the head.* $(c\Pi)$

Proof. Before proving the stated bi-implication, observe that the above conditions result in two distinct rule cycles. It is easy to see that Π acts as a chord path for the rule cycle c, and consequently, provides an alternative rule cycle. Fig. 5 demonstrates an example. For the rule cycle c condition (c) and for the rule path Π condition (Π) urge the suitable nonterminals for gluing Π as a chord to c. The requirements stated in $(c\Pi)$, concerning the nonterminals' positions in head or tail ensure the right orientation of the edge labeled q. And, this alternative rule cycle provided by Π is the crux of the matter. Since the grammar has always two options to continue the derivation, it always has to remember the dangling edges of the alternative it did not decide to generate next. Thus, in every cycle iteration, regardless of the decision how to proceed, a new meta-state is needed since the number of dangling edges grows in an unbounded fashion.

We start with assuming that $\mathscr{G}$'s rules have the stated form and then prove that $\mathscr{Q}_{\min}(\mathscr{G})$ is infinite. The proof by construction relies on two techniques. We construct a graph G_n for which the number of meta-states in $\mathscr{Q}_{\min}(\mathscr{G})$ needed to generate it depends on the number of rule cycle iterations $n \in \mathbb{N}$. The first technique is the (directed or undirected) path whose length depends on n: Our grammar $\mathscr{G}$ can generate the above stated DAG G_n since a rule cycle can yield a path of length $k \cdot n$ where k is the number of edges generated by the marked nonterminals in one cycle iteration. A path as such is connected, and our path is a subgraph of a connected prefix DAG, generated top-down by $\mathscr{G}$. As such, it is a connected prefix DAG which, according to Lemma 3.4, $\mathscr{G}$ can complete to a language-useful DAG. The second technique is to disregard all the edges and vertices not mentioned in the conditions stated about the rules of the grammar. Obviously, they are irrelevant since, due to the swapping operation (Theorem 2.11), they would only *increase* the size and number of the meta-states.[6] Like that, $\mathscr{G}$ causes $\mathscr{Q}_{\min}$ to be infinite by deriving G_n with a path of length n as a subgraph via:

- *directed* rule cycles and therefore fully balanced binary trees as subgraphs (Lemma 5.3)
- *undirected* rule cycles and therefore proper DAGs as shown in Lemma 5.4.

Consequently, $\mathscr{G}$ generates a DAG G_n whose size and number of meta-states depend on the number of rule cycle iterations which are unbounded. Like that, $\mathscr{Q}_{\min}(\mathscr{G})$ is infinite.

The direction vice versa assumes an infinite $\mathscr{Q}_{\min}(\mathscr{G})$ and arguments by negating the conditions one by one: The existence of the stated rule cycle and rule path, as well as the requirements (c), (Π) and $(c\Pi)$ have to meet are iteratively shown to hold by negating them and concluding a contradiction.

- Assume no Π exists. By their definition, grammars without rule paths do not generate DAGs. With $\mathscr{Q}_{\min}(\mathscr{G})$ being infinite, a rule path must exist. Assume no c exists. Languages without a

[6] Note that, with this second technique applied, k – the number of edges generated by the marked nonterminals in one cycle iteration – equals to one. We abstract away how many edges the cycle exactly generates. Relevant to distinguish between finite and infinite $\mathscr{Q}_{\min}$ is not the constant factor k, but only the number of iterations n.

rule cycle are finite (Lemma 2.9). Finite languages have a finite $\mathscr{Q}_{\min}$ (Lemma 4.2), contradicting our assumption that no rule cycle exists. Infinite languages are necessarily generated by cycles (Lemma 2.9). Thus, both rule path and cycle, called Π and c, exist.

- Assume condition (c) does not hold. By definition, no rule cycles without marked nonterminals exist, thus $\bar{p}$ must occur in c. A rule cycle without an unmarked nonterminal, here q, yields finitely many meta-states (Lemma 4.4, 3.). Condition (c) holds.
- Without condition (Π), thus without Π being connected with its both ends q and p to c, there would be no chord path[7]. First, assume that there would be no rule path with q as one of its ends. However, it is immediate that some rule path is indeed connected to c: Since q occurs unmarked in c, a rule path Π connected by q necessarily exists, yielding – without the chord – an infinite set of meta-states, indeed. But, it is $\mathscr{Q}_{\mathbf{0}}(\mathscr{G})$ which is infinite, not $\mathscr{Q}_{\min}(\mathscr{G})$ – by Lemma 4.4 again – in case Π does not lead back to c via $\bar{p}$, our second assumption when negating (Π). This contradiction shows that a path Π between q and p must exist for an infinite $\mathscr{Q}_{\min}$.
- Negating condition $(c\Pi)$ means allowing q as unmarked in a head of c while it is in a tail of Π's two weak rules or vice versa. This would induce the wrong orientation of q so that Π would not be a chord. And, we already know from the previous point that Π has to be a chord.

