Single-Rooted DAGs in Regular DAG Languages: Parikh Image and Path Languages

Martin Berglund
Center for AI Research, CSIR
Dept. of Information Science
Stellenbosch University
pmberglund@sun.ac.za

Henrik Björklund
Dept. of Computing Science
Umeå University, Sweden
henrikb@cs.umu.se

Frank Drewes
Dept. of Computing Science
Umeå University, Sweden
drewes@cs.umu.se

Abstract

In a recent survey (Drewes, 2017) of results on DAG automata some open problems are formulated for the case where the DAG language accepted by a DAG automaton A is restricted to DAGs with a single root, denoted by $L(A)_u$. Here we consider each of those problems, demonstrating that: (i) the finiteness of $L(A)_u$ is decidable, (ii) the path languages of $L(A)_u$ can be characterized in terms of the string languages accepted by partially blind multicounter automata, and (iii) the Parikh image of $L(A)_u$ is semilinear.

1 Introduction

In many applications such as natural language processing (NLP) great value can be extracted from extending beyond the strings and trees which are most commonly offered by traditional language formalisms. In the other direction, however, moving to general graphs is a step too far for many NLP tasks, and directed acyclic graphs form a natural step between trees and general graphs, enforcing the hierarchical nature of sentences. For these reasons, the study of DAG automata, devices which accept a class of languages known as the regular DAG languages, have drawn some interest, see for example Blum and Drewes (2017); Chiang et al. (2016). (Here, we only consider the *ordered* case, but this makes no practical difference to the results we present, which apply equally to the unordered case.) In a recent survey (Drewes, 2017) the distinction between DAG automata which permit multiple roots (which is more natural in the way the formalism is defined) and ones restricted to a single root (which is natural from the perspective of many NLP applications) is considered at some length. The latter case is, somewhat surprisingly, significantly stronger, and the survey states four open questions relating to it. We consider all these questions here, with the following outcomes:

Problem 1. Given a DAG automaton A as input, is the problem of deciding whether $L(A)_u$ is finite decidable? We demonstrate that it is.

Problem 2. Is there a natural characterization of the path languages of the regular DAG languages restricted to a single root? We relate, in both directions, this class of languages to the string languages accepted by partially blind multicounter automata (a formalism akin to a string-accepting Petri net). The only caveat preventing a full equivalence is that a single-rooted DAG automaton simulating a multicounter automaton will contain some extraneous strings, which are, however, clearly labeled by an additional alphabet symbol.

Problem 3. This is a variation on Problem 2, asking for a characterization of the path languages of regular DAG languages intersected with regular languages. Intersection with a suitably chosen regular language makes the language equivalence with multicounter automata direct. In fact, as noted above, an intersection with a language of the form Σ^* is already sufficient to achieve this.

Problem 4. Are the Parikh images (see e.g. (Parikh, 1961)) of $L(A)$ and $L(A)_u$ semilinear for all DAG automata A? We demonstrate that they are.

These results provide further theoretical evidence for the claim by Chiang et al. (2016) and Drewes (2017) that the regular DAG languages exhibit an appropriate level of complexity for the formalization of linguistically reasonable sets of Abstract Meaning Representations (Banarescu et al., 2013) and that it, in particular, is not advisable to consider languages of the form L_u instead. It was already known (Blum and Drewes, 2017; Chiang et al., 2016) that the path languages of regular DAG languages have the desired property of being regular. Thanks to the results of this paper, we now also know that their Parikh images are semilinear, which is another highly desirable property. Moreover, while it was concluded in the above-mentioned papers that the single-root restriction yields too powerful path languages, we can now precisely characterize how powerful they actually are.

2 Preliminaries

For $n \in \mathbb{N}$, we write $[n]$ for the set $\{1, \ldots, n\}$. For a vector $I \in \mathbb{N}^n$, we write I_i for the ith element of I. The empty string is denoted ε. If f is an is a function from set S to set T, we extend it to finite sequences in S^* by letting $f(s_1 s_2 \cdots s_n) = f(s_1)f(s_2)\cdots f(s_n)$. Given a string language $L \subseteq S^*$ (or a regular expression denoting such a language) we let L_ε denote $L \cup \{\varepsilon\}$.

Given two strings u and v over alphabet Σ, the *shuffle* of u and v, written $u \odot v$, is the set of all interleavings of u and v. We define the shuffle inductively. The base case is $\varepsilon \odot u = u \odot \varepsilon = \{u\}$, for every string $u \in \Sigma^*$. If $u = au'$ and $v = bv'$, with $a, b \in \Sigma$, then $u \odot v = a(u' \odot v) \cup b(u \odot v')$. For languages L_1 and L_2 we have $L_1 \odot L_2 = \{u \odot v \mid u \in L_1 \wedge v \in L_2\}$. The *shuffle closure* of a language L, written $L^\odot$, is the set of interleavings of zero or more strings from L, i.e., $L^\odot = \bigcup_{i=0}^{\infty} L^{\odot_i}$, where $L^{\odot_0} = \{\varepsilon\}$ and $L^{\odot_{i+1}} = L \odot L^{\odot_i}$.