We cannot sacrifice any condition without contradicting our assumption of an infinite $\mathscr{Q}_{\min}(\mathscr{G})$ which proves the second direction and by that completes the proof. □

What if we restrict a grammar $\mathscr{G}$ with a set of meta-states $\mathscr{Q}$ instead of deriving this set out of the grammar? In that case the grammar's language possibly changes. Whereas when just extracting $\mathscr{Q}_{\min}(\mathscr{G})$, the grammar's language is not altered.

Definition 6.3 (Finite meta-state DAG language (FD))**.** *A minimal deterministic grammar* $\mathscr{G} = (N, \Sigma, R)$ *generates a* finite meta-state language $L^{\mathscr{Q}}(\mathscr{G})$ *where a finite set* $\mathscr{Q} \subseteq \mathbb{N}^Q$ *is given to restrict which rules in R can be used. The derivation step* $G_1 \Rightarrow G_2$ *is only allowed if the meta-state* $\underline{G_2} \in \mathscr{Q}$.

Languages in the classes `FID` and `FD` can use their finite sets of meta-states, $\mathscr{Q}_{\min}$ and $\mathscr{Q}$, respectively, to construct a classical finite state automaton to recognize themselves.

7 Classical Finite State Automata for DAG Languages

This section describes how classical finite state automata come into play when recognizing certain DAG languages. An `FD` language can be recognized by a classical deterministic finite state automaton (DFA). As an example, see Figure 3b which shows the automaton for accepting DAGs of $L(\mathscr{G}_{star})$ as defined in Example 3.6. The top-down reading process induces merely a partial order on the vertices. The deterministic automaton thus reads a DAG in a partly nondeterministic fashion.

A DFA is a five-tuple $A = (Q, \Gamma, \delta, q_0, q_\$)$ with the finite sets Q and Γ being the states and the alphabet, resp., $q_0, q_\$ \in Q$ being the start and final state, resp., and $\delta : (Q \times \Gamma) \to Q$ being the transition function, extended inductively to strings $\delta(Q \times \Gamma^*) \to Q$ by applying δ symbol-wise. We omit entries of δ if they do not lead to an accept state and thus consider only partial DFAs.

Every DAG grammar $\mathscr{G} = (N, \Sigma, R)$ with $L(\mathscr{G})$ being a finite meta-state DAG language gives rise to a DFA $M = (R, \mathscr{Q}_{\min}(\mathscr{G}), \delta, \emptyset, \emptyset)$ such that for all $\mathbbm{q} \in \mathscr{Q}_{\min}$ the transition function yields the following if $\alpha \subseteq \mathbbm{q}$ then

$$\delta(\mathbbm{q}, (\alpha \twoheadrightarrow \textcircled{\sigma} \twoheadrightarrow \beta)) = (\mathbbm{q} \setminus \alpha) \cup \beta.$$

[7] Recall the introduction of the proof for the explanation of Π being a chord path.

Although M meets the requirements of a DFA, we call its states in $\mathscr{Q}_{\min}$ meta-states instead to avoid confusion. While reading a string with a DFA is as easy as reading it symbol by symbol, reading a DAG is somewhat more complicated since the vertices do not exhibit an obvious total order. Therefore, M reads the vertices partly nondeterministically. Let $G = (V, E, \ell, in, out)$ be a DAG in $\mathscr{D}_\Sigma$. Such a DAG has no labels assigned to its edges. We denote this by $\ell(e) = \#$. Since we are restricted to top-down procedures, reading a vertex means assigning states to the outgoing edges: Consequently, M may read a vertex only if all of its ingoing vertices have been assigned labels, thus $\# \notin [\ell(in)]$. If $\ell(v)$ matches the σ of a rule $r = (\alpha \twoheadrightarrow \textcircled{\sigma} \twoheadrightarrow \beta)$ such that $\delta(\mathtt{q}, r)$ yields a new state $\mathtt{q}'$ of M, M can read v and assigns the labels β to its outgoing edges: $\ell(out(v)) = \beta$.[8]

By reading the vertices top-down, M will accept a DAG if it nondeterministically chooses the right order of vertices. The order in which the vertices are read is already restricted by imposing the requirement to read top-down. But, we can improve upon this by fine-tuning the order in which the roots are read. When M cannot read any non-root, then there exists no vertex with all its ingoing edges labeled. Instead of choosing an arbitrary root next – since in a top-down a root has no prerequisites and can always be read top-down – M can choose a root which is needed next. Which root $v_{\text{root}} \in V$ do we need next? One where the DAG G has a path from v_{root} to a vertex v_{next} whose ingoing vertices are labeled as well as not labeled.