The graphs we are interested in here are directed, node-labeled, and ordered, in the sense that for each node, there is an order on the incoming and on the outgoing edges from the node. We formally define them as follows.

Definition 1. Let Σ be an alphabet. A *graph* over Σ is a tuple $G = (V, E, in, out, lab)$, where V and E are the sets of nodes and edges, respectively, $in : V \to E^*$ and $out : V \to E^*$ assign incoming and outgoing edges to the nodes, and $lab : V \to \Sigma$ assigns labels to the nodes. An edge goes from precisely one node to precisely one, that is, $|in^{-1}(e)| = |out^{-1}(e)| = 1$ for all $e \in E$.

A (directed) path in such a graph is a sequence $v_0 e_1 v_1 e_2 \cdots v_n$ such that, for each $i \in [n]$, e_i points from v_{i-1} to v_i, i.e., it occurs in both $out(v_{i-1})$ and $in(v_i)$. Such a path is a *cycle* if $v_0 = v_n$ and $n > 0$. A graph is a *DAG* (directed acyclic graph) if it has no cycles.

We next give the definition, taken from (Blum and Drewes, 2017; Chiang et al., 2016), of the automata on DAGs that interest us.

Definition 2. A *DAG-automaton* is a structure $A = (Q, \Sigma, R)$, where Q is a finite set of states, Σ is an alphabet, and R is a finite set of rules. Each rule in R has the form $\alpha \overset{\sigma}{\leftrightarrow} \beta$, where $\alpha, \beta \in Q^*$ and $\sigma \in \Sigma$.

For a DAG $G = (V, E, in, out, lab)$ over Σ a run of the automaton $A = (Q, \Sigma, R)$ on G is an assignment $\rho\colon E \to Q$ of states to the edges of G such that

$$\rho(in(v)) \xleftrightarrow{lab(v)} \rho(out(v)) \in R$$

for every node $v \in V$. A DAG G is accepted by A if there is a run of A on G. The language consisting of all connected DAGs that are accepted (or recognized) by A is denoted by $\mathcal{L}(A)$. A DAG language that can be recognized by a DAG automaton is called a regular DAG language.

Note that we include only connected DAGs in $L(A)$. Thus, in the following we implicitly assume that all DAGs we consider are connected. This is reasonable because a DAG is accepted by A if and only if all its connected components are; for details, see (Drewes, 2017).

A node in a DAG is a *root* if it has no incoming edges. Given a DAG language L, we write L_u for the set of all graphs in L that are single-rooted, i.e., that have exactly one root. A node is a *leaf* if it has no outgoing edges.

Definition 3. A *partially blind multicounter automaton* is a machined defined by a tuple $M = (Q, \Sigma, m, \delta, q_0, F)$ where, as in ordinary (non-deterministic) finite-state automata, Q is the finite set of states, Σ is the input alphabet, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. Furthermore, $m \in \mathbb{N}$ is the number of counters, and $\delta \subseteq (Q \times \Sigma_\varepsilon \times \mathbb{N}^m \times \mathbb{N}^m \times Q)$ is the transition relation.

The counters are numbered from 1 to m and take values from $\mathbb{N}$. The automaton starts reading a string in state q_0 and with all counters set to 0. When the automaton makes a transition (p, σ, D, I, q), the vector D is first subtracted from the counter values. If this results in some counter having a negative value, the transition and the computation fail (i.e. the transition cannot be taken). Otherwise, I is added to the counters, σ is read, and the automaton changes state to q.

The automaton accepts $w \in \Sigma^*$ if, after reading the string, it can reach a configuration with an accepting state and all counters set to 0. In this case, w belongs to the language $\mathcal{L}(M)$ accepted (or recognized) by M.

Note that there is no explicit zero-testing during the computation, only at the end. This is the "partial blindness". As a consequence, the automata, as opposed to 2-counter automata with zero tests, are not Turing complete. Instead, they can be seen as string-accepting vector addition systems with states (VASS) or Petri nets (Greibach, 1978). Among other things, this means that the emptiness problem is decidable for these machines (Kosaraju, 1982; Mayr, 1984).

Since we are only interested in partially blind multicounter automata here, we will simply refer to them as multicounter automata.

3 Semilinearity and finiteness

Let Σ be an alphabet and let $\sigma_1, \sigma_2, \ldots, \sigma_n$ be an (arbitrary) ordering of its elements. If s is a string over Σ, the *Parikh image* of s is a vector $\psi(s)$ such that $\psi(s)_i$ is the number of occurrences of σ_i in s for every $i \in [n]$. Similarly, for a DAG G, $\psi(G)_i$ is the number of occurrences of σ_i as the label of a node in G. For a language L, over strings or graphs, $\psi(L)$ is the set $\{\psi(s) \mid s \in L\}$.

A subset X of $\mathbb{N}^n$ is *linear* if there are vectors $v_0, v_1, \ldots, v_n$ such that

$$X = \{v_0 + i_1 v_1 + \cdots i_n v_n \mid i_1, \ldots, i_n \in \mathbb{N}\}.$$

A subset of $\mathbb{N}^n$ is *semilinear* if it is the union of a finite number of linear sets. In the following, we are going to use the fact that the semilinear sets are effectively closed under intersection and projection to a subset of their dimensions. The former fact was proved by Ginsburg and Spanier (1964). The latter is easily seen from the definition of linear and semilinear sets.

Theorem 1. *For every regular DAG language L, the set $\psi(L)$ is semilinear.*

Proof. Let $A = (Q, \Sigma, R)$ be a DAG automaton such that $\mathcal{L}(A) = L$. Let $G = (V, E, \mathit{in}, \mathit{out}, \mathit{lab})$ be a DAG in L and let $\rho\colon E \to Q$ be a run of A on G. We describe a kind of linearization of such a run in the form of a string over the alphabet $\Gamma = \Sigma \cup Q \cup \overline{Q}$, where $\overline{Q} = \{\overline{q} \mid q \in Q\}$. Let the sequence $t = v_1, v_2, \ldots, v_n$ be a topological sorting of the nodes in G. For each node $v_i \in V$ we define $\mathit{in}_\rho(v_i)$ to be the sequence $\overline{\rho(\mathit{in}(v_i))}$, i.e., the sequence of states assigned to the incoming edges of v_i by ρ, but with each state replaced by its counterpart in $\overline{Q}$. Similarly, let $\mathit{out}_\rho(v_i)$ be the sequence $\rho(\mathit{out}(v_i))$, i.e., the sequence of states assigned to the outgoing edges from v_i by ρ. Then $\mathit{lin}_{G,\rho,t}$, the linearization of ρ on G (with respect to t) is given by $\mathit{lin}_{G,\rho,t} = w_1 \cdots w_n$ where

$$w_i = \mathit{in}_\rho(v_i)\mathit{lab}(v_i)\mathit{out}_\rho(v_i)$$

for all $i \in [n]$. For each rule $r = \alpha \overset{\sigma}{\leftrightarrow} \beta$, let $\mathit{string}(r) = \overline{\alpha}\sigma\beta$ and let $\mathit{string}(R) = \{\mathit{string}(r) \mid r \in R\}$. Then, for every DAG $G \in L$, topological sorting t of V_G, and run ρ of A on G, the string $\mathit{lin}_{G,\rho,t}$ belongs to the regular language $r_A = \mathit{string}(R)^*$.

Not every string in r_A is a linearization of a run of A on some DAG. We need to make sure that every state on an outgoing edge (represented by an element of Q) is matched by the same state on an incoming edge (represented by an element of $\overline{Q}$) and vice versa. For example, in the substring $\mathit{in}_\rho(v_1)\mathit{lab}(v_1)\mathit{out}_\rho(v_1)$ of $\mathit{lin}_{G,\rho,t}$, we must have that $\mathit{in}_\rho(v_1) = \varepsilon$, since v_1 must be a root of G and thus has no incoming edges. Let r_Q be the language $\{q\overline{q} \mid q \in Q\} \cup \Sigma$. Then $r_Q^\odot$ is the language over Γ such that for every $q \in Q$ and $w \in r_Q^\odot$, q and $\overline{q}$ appear the same number of times in w and every prefix of w contains at least as many instances of q as of $\overline{q}$.

We now argue that a string belongs to the language $r_A \cap r_Q^\odot$ if and only if it is a linearization of a run of A on some DAG. Consider a string $w = \alpha_1 \cdots \alpha_n \in \Gamma^*$. Let $O = \{i \in [n] \mid \alpha_i \in Q\}$ and $I = \{i \in [n] \mid \alpha_i \in \overline{Q}\}$. Then $w \in r_Q^\odot$ if and only if there exists a *matching*, i.e., a bijection $\mu\colon O \to I$ such that $\mu(i) > i$ and $\alpha_{\mu(i)} = \overline{\alpha_i}$ for all $i \in O$. This can be shown by a straightforward induction on n. With this in hand, for the "if" direction, a linearization will be in r_A by construction, and to demonstrate it is also in $r_Q^\odot$ a valid μ can be constructed along the edges of the DAG, as follows. By the definition of $\mathit{lin}_{G,\rho,t}$, if the topological sorting t of V is $v_1, \ldots, v_m$, then $w = w_1 \cdots w_m$ where $w_j = \mathit{in}_\rho(v_j)\mathit{lab}(v_j)\mathit{out}_\rho(v_j)$. Thus, every edge $e \in E$ leading from v_ℓ to $v_{\ell'}$ gives rise to an occurrence of $\rho(e)$ in w_ℓ and a corresponding occurrence of $\overline{\rho(e)}$ in $w_{\ell'}$. Let $i \in O$ and $i' \in I$ be the positions of these two occurrences in w, and define $\mu(i) = i'$. Since $\ell' > \ell$ we have $i' > i$, which means that μ is a matching because it is bijective by construction, and $\alpha_{\mu(i)} = \alpha_{i'} = \overline{\rho(e)} = \overline{\alpha_i}$.