We defined the language class FD at the end of the last section (c.f. Definition 6.3). Now that we know how a grammar $\mathscr{G}$ in combination with a finite set of meta-states $\mathscr{Q}(\mathscr{G})$ can determine a DFA for the DAG language $L(\mathscr{G})$, let us come back to the idea of restricting a grammar with a set of meta-states not derived from $\mathscr{G}$. Since languages easily become ID languages, the rules which are allowed to remain in the class of FID limit the expressiveness of languages. The motivation behind restricting via a finite $\mathscr{Q}$ strives to increase the expressiveness of a language while keeping the number of meta-states to recognize or generate it finite in order to profit from transferring the good algorithmic properties of regular string languages. We use the language which comprises DAGs looking like a rainbow to illustrate restricting a given ID language to become an FD language recognizable by an FSA, see Figure 6.

Theorem 7.1 (FD $\nsubseteq$ RDL^{det}). *The class of finite meta-state DAG languages is not a subset of the class of top-down deterministic regular DAG languages RDL^{det}.*

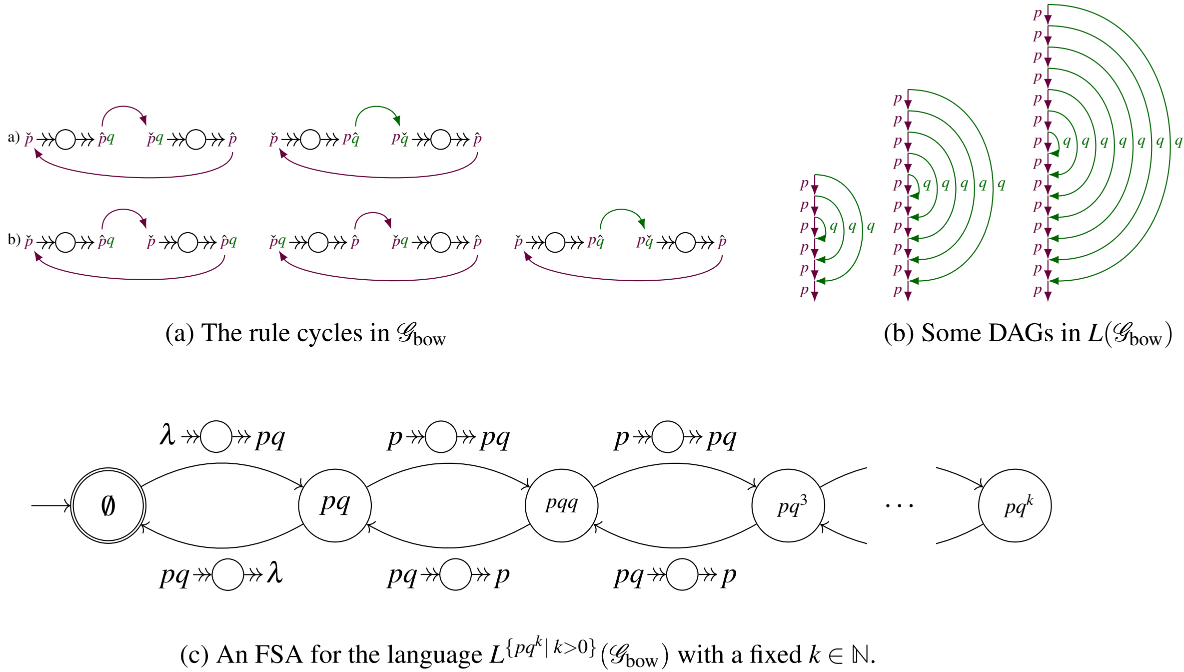
Proof. Given is a minimal deterministic DAG grammar $\mathscr{G} = (N, \Sigma, R)$ and a finite set of meta-states $\mathscr{Q}$. If FD $\nsubseteq$ RDL^{det} holds, then there exists a DAG language $L^{\mathscr{Q}}(\mathscr{G})$ that is not in the class of RDL^{det}. Our $\mathscr{G}$ is a deterministic DAG grammar and as such generates a language $L(\mathscr{G}) \in RDL^{\text{det}}$.

We try to construct a language not in RDL^{det} by limiting $\mathscr{G}$'s language to a finite one. Suppose, $\mathscr{G}$ generates an infinite language $L(\mathscr{G})$ and $\mathscr{Q}$ prevents derivations of any rule cycle in R. According to Lemma 2.9, then, $L^{\mathscr{Q}}(\mathscr{G})$ will be finite. However, Lemma 4.2 tells us that any finite language can be generated by a deterministic DAG grammar $\mathscr{G}' \neq \mathscr{G}$ such that $L(\mathscr{G}') = L^{\mathscr{Q}}(\mathscr{G})$. This attempt did not work out.