For the "only if" direction one can similarly pick a matching μ (which exists as the string is in $r_Q^{\odot}$), and use the related positions as edges (each rule position forming one vertex), which produces a graph in L. More precisely, note first that $w = \alpha_1 \cdots \alpha_n \in r_A$ can uniquely be decomposed into $w = w_1 \cdots w_m$ such that $w_j \in \mathit{string}(R)$ for all $j \in [m]$. Define $V = \{v_1, \dots, v_m\}$ and let $\mathit{lab}(v_j)$ be the label in w_j, for $j \in [m]$. To define the edges, assume in addition that $w \in r_Q^{\odot}$ and let $\mu\colon O \to I$ be a corresponding matching. Then we define $E = \{e_i \mid i \in O\}$. Moreover, if $w_j = p_1 \cdots p_r \sigma \overline{q}_1 \cdots \overline{q}_s$, $c = |w_1 \cdots w_{j-1}|$, and $d = |w_1 \cdots w_j|$, then

$$\begin{aligned} \mathit{out}(v_j) &= e_{d-s} \cdots e_{d-1} \\ \mathit{in}(v_j) &= e_{\mu^{-1}(c+1)} \cdots e_{\mu^{-1}(c+r)}. \end{aligned}$$

Since $\mu(i) > i$ for all $i \in O$, this defines a DAG, and by construction defining $\rho(e_i) = \alpha_i$ for all $i \in O$ yields a run of A on $G = (V, E, \mathit{in}, \mathit{out}, \mathit{lab})$.

The next step is to note that $\psi(r_A \cap r_Q^{\odot})$ is a semilinear set. We know that r_A is regular and thus that $\psi(r_A)$ is semilinear (Parikh, 1961). Also, r_Q^* is regular and $\psi(r_Q^{\odot}) = \psi(r_Q^*)$, which shows that $\psi(r_Q^{\odot})$ is semilinear. This means that $\psi(r_A \cap r_Q^{\odot})$ is semilinear since the semilinear sets are closed under intersection.

Finally, to complete the proof, since the semilinear sets are closed under projection to any subset of their dimensions, $\psi(L)$, which is the projection of $\psi(r_A \cap r_Q^{\odot})$ onto the dimensions which correspond to Σ, is semilinear. □

The following corollary is obtained by combining the above with an observation in (Drewes, 2017) that states that $\psi(L_u)$ is semilinear whenever $\psi(L)$ is.

Corollary 1. *For every regular DAG language L, the set $\psi(L_u)$ is semilinear.*

The semilinearity of the Parikh image also gives us the following decidability result.

Corollary 2. *Given a DAG automaton A, it is decidable whether $\mathcal{L}(A)_u$ is finite.*

Proof. Let $R_{\text{root}} = \{(\alpha \overset{\sigma}{\leftrightarrow} \beta) \in R \mid \alpha = \varepsilon\}$ and $R_{\text{non-root}} = R \setminus R_{\text{root}}$. By replacing the regular language r_A in the proof of Theorem 1 by $R_{\text{root}} R^*_{\text{non-root}}$ one ensures that only runs on DAGs with exactly one root are considered. Since both intersection and projection are effective for semilinear sets, this gives us a way of effectively constructing the set $\psi(\mathcal{L}(A)_u)$. Since finiteness for semilinear sets is trivially decidable, the indicated result follows directly. □

4 Path languages

As seen in the proof of Theorem 1 above, the single-rooted regular DAG languages are intimately connected with string languages of the form $L \cap K^{\odot}$, where L and K are regular. These, in turn, are closely connected to multicounter automata, as shown by the following result, essentially due to Gisher (1981), explicitly stated, e.g., in (Björklund and Bojańczyk, 2007).

Theorem 2. *The following language classes are equal, modulo morphisms.*

1. *Languages recognized by multicounter automata.*

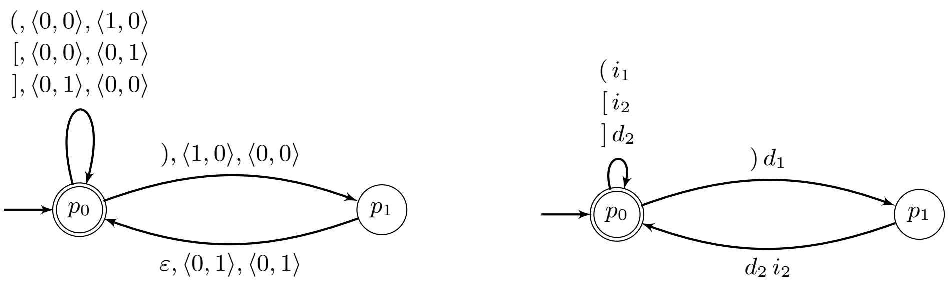
2. *Languages of the form $L \cap K^{\odot}$, where L and K are regular.*

The statement remains valid if the language K is required to be finite.

To be precise, every language in class 2 also belongs to class 1 and every language in class 1 is the morphic image of a language in class 2. When representing the language of a multicounter automaton as a language from class 2, we sometimes need to include some bookkeeping that represents the counter configurations. This bookkeeping can then be erased by a morphism.

Example 1. Consider the multicounter automaton at the top of Figure 1. The automaton has two counters. When the symbol (is read, counter one is increased by one. When [is read, counter two is increased by one. Conversely, when the symbol) or] is read, the respective counter is decreased by one. If there were only one state, and only these transitions, the automaton would accept the shuffle of the Dyck language over (and) and the Dyck language over [and]. The final ϵ-transition, however, first decreases counter two by one and then increases it by one, in effect ensuring that its value is not zero. This means that the language accepted by the automaton is the shuffle of the two Dyck languages, but with the restriction that an open (can be closed by) only if at least one [is still open.

In order to form a language of class 2 that captures the same behavior, we introduce the four new



$(, \langle 0,0\rangle, \langle 1,0\rangle$
$[, \langle 0,0\rangle, \langle 0,1\rangle$
$], \langle 0,1\rangle, \langle 0,0\rangle$
$), \langle 1,0\rangle, \langle 0,0\rangle$
$\varepsilon, \langle 0,1\rangle, \langle 0,1\rangle$
p_0
p_1
$(\, i_1$
$[\, i_2$
$]\, d_2$
$)\, d_1$
$d_2\, i_2$
p_0
p_1

Figure 1: The multicounter automaton (with its counter operations indicated in brackets) and the corresponding finite automaton from Example 1.

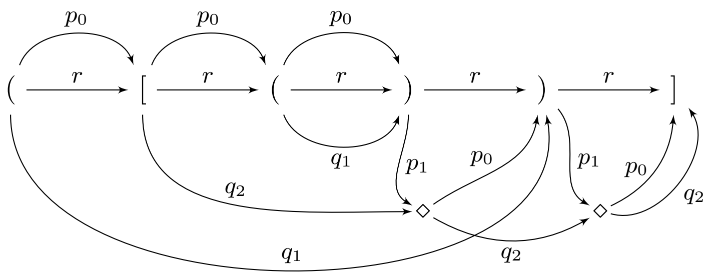


Figure 2: An example DAG of the form recognized by so-called special DAG automata. Its *main path*, consisting of the edges labeled with state r, corresponds to the input string processed by the multicounter automaton in Figure 1. The relationship will be formalized in Definition 4 and Lemma 2 (though for technical reasons we will require M to have dedicated initial and final states). Edges labeled with states q_1 and q_2 mimic counter actions. For example, an outgoing q_1 edge mimics incrementing counter 1 by 1, whereas an incoming q_1 edge decrements counter 1 by 1. The (off the main path) diamonds correspond to uses of the ε-transition from p_0 to p_1.

alphabet symbols i_1, i_2, d_1, d_2, representing increases and decreases to counters 1 and 2. The automaton to the right in Figure 1 reads an increase or decrease symbol each time it reads a parenthesis symbol. (Here, we let each transition read a string of symbols. The automaton can easily be transformed into a standard finite automaton by adding intermediate states.) Additionally, after reading a $)$ with its corresponding d_1, it reads first a d_2 and then an i_2. Let L be the language accepted by this finite automaton, $K = \{(\,), [\,], i_1d_1, i_2d_2\}$, and h the morphism that is the identity on the four parenthesis symbols while simply erasing the increase and decrease symbols. Then $h(L \cap K^{\odot})$ is the language of the original multicounter automaton.

Let us now turn to the connection between path languages and multicounter automata. A path $v_0e_1v_1e_2 \cdots v_n$ in a DAG is a *full path* if v_0 is a root and v_n is a leaf. The string corresponding to a path $v_0e_1v_1e_2 \cdots v_n$ is $lab(v_0 \cdots v_n)$. Given a DAG G and an alphabet Σ, we write $\pi_\Sigma(G)$ for the set of all strings in Σ^* corresponding to full paths in G. For a DAG language L, $\pi_\Sigma(L)$ is the set $\{\pi_\Sigma(G) \mid G \in L\}$.

As the main result of this section (stated formally in Theorem 3 below), we want to prove that the languages $\pi_\Sigma(\mathcal{L}(A)_u)$ for DAG automata A are exactly the languages $\mathcal{L}(M) \setminus \{\varepsilon\}$ accepted by multicounter automata M. For this, we show first that A can be brought into a special form which accepts only DAGs which contain a unique "main" path labeled with a string accepted by M, with nodes not on this path used for bookkeeping and corresponding to ε transitions in M. An example graph which is consistent with this form is previewed in Figure 2.

For the remainder of this section, fix an alphabet Σ, and let $\Sigma_\diamond = \Sigma \cup \{\diamond\}$ where $\diamond$ is a special symbol not in Σ. A DAG $G = (V, E, in, out, lab)$ over $\Sigma_\diamond$ is *special* if it has a unique full path with all node labels in Σ and all nodes not on this path are labeled $\diamond$. We call this path the *main path* of G. Consider a DAG automaton $A = (\{r\} \cup P \cup Q, \Sigma_\diamond, R)$, where $P \cap Q =$

$\{r\} \cap (P \cup Q) = \emptyset$. The DAG automaton A is *special* if all rules $\alpha \overset{\sigma}{\leftrightarrow} \beta$ are such that either $\alpha, \beta \in rPQ^* \cup \{\varepsilon\}$ and $\sigma \in \Sigma$ or $\alpha, \beta \in PQ^*$ and $\sigma = \diamond$. We call rules of the former kind *path-generating rules* and those of the latter kind *bookkeeping rules*. Note that special DAG automata recognize languages of special DAGs. A run of such a DAG automaton assigns state r to the edges on the path whose nodes carry labels in Σ (the leftmost path) while all other nodes are labeled by $\diamond$.