Our next attempt is the grammar $\mathscr{G}_{\text{bow}}$ with its infinite language $L(\mathscr{G}_{\text{bow}}) \in$ ID. Its rule set comprises $(\lambda \twoheadrightarrow \textcircled{\texttt{r}} \twoheadrightarrow pq), (p \twoheadrightarrow \textcircled{\texttt{o}} \twoheadrightarrow pq), (pq \twoheadrightarrow \textcircled{\texttt{c}} \twoheadrightarrow p)$ and $(pq \twoheadrightarrow \textcircled{\texttt{l}} \twoheadrightarrow \lambda)$. The grammar $\mathscr{G}_{\text{bow}}$ can among others generate DAGs similar to garlands and rainbows. To generate an arbitrary long DAG looking like a garland, the set of meta-states $\mathscr{Q} = \{\varnothing, p, pq\}$ suffices. If, however, $\mathscr{G}_{\text{bow}}$ generates a DAG looking like a rainbow, it repeats the rule with the vertex label o an unbounded number of times.[9] Such a rainbow DAG in

[8] Note that, by definition, the empty DAG $\varnothing$ is not in the language although the automaton accepts it due to the start state as accepting state. Only DAGs $G \in D_\Sigma$ are considered and $\varnothing$ is not in D_Σ.

[9] Allowing pennants spanning more than one vertex is possible, too. With $\mathscr{Q} = \{pq^n \mid p, q \in N \text{ and } n \in \mathbb{N}\}$ the bow(s) for the pennants in a DAG $G_{garland} \in L^{\mathscr{Q}}(\mathscr{G})$ can span the maximum of n vertices.



(a) The rule cycles in $\mathscr{G}_{\text{bow}}$

(b) Some DAGs in $L(\mathscr{G}_{\text{bow}})$

(c) An FSA for the language $L^{\{pq^k \mid k>0\}}(\mathscr{G}_{\text{bow}})$ with a fixed $k \in \mathbb{N}$.

Figure 6: The grammar $\mathscr{G}_{\text{bow}}$ generates, among others, DAGs looking like rainbows (*vertices implicit*) (a) via the rule cycles in (b). For a fixed $k \in \mathbb{N}$, restricting the grammar by $\mathscr{Q} = \{pq^k \mid k > 0\}$ allows the construction of an FSA and imposes a bound on the number of bows in a 'rainbow'.

$L^{\mathscr{Q}}(\mathscr{G})$, with the length of the bow(s) limited, can only be generated by a deterministic DAG grammar if the grammar's language is finite. But, a finite language cannot include DAGs with n-sized bows of arbitrary length. On the other hand, any grammar generating garland DAGs of unbounded length will also include the rainbow DAGs due to the possibility to swap the edges (Lemma 2.11). And, according to Lemma 4.6, we can swap the edges since the edge labels (the nonterminals) must be repeated for a DAG of unbounded length and distinct edges labeled with the same nonterminal in a derivation DAG can be swapped. Thus, restricting the language to $L^{\mathscr{Q}}(\mathscr{G}_{\text{bow}})$ by above given $\mathscr{Q}$ results in a language not in RDL^{det}, completing the proof. □

Again, swapping (Lemma 2.11) would allow us to generate unbounded rainbows, so we conclude: restricting a grammar by a set of meta-states can prevent the swapping operation.

Corollary 7.2 (Swapping in FD)**.** *For a grammar $\mathscr{G}$ with its $L^{\mathscr{Q}}(\mathscr{G}) \in$ FD $\setminus$ FID holds:*

- $\exists\, \lfloor D \rfloor \in L^{\mathscr{Q}}(\mathscr{G}) : \lfloor D(e_0 \bowtie e_1) \rfloor \notin L^{\mathscr{Q}}(\mathscr{G})$
- $|L^{\mathscr{Q}}(\mathscr{G})| < |L(\mathscr{G})|$
- $L(\mathscr{G}) = \{\, \lfloor D(e_0 \bowtie e_1) \rfloor \mid D(e_0 \bowtie e_1) \in L^{\mathscr{Q}}(\mathscr{G}) \,\}$
- $L(\mathscr{G}) \in$ ID

Contrary to FID-grammars on whose derivation DAGs swapping is always allowed, the derivation DAGs of grammars generating languages in $L^{\mathscr{Q}}(\mathscr{G}) \in$ FD$\setminus$FID the swapping operation is restricted by the given set of meta-states $\mathscr{Q}$. Restricting a grammar by allowing only certain meta-states corresponds to restricting swapping on the derivation DAGs. Via those restrictions, DAGs are lost which cannot be

accepted without the missing meta-states resulting in a language with less graphs. The transitive closure of the swapping operation on the language $L^{\mathscr{Q}}(\mathscr{G})$ returns the original language $L(\mathscr{G})$ which must be in the class ID.

For free – by FD definition – we can observe the closure properties valid for DFAs, since FD languages are recognized by DFAs.

Observation 7.3 (FD – Closure under Union and Intersection)**.**
The language class FD *is closed under union and intersection.*

8 Conclusion

We have defined the DAG language classes ID and FD and characterized them. By imposing the set of meta-states as a given restriction, we additionally have defined FD, which intersects with *RDL* but is not a subset of it – adding to expressiveness. For languages in FID and FD, we proved that it is possible to construct a classical string automaton recognizing the language. Analysing ID further, we expect similarities with the Chomsky hierarchy.