Lemma 1. *Let $A = (Q, \Sigma, R)$ be a DAG automaton and $\Sigma_0 \subseteq \Sigma$. Then there is a special DAG automaton A' such that*

$$\pi_{\Sigma_0}(\mathcal{L}(A)_u) = \pi_\Sigma(\mathcal{L}(A')_u).$$

Proof. Let p, r be distinct states not in Q and define $P = \{p\}$. Let $h\colon \{r, p\} \cup Q \to Q$ be the homomorphism such that $h(q) = q$ for all $q \in Q$ and $h(r) = h(p) = \varepsilon$. Let R' be defined as follows: for every rule $\alpha \overset{\sigma}{\leftrightarrow} \beta$ in R, R' contains the bookkeeping rule $p\alpha \overset{\diamond}{\leftrightarrow} p\beta$. If $\sigma \in \Sigma_0$, then R' furthermore contains all path-generating rules $\alpha' \overset{\sigma}{\leftrightarrow} \beta'$ such that $h(\alpha') = \alpha$, and $h(\beta') = \beta$. If a bookkeeping rule obtained from $\alpha \overset{\sigma}{\leftrightarrow} \beta$ is applied in a run, we say that the label $\diamond$ of the corresponding node is a *hidden σ*.

By construction $A' = (\{r\} \cup P \cup Q, \Sigma_\diamond, R')$ is special. To see that $\pi_{\Sigma_0}(\mathcal{L}(A)_u) = \pi_\Sigma(\mathcal{L}(A')_u)$, we consider the two inclusions. Clearly, if ρ is a run of A' on a special DAG G' with $\pi_\Sigma(G') = \{w\}$, then $w \in \Sigma_0^*$ and by deleting all edges with states r or p and replacing every hidden σ by σ we obtain a run of A on a DAG G with $w \in \pi_{\Sigma_0}(G)$.

For the other direction, consider a DAG $G = (V, E, in, out, lab) \in \mathcal{L}(A)_u$ with an accepting run ρ and let $v_0 e_1 v_2 e_2 \cdots v_m$ be a full path with $lab(v_0 \cdots v_m) \in \Sigma_0^*$. Let $u_0, \ldots, u_n$ be a topological sorting of V such that $u_n = v_m$ (and $u_0 = v_0$ because v_0 is the unique root). We define a DAG G' by adding fresh edges $e_1, \ldots, e_n, e'_1, \ldots, e'_m$ as follows:

- e_i ($i \in [n]$) is added as the first outgoing edge of u_{i-1} and the first incoming edge of u_i, and
- afterwards, e'_j ($j \in [m]$) is added as the first outgoing edge of v_{j-1} and the first incoming edge of v_j.

Finally, the labels of all nodes not in $\{v_0, \ldots, v_m\}$ are replaced by $\diamond$. Now ρ can be extended to a run ρ' on G' by defining $\rho'(e_i) = p$ and $\rho'(e'_j) = r$ for all $i \in [n], j \in [m]$. $\square$

Next, we define the notion of *relatedness* between a (special) DAG automaton A with states in $\{r\} \cup P \cup Q$ and a multicounter automaton M with states in $P \cup \{q_0, q_f\}$. Each state in Q corresponds to a counter of M. Viewing a run of A as a top-down computation, the counter actions of m keep track of how many copies of each $q \in Q$ there are at the frontier of the computation. A path-generating rule $rpw \overset{\sigma}{\leftrightarrow} rp'w'$ of A relates to a transition of M that reads σ in state p and continues in state p'. The counter actions reflect how many times each state in Q occurs in w and w'. For example, the rule $rpq_1q_3q_1 \overset{\sigma}{\leftrightarrow} rp'q_1q_2$ would correspond to a transition from p to p' reading σ, with counter actions given by the vectors $(2, 0, 1)$ and $(1, 1, 0)$ (assuming that $Q = \{q_1, q_2, q_3\}$ and position i in the vectors corresponds to q_i). bookkeeping rules of A relate to ε-transitions of M in a similar way.

Definition 4. Let $A = (\{r\} \cup P \cup Q, \Sigma_\diamond, R)$ be a special DAG automaton and $M = (Q', \Sigma, m, \delta, q_0, \{q_f\})$ a multicounter automaton, where $Q = \{q_1, \ldots, q_m\}$ and $Q' = P \cup \{q_0, q_f\}$ with $q_0 \neq q_f$ and $\{q_0, q_f\} \cap P = \emptyset$.