Pushdown Conjecture. *Analogous to languages in* FD *being recognized by a finite state automaton, we conjecture all languages in* ID *to be recognized by a pushdown automaton.*

The FSA construction by meta-states, could be applied not only to the top-down deterministic version, but also to the plain regular DAG automaton. By dropping the determinism restriction, similar to dropping the planarity restriction [32] imposed in [21], possibly the NP-completeness for the membership problem could be tackled. Imposing useful restrictions to be provided by the set of meta-states $\mathscr{Q}$, as the language class FD requires it, is the task for more application centric research, like semantic NLP parsing [29, 3].

References

[1] Lene Antonsen, Erik Axelson, Eckhard Bick, Børre Gaup, Sam Hardwick, Katri Hiovain-Asikainen, Arvi Hurskainen, Fred Karlsson, Kimmo Koskenniemi, Krister Lindén, Inari Listenmaa, Inga Mikkelsen, Sjur Nørstebø Moshagen, Flammie A Pirinen, Aarne Ranta, Jack Rueter, Daniel G. Swanson, Trond Trosterud & Linda Wiechetek (2023): *Rule-Based Language Technology*. *NEJLT Monographs* 2, Northern European Association for Language Technology (NEALT). Available at `http://hdl.handle.net/10062/89595`.

[2] Andrew Badr (2009): *HYPER-MINIMIZATION IN O(n^2)*. *Int. J. Found. Comput. Sci.* 20(4), pp. 735–746, doi:10.1142/S012905410900684X.

[3] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer & Nathan Schneider (2013): *Abstract Meaning Representation for Sembanking*. In: *Proc. 7th Linguistic Annotation Workshop, ACL 2013 Workshop*.

[4] Darcy Best & Max Ward (2022): *A faster algorithm for maximum independent set on interval filament graphs*. *J. Graph Algorithms Appl.* 26(1), pp. 199–205, doi:10.7155/JGAA.00588.

[5] Johannes Blum & Frank Drewes (2019): *Language theoretic properties of regular DAG languages*. *Inf. Comput.* 265, pp. 57–76, doi:10.1016/j.ic.2017.07.011. Available at `https://doi.org/10.1016/j.ic.2017.07.011`.

[6] Tommaso Boccato, Matteo Ferrante, Andrea Duggento & Nicola Toschi (2024): *4Ward: A relayering strategy for efficient training of arbitrarily complex directed acyclic graphs*. *Neurocomputing* 568, p. 127058, doi:10.1016/J.NEUCOM.2023.127058.

[7] Dietrich Braess (1968): *Über ein Paradoxon aus der Verkehrsplanung*. *Unternehmensforschung* 12(1), pp. 258–268, doi:10.1007/BF01918335.

[8] David Chiang, Frank Drewes, Daniel Gildea, Adam Lopez & Giorgio Satta (2018): *Weighted DAG Automata for Semantic Graphs*. *Computational Linguistics* 44(1), doi:10.1162/COLI_a_00309.

[9] Marco Damonte & Shay B. Cohen (2018): *Cross-lingual Abstract Meaning Representation Parsing*. In: *Proceedings of NAACL*, doi:10.18653/v1/N18-1104.

[10] Marco Damonte, Shay B. Cohen & Giorgio Satta (2017): *An Incremental Parser for Abstract Meaning Representation*. In: *Proceedings of EACL*, doi:10.18653/v1/E17-1051.

[11] Martin Dias, Mariano Martinez Peck, Stéphane Ducasse & Gabriela Arévalo (2014): *Fuel: a fast general purpose object graph serializer*. *Softw. Pract. Exp.* 44(4), pp. 433–453, doi:10.1002/SPE.2136.

[12] Frank Drewes (2017): *DAG Automata for Meaning Representation*. In Makoto Kanazawa, Philippe de Groote & Mehrnoosh Sadrzadeh, editors: *Proceedings of the 15th Meeting on the Mathematics of Language, MOL 2017, London, UK, July 13-14, 2017*, ACL, pp. 88–99, doi:10.18653/v1/w17-3409.

[13] Frank Drewes (2017): *On DAG Languages and DAG Transducers*. *Bulletin of the EATCS* 121.

[14] Senka Drobac, Krister Lindén, Tommi A. Pirinen & Miikka Silfverberg (2014): *Heuristic Hyperminimization of Finite State Lexicons*. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asunción Moreno, Jan Odijk & Stelios Piperidis, editors: *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland, May 26-31, 2014*, European Language Resources Association (ELRA), pp. 3319–3324. Available at `http://www.lrec-conf.org/proceedings/lrec2014/summaries/784.html`.

[15] Vida Dujmovic & Pat Morin (2019): *Dual Circumference and Collinear Sets*. In Gill Barequet & Yusu Wang, editors: *35th International Symposium on Computational Geometry, SoCG 2019, June 18-21, 2019, Portland, Oregon, USA*, *LIPIcs* 129, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 29:1–29:17, doi:10.4230/LIPIcs.SoCG.2019.29.

[16] Anjan Dutta, Josep Lladós & Umapada Pal (2013): *A symbol spotting approach in graphical documents by hashing serialized graphs*. *Pattern Recognit.* 46(3), pp. 752–768, doi:10.1016/J.PATCOG.2012.10.003.

[17] Krister Lindén Erik Axelson, Sam Hardwick (2023): *HFST Training Environment and Recent Additions (61-69) pdf*, pp. 60–69. 2 of Hurskainen et al. [1]. Available at `http://hdl.handle.net/10062/89595`.

[18] Akio Fujiyoshi (2010): *Recognition of directed acyclic graphs by spanning tree automata*. *Theor. Comput. Sci.* 411(38-39), pp. 3493–3506, doi:10.1016/j.tcs.2010.06.006.

[19] Abdullah Gharaibeh, Elizeu Santos-Neto, Lauro Beltrão Costa & Matei Ripeanu (2013): *Efficient Large-Scale Graph Processing on Hybrid CPU and GPU Systems*. *CoRR* abs/1312.3018. arXiv:1312.3018.

[20] Annegret Habel & Hans-Jörg Kreowski (1986): *May we introduce to you: hyperedge replacement*. In Hartmut Ehrig, Manfred Nagl, Grzegorz Rozenberg & Azriel Rosenfeld, editors: *Graph-Grammars and Their Application to Computer Science, 3rd International Workshop, Warrenton, Virginia, USA, December 2-6, 1986*, *Lecture Notes in Computer Science* 291, Springer, pp. 15–26, doi:10.1007/3-540-18771-5_41.

[21] Tsutomu Kamimura & Giora Slutzki (1981): *Parallel and Two-Way Automata on Directed Ordered Acyclic Graphs*. *Information and Control* 49, pp. 10–51, doi:10.1016/S0019-9958(81)90438-1.

[22] Tsutomu Kamimura & Giora Slutzki (1982): *Transductions of Dags and Trees*. *Mathematical Systems Theory* 15(3), pp. 225–249, doi:10.1007/BF01786981.

[23] Krister Lindén, Tommi Pirinen et al. (2009): *Weighting finite-state morphological analyzers using hfst tools*. In: *Finite-State Methods and Natural Language Processing-FSMNLP 2009 Eight International Workshop*.

[24] Andreas Maletti & Daniel Quernheim (2011): *Optimal Hyper-Minimization*. *Int. J. Found. Comput. Sci.* 22(8), pp. 1877–1891, doi:10.1142/S0129054111009094.

[25] Akira Matsubayashi & Yushi Saito (2023): *A Faster Algorithm for Recognizing Directed Graphs Invulnerable to Braess's Paradox*. In Daniele Frigioni & Philine Schiewe, editors: *23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2023, September 7-8, 2023, Amsterdam, The Netherlands*, *OASIcs* 115, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 12:1–12:19, doi:10.4230/OASICS.ATMOS.2023.12.

[26] Sjur Nørstebø Moshagen, Flammie Pirinen, Lene Antonsen, Børre Gaup, Inga Mikkelsen, Trond Trosterud, Linda Wiechetek & Katri Hiovain-Asikainen (2023): *The GiellaLT infrastructure: A multilingual infrastructure for rule-based NLP*. 2 of Hurskainen et al. [1]. Available at `http://hdl.handle.net/10062/89595`.

[27] Flammie A Pirinen (2023): *Finite-State Technology in Rule-Based Natural Language Processing*, pp. 49–59. 2 of Hurskainen et al. [1]. Available at `http://hdl.handle.net/10062/89595`.

[28] Tommi Pirinen, Krister Lindén et al. (2010): *Finite-state spell-checking with weighted language and error models*. In: *Proceedings of LREC 2010 Workshop on Creation and use of basic lexical resources for less-resourced languages*.

[29] Daniel Quernheim & Kevin Knight (2012): *Towards Probabilistic Acceptors and Transducers for Feature Structures*. In: *Proc. 6th Workshop on Syntax, Semantics and Structure in Statistical Translation*, Association for Computational Linguistics, pp. 76–85.

[30] Eloize Rossi Marques Seno, Helena de Medeiros Caseli, Marcio Lima Inácio, Rafael T. Anchiêta & Renata Ramisch (2022): *XPTA: um parser AMR para o Português baseado em uma abordagem entre línguas*. *Linguamática* 14(1), pp. 49–68, doi:10.21814/lm.14.1.359.

[31] Daniel G. Swanson (2023): *Apertium*, pp. 95–111. 2 of Hurskainen et al. [1]. Available at `http://hdl.handle.net/10062/89595`.