Let ψ be the Parikh mapping associated with Q. A rule $\alpha \overset{\sigma}{\leftrightarrow} \beta$ in R and a transition (p, σ, D, I, p') in δ are *related* if the following hold:

If $\alpha \overset{\sigma}{\leftrightarrow} \beta$ is path-generating then $\sigma \in \Sigma$ and

- p is the unique state in P that occurs in α unless $\alpha = \varepsilon$ and $p = q_0$,
- p' is the unique state in P that occurs in β unless $\beta = \varepsilon$ and $p' = q_f$, and
- $D = \psi(h(\alpha))$ and $I = \psi(h(\beta))$.

If $\alpha \overset{\sigma}{\leftrightarrow} \beta$ is bookkeeping then $\sigma = \varepsilon$ and

- p and p' are the unique states in P that occur in α and β, respectively, and
- $D = \psi(h(\alpha))$ and $I = \psi(h(\beta))$.

A and M are related if for every rule in R there is a related transition in M and vice versa.

Note that relatedness between rules is injective in both directions. Thus, for every special DAG automaton A a related multicounter automaton M can be constructed, and the converse holds as well provided that the initial state of M has no incoming transitions, there is a single final state that has no outgoing transitions, and no initial or final transition is an ε-transition.

To simplify the technicalities of the proof of the next lemma, let us say that a *partial DAG* (pDAG, for short) is defined just like a DAG $G = (V, E, in, out, lab)$, except that $in^{-1}(e)$ may be empty for some edges $e \in E$, i.e., those edges are "dangling" without a target. A run of a DAG automaton on a pDAG is defined exactly as a run on a DAG. The notion of special DAGs carries over to the partial case in the obvious way.

Lemma 2. *If a special DAG automaton A and a multicounter automaton M are related, then $\mathcal{L}(M) = \pi_\Sigma(\mathcal{L}(A)_u)$.*

Proof. Let A, M, ψ, and h be as above. Note that each tuple in $\mathbb{N}^m$ can be seen as an assignment of values to the counters of M. Moreover, given a run ρ on a pDAG G and a subset $D = \{e_1, \ldots, e_\ell\}$ of the set of edges of G, we let $\psi_\rho(D) = \psi(h(\rho(e_1) \cdots \rho(e_\ell)))$.

For the inclusion $\mathcal{L}(M) \subseteq \pi_\Sigma(\mathcal{L}(A)_u)$, let $t_1, \ldots, t_{n+1} \in \delta$ be a sequence of transitions by which M accepts a string w. If $w = \sigma \in \Sigma$ then $t_1 = (q_0, \sigma, 0^m, 0^m, q_f)$ and R contains the path-generating rule $\varepsilon \overset{\sigma}{\leftrightarrow} \varepsilon$, thus accepting the DAG consisting of a single node labeled σ. Assume therefore that $n \geq 1$. We inductively construct runs $\rho_1, \ldots, \rho_n$ on single-rooted pDAGs $G_1, \ldots, G_n$ such that the following hold when M has executed transition t_ℓ ($\ell \in [n]$), where D_ℓ is the set of dangling edges of G_ℓ:

(i) D_ℓ contains exactly one edge with label r and exactly one edge e_ℓ with a label in P.

(ii) the counters of M store $\psi_{\rho_\ell}(D_\ell)$, and

(iii) the prefix v of w read by M so far labels the main path in G_ℓ, and $\rho_\ell(e_\ell)$ is the current state of M.

Transition t_1 is of the form $t_1 = (q_0, \sigma, 0^m, I, p)$, which means that R contains a path-generating rule $\varepsilon \overset{\sigma}{\leftrightarrow} rp\alpha$ with $\psi(\alpha) = I$. Choosing G_1 to be the single-node pDAG to which this rule applies, and defining ρ_1 accordingly, obviously satisfies (i)–(iii). For $\ell > 1$, assume that $G_{\ell-1}$ and $\rho_{\ell-1}$ have been constructed.

There are two cases. If $t_\ell = (p, \sigma, D, I, p')$ (with $p, p' \in P$) then it is related to a path-generating rule $rp\alpha \overset{\sigma}{\leftrightarrow} rp'\beta$ in R, and by the induction hypothesis $\psi_{\rho_{\ell-1}}(D_{\ell-1}) \geq D$. Hence we can construct G_ℓ from $G_{\ell-1}$ by taking any sequence $s \in D_{\ell-1}$ of pairwise distinct edges such that $\rho_{\ell-1}(s) = rp\alpha$, and adding a fresh node v labeled σ with $in(v) = s$. Further, add $|\beta| + 2$ dangling outgoing edges to v, and let ρ_ℓ be the extension of $\rho_{\ell-1}$ obtained by setting $\rho_\ell(out(v)) = rp'\beta$. Clearly, this satisfies (i)–(iii) with e_ℓ being the second outgoing edge of v. The case where $t_\ell = (p, \varepsilon, D, I, p')$ is is similar, except that the rule in R to which it is related is a bookkeeping rule, and thus the label of v is $\diamond$ and it lacks incoming and outgoing edges whose state is r. Again, (i)–(iii) are clearly satisfied.