[32] Ieva Vasiljeva, Sorcha Gilroy & Adam Lopez (2018): *The problem with probabilistic DAG automata for semantic graphs*. *CoRR* abs/1810.12266. arXiv:1810.12266.

[33] Magnus Wahlström (2017): *LP-branching algorithms based on biased graphs*. In Philip N. Klein, editor: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, SIAM, pp. 1559–1570, doi:10.1137/1.9781611974782.102.

[34] Douglas B. West (2001): *Introduction to Graph Theory*, 2 edition. Prentice Hall.

9 Appendix

Lemma 9.1 (Meta-state dependent)**.** *Let G, G' be prefix DAGs with $\underline{G} = \underline{G'}$ derived by a DAG grammar $\mathscr{G} = (N, \Sigma, R)$. Then, $\mathscr{G}$ can apply the rule $r \in R$ as the next derivation step $G \Rightarrow_r H$ iff $G' \Rightarrow_r H'$. Similarly, G is useful if and only if G' is useful. However, if G is language-useful this only implies that G' is useful.*

Proof of Theorem 3.3. By the definition of a DAG grammar, a rule application depends only on the temporary vertices labeled by non-terminals – elements of the meta-states $\underline{G}$ and $\underline{G'}$. Consequently, $G \Rightarrow_r H$ is possible iff $G' \Rightarrow_r H'$ is. Valuability requires a rule sequence and is therefore also dependent on the meta-state only. It follows that G is useful if and only if G' is useful. On the other hand, a common meta-state does not preserve connectivity. With the given equality $\underline{G} = \underline{G'}$, G could be a connected DAG while G' is not. In that case G is language-useful, but G' not necessarily – only if the preceding derivation connects the unconnected components of G'. But since the language-useful G is useful, G' is, too. □

Lemma 9.2 (Useful Prefix DAG)**.** *Let G be a prefix DAG and $\mathscr{G} = (N, \Sigma, R)$ be a DAG grammar without useless rules. If $\varnothing \Rightarrow_R^* G$, then G is* useful *with respect to $\mathscr{G}$, i.e. there exists a derivation $\varnothing \Rightarrow_R^* G \Rightarrow_R^* G'$ for a complete DAG G'. If G' is connected it is* language-useful*.*

Proof of Theorem 3.4. We prove this by induction on the length n of the derivation $\varnothing \Rightarrow_{r_1 \ldots r_n} G$. For the base case consider a derivation $\varnothing \Rightarrow_r G$ of length $n = 1$. As the rule set R contains no useless rules, there is a derivation $\varnothing \Rightarrow_{r'_1 \ldots r'_k} G'$ where $G' \in D_\Sigma$ and $r = r'_i$ for some i. Moreover, as $\varnothing \Rightarrow_r G$, the head of the rule r is the empty string and we may assume that $r = r'_1$. This means that G is useful with respect to $\mathscr{G}$ (for the sake of brevity, we say useful subsequently).

Consider now a derivation $\varnothing \Rightarrow_{r_1 \ldots r_n} G$ of length $n > 1$. Let $\rho_1 = r_1 \ldots r_{n-1}$. By the induction hypothesis we know that there is a sequence ρ_2 of rules such that $\varnothing \Rightarrow_{\rho_1 \rho_2} H$ for some $H \in D_\Sigma$. Moreover, as R contains no useless rules, there are rule sequences ρ'_1, ρ'_2, such that $\varnothing \Rightarrow_{\rho'_1 r_n \rho'_2} H'$ for some $H' \in D_\Sigma$. It is also possible to concatenate these two derivations and to interleave the individual derivation steps. This yields a derivation $\varnothing \Rightarrow_{\rho_1 \rho'_1 r_n} H^* \Rightarrow_{\rho_2 \rho'_2} (H \& H')$ where $H \& H'$ denotes the disjoint union of H and H'.

Consider now the derivation $\varnothing \Rightarrow_{\rho_1 r_n} G$. We want to show that G is useful. As ρ'_1 can be applied on the empty graph, there is some DAG G^* such that $\varnothing \Rightarrow_{\rho_1 r_n} G \Rightarrow_{\rho'_1} G^*$. Moreover, G^* and H^* have the same meta-state, as they were generated through the same multiset of rules. As H^* is useful it follows from Lemma 3.3, that G^* is useful, and therefore, G is useful.