Thus we have shown that G_n satisfies (i)–(iii). We also know that $t_{n+1} = (p, \sigma, D, 0^m, q_f)$ and is thus related to a path-generating rule $rp\alpha \overset{\sigma}{\leftrightarrow} \varepsilon$. Constructing G_{n+1} similarly to the path-generating case above, but the outgoing dangling edge thus ensures that G_{n+1} contains a path labeled w from the root to the newly added node, which is a leaf. Note that G_{n+1} does not contain any dangling edges because M accepts only with all counters zero. Consequently, $G_{n+1} \in \mathcal{L}(A)_u$ and thus $w \in \pi_\Sigma(\mathcal{L}(A)_u)$.

For the other inclusion, note that every run ρ on a DAG G with $n + 1$ nodes $v_1, \ldots, v_{n+1}$ arranged in topological order can be decomposed into runs $\rho_1, \ldots, \rho_{n+1}$ on pDAGs $G_1, \ldots, G_{n+1}$ such that G_ℓ is the restriction of G to $v_1, \ldots, v_\ell$ and their outgoing edges, and ρ_ℓ is the restriction of ρ to G_ℓ. Now, given a root-to-leaf path in G labeled w it should be clear how transitions $t_1, \ldots, t_{n+1}$ related to the rules in R applied to $v_1, \ldots, v_\ell$ can be chosen to mirror the arguments above, yielding a computation of M that accepts w. □

Theorem 3. *A string language $L \subseteq \Sigma^* \setminus \{\varepsilon\}$ can be recognized by a multicounter automaton M if and only if $L = \pi_\Sigma(L_u)$ for a regular DAG language L.*

Proof. By Lemmas 1 and 2 it suffices to show that L can be recognized by a multicounter automaton M' which is related to some special DAG automaton, i.e., the initial state of M' has no incoming transitions, there is a single final state that has no outgoing transitions, and no initial or final transition is an ε-transition. The first two requirements are easily ensured by adding two new states, an initial and a final one. Making sure that the third requirement is met is also a straightforward matter since $\varepsilon \notin L$. □

The relationship between multicounter automata and path languages of single-rooted regular DAG languages thus resembles that between multicounter automata and languages of the form $L \cap K^{\odot}$. Theorem 3 shows that for every DAG automaton A, the language $\pi(\mathcal{L}(A)_u)$ can be recognized by a multicounter automaton, the other direction needs some bookkeeping that must be removed by intersecting $\pi(\mathcal{L}(A)_u)$ with Σ^*. In this direction, we also have to exclude the empty string as full paths in DAGs are always nonempty.

An interesting consequence of Theorem 3 is that the path languages of single-rooted regular DAG languages are orthogonal to the Chomsky hierarchy. While multicounter automata can recognize all regular languages and some languages that are not context-free, such as, e.g., $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, they cannot recognize all context-free languages. For instance, they can recognize the language $\{a^n b^n \mid n \in \mathbb{N}\}$, but not its Kleene closure, which is also context-free (Greibach, 1978). While the multicounter languages are closed under union, intersection, concatenation, and shuffle, they are thus not closed under Kleene star; for details, see, e.g., (Priese and Wimmel, 2008). Their Parikh images are semilinear, which for example follows from Theorem 3.

5 Conclusions

To summarize, each of the open problems given in (Drewes, 2017) (and recalled in the introduction) has been solved. Corollaries 1 and 2 close problems 4 and 1 respectively. Problem 3 is closed by Theorem 3, directly relating the languages captured by multicounter automata to the path languages in question. For problem 2 the characterization by multicounter automata very naturally models the computational power of the formalism itself, but the language equivalence involves projecting away some bookkeeping information by intersection with Σ^*.

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proc. 7th Linguistic Annotation Workshop, ACL 2013 Workshop*.

Henrik Björklund and Mikołaj Bojańczyk. 2007. Shuffle expressions and words with nested data. In *Mathematical Foundations of Computer Science (MFCS)*. pages 750–761.

Johannes Blum and Frank Drewes. 2017. Language theoretic properties of regular DAG languages. *Information and Computation* (to appear).

David Chiang, Frank Drewes, Daniel Gildea, Adam Lopez, and Giorgio Satta. 2016. Weighted DAG automata for semantic graphs. Unpublished manuscript.

Frank Drewes. 2017. On DAG languages and DAG transducers. *Bulletin of the EATCS* 121.

Seymour Ginsburg and Edwin H. Spanier. 1964. Bounded Algol-like languages. *Transactions of the American Mathematical Society* 113(2):333–368.

Jay Gisher. 1981. Shuffle languages, petri nets, and context-sensitive grammars. *Communications of the ACM* 24(9):597–605.

Sheila A. Greibach. 1978. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science* 7:311–324.

S. Rao Kosaraju. 1982. Decidability of reachability in vector addition systems. In *ACM Symposium on Theory of Computing (STOC)*. pages 267–281.

Ernst W. Mayr. 1984. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing* 13(3):441–460.

Rohit J. Parikh. 1961. Language generating devices. Quarterly Progress Report 60, M.I.T.

Lutz Priese and Harro Wimmel. 2008. *Petri-Netze*, Springer, chapter Petri-Netz-Sprachen. 2nd edition.