Therefore, if G is connected, so is the connected component G'' in the complete DAG G' which finalized the prefix DAG G. This means that $\mathscr{G}$ can decide to choose only those derivation steps which yield a connected DAG $G' = G''$. Consequently, a connected prefix DAG G is not only useful but also language-useful with respect to its grammar $\mathscr{G}$. □

Lemma 9.3 (Unbounded without Rule Cycle)**.** *There exists a DAG grammar $\mathscr{G} = (N, \Sigma, R)$ and a vertex or edge label* u $\in \Sigma \cup N$ *such that the number of occurrences of* u *in a (derivation) DAG generated by $\mathscr{G}$ is unbounded, although* u *does not occur in any rule cycle.*

Proof of Theorem 4.5. Consider the DAG grammar $\mathscr{G} = (\{r, c, l\}, \{\texttt{r}, \texttt{c}, \texttt{l}\}, R)$ containing the following rules $R = \{\lambda \twoheadrightarrow \circled{\texttt{r}} \twoheadrightarrow rr,\ rr \twoheadrightarrow \circled{\texttt{c}} \twoheadrightarrow c, c \twoheadrightarrow \circled{\texttt{c}} \twoheadrightarrow l,\ l \twoheadrightarrow \circled{\texttt{l}} \twoheadrightarrow \lambda\}$ and let u $\in \{l, \texttt{l}\}$, thus let u either denote the edge label l or the vertex label $\texttt{l}$. Then, the vertex label $\texttt{l}$ cannot occur in a *marked rule* $\bar{\alpha} \twoheadrightarrow \circled{\sigma} \twoheadrightarrow \bar{\beta}$, which, by definition, comprises two marked nonterminals. On the contrary, the only rule with the label

$\mathtt{l}$ is $l \twoheadrightarrow (\mathtt{l}) \twoheadrightarrow \lambda$ with $|\alpha\beta| = |l\lambda| = 1$ and thus comprises only one nonterminal. Consequently, $\mathtt{l}$ cannot occur in a rule cycle since a rule cycle contains marked rules only. And, neither can the edge label $l \in N$. In a rule cycle, l would occur both in a head as well as in a tail which it does in R, but, again, the rule $l \twoheadrightarrow (\mathtt{l}) \twoheadrightarrow \lambda$ cannot take part in a rule cycle. Thus, by definition, u cannot participate in any of $\mathscr{G}$'s rule cycles.

The DAG $G \in L(\mathscr{G})$ in Figure 4a uses each rule $r \in R$ only once, just as both vertex label $\mathtt{l}$ and edge label l. We can take k disjoint isomorphic copies of G for any $k \in \mathbb{N}$ and connect them by swapping the copies of e, as shown in Figure 4c, by Theorem 2.11. The resulting DAG $G[e_{k-1} \bowtie e_k]^k$ is still accepted by $\mathscr{G}$ and connected. Moreover, it contains k occurrences of label u, which proves the lemma. □

Lemma 9.4 (Unbounded Label Occurrence)**.** *Let $\mathscr{G} = (N, \Sigma, R)$ be a minimal deterministic DAG grammar and* u $\in \Sigma \cup N$ *a label of a vertex or an edge. The number of occurrences of* u *in graphs $G \in L(\mathscr{G})$, or, for edge labels, in their corresponding derivation DAGs D, is unbounded, iff one of the two following conditions is fulfilled:*

a) Label u *occurs in some rule cycle of $\mathscr{G}$.*

b) There exist both a rule cycle c in which an unmarked $q \in N$ occurs as well as a rule path Π *between this nonterminal q and the label* u*.*

Proof of Theorem 4.6. Suppose that a) is true. Then $L(\mathscr{G})$ is infinite, according to Theorem 2.9, which in turn means that there is no bound on the length of (possibly undirected) paths in graphs $G \in L(\mathscr{G})$. To obtain paths of unlimited length by the pigeonhole principle the repetition of rules is needed since R is finite. We may do so by using the rule cycle c_u comprising u. The derivation $\emptyset \Rightarrow_R G' \Rightarrow^*_{c_u} \Rightarrow_R G$ does not impose a bound on the number of label u occurrences on Gs derived like that, which proves that a) implies unboundedness of u occurrences.

Suppose that b) is true. The cycle c with the unmarked state q generates not only the labels it comprises arbitrarily often, as showed in above paragraph, but also an unbounded number of the nonterminal q may appear when taking the intermediate graphs into account that are generated in the various steps to yield G. At every q the derivation steps $\Rightarrow_\Pi$ generate a useful DAG with respect to $L(\mathscr{G})$ with a path comprising u. But $\mathscr{G}$ does not bound the generations of q, thus neither on the derivation step $\Rightarrow_\Pi$ and consequently also the number of the labels us which shows that b) implies that the number of occurrences of u is not bounded.

Turning now to the second direction we assume that no bound on the number of us exists for the DAG language $L(\mathscr{G})$. This means that $\mathscr{G}$ can repeatedly generate u. With a finite number of rules R, this is only possible by applying rules with label u an unbounded number of times. Repetition of rules is obtained either by a rule cycle, a) or by rule paths to a cycle b). In every iteration of the cycle, also the rule path Π is repeated and like that our label u.

Obviously, rules that do not participate in any rule cycle, not having a path to a rule cycle cannot be used in a derivation multiple times, which proves the second direction. This result carries over without difficulty when regarding u as the edge label $u \in N$ in a derivation DAG D, instead of the vertex label $u \in \Sigma$ of a DAG $\lfloor D \rfloor = G$, which completes the proof. □