A Unifying Approach to Picture Automata

Yvo Ad Meeres[1,*], František Mráz[2,*]

[1]University of Bremen, Department of Theoretical Computer Science, Bibliothekstr. 5, 283 59 Bremen, Germany
[2]Charles University, Department of Software and Teacher Training, Malostranské nám. 25, 118 00 Prague 1, Czech Republic

Abstract

A directed acyclic graph (DAG) can represent a two-dimensional string or picture. We propose recognizing picture languages using DAG automata by encoding 2D inputs into DAGs. An encoding can be input-agnostic (based on input size only) or input-driven (depending on symbols). Three distinct input-agnostic encodings characterize classes of picture languages accepted by returning finite automata, boustrophedon automata, and online tessellation automata. Encoding a string as a simple directed path limits recognition to regular languages. However, input-driven encodings allow DAG automata to recognize some context-sensitive string languages and outperform online tessellation automata in two dimensions.

Keywords

directed acyclic graph, DAG automaton, picture language, regular language

A picture can be represented as a rectangular array of symbols. Thus, pictures can be considered as the two-dimensional extension of strings. A picture language is then a set of pictures. While many different automata models that accept two-dimensional inputs were introduced (see [1]), there is still no class of such automata whose corresponding class of picture languages has properties similar to the class of regular languages.

Probably the closest class of picture languages with respect to its properties is the class of recognizable languages (REC), which are languages accepted by two-dimensional online tessellation automata (2OTA). Actually, each language from REC can be obtained as a projection of a local picture language, where a picture language is local if it is the set of all pictures having all subpictures of dimension two-by-two from a given finite set.

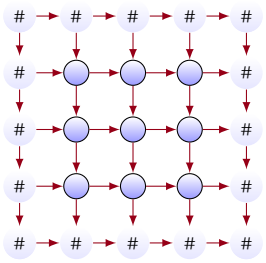
DAG automata introduced by Kamimura and Slutzki in [2] process single-rooted DAGs, just like the DAGs we propose for encoding strings and pictures. Discussions at a Dagstuhl Seminar on graph transformations with insights from Björklund and Maletti [3] gave rise to a variant of a DAG automaton subsequently introduced by Blum and Drewes [4]. A DAG automaton has a finite set of states, a finite alphabet, and a set of rules. For an input DAG, the DAG automaton assigns states to the edges of the input DAG and checks whether all nodes comply with the rules of the DAG automaton. In the positive case, the automaton accepts the input graph. Otherwise, it rejects. So, the DAG automaton, similarly to a 2OTA, checks only local properties (labeling of the incoming and outgoing edges for each node).

Regarding one-dimensional input, it is easy to represent a word (a string of symbols) as a graph. We can, e.g., take the set of positions in the string as the set of nodes labeled with the corresponding symbols of the string and connect them in a path from the first to the last. In two dimensions, we can represent a picture as a rectangular grid of nodes connected using horizontal and vertical edges. If we use oriented edges, a directed acyclic graph (DAG) representing a given picture, as illustrated in Fig. 1, can be built easily. We call such encodings input-agnostic, as the edges of the encoding DAG do not depend on the contents of the input.

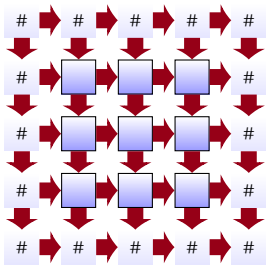
Simple input-agnostic DAG encodings for strings yield automata recognizing only regular languages, but for pictures, different input-agnostic encodings yield DAG automata corresponding to known 2D automata like returning finite automata, boustrophedon automata, and online tessellation automata.

We also introduce input-driven encodings of strings and pictures into DAGs, where a node's in-degree and out-degree depend on its label. Thus, DAG edges can vary between different pictures even if they share the same dimensions.

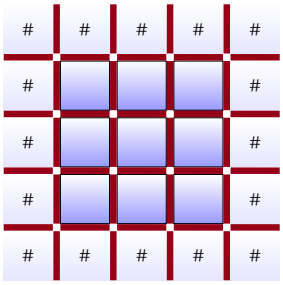
ITAT'25: Information Technologies – Applications and Theory, September 26–30, 2025, Telgárt, Slovakia
*Corresponding author.
Yvo.AdMeeres@mailbox.org (Y. Ad Meeres); frantisek.mraz@mff.cuni.cz (F. Mráz)



(a) ▦($\hat{P}$) encodes the picture $P \in \{\bigcirc\}^{3,3}$ as a DAG with the left upper corner as root.



(b) The picture DAG is drawn less traditionally but more like a rasterized picture.



(c) Visualizing the arrows defined by ▦ as colored borders lets the DAG look like $\hat{P}$.

Figure 1: The DAG encodes a picture P of dimensions $(3, 3)$ over the unary alphabet $\Sigma = \{\bigcirc\}$ as a *picture DAG* with the edges specified by the *DAG encoding* ▦. A DAG encodes a picture illustrated with vertices in the form of circles as shown in (a). A transition between the traditional illustration of graphs and rasterized pictures is the picture DAG in (b). The visualization of the DAG, omitting the orientation of the arrows is shown in (c).

Input-driven encodings enhance DAG automata's power, enabling recognition of some context-sensitive string languages and a proper super-class of REC in two dimensions.

DAG automata have many appealing properties. Blum and Drewes [4] proved that emptiness and finiteness are decidable for DAG automata, and the class of DAGs accepted by DAG automata is closed under union and intersection. These properties carry over to the case of accepting encodings of picture languages. Deterministic DAG automata can be reduced and minimized, and their equivalence is decidable in polynomial time [4]. Equivalence for deterministic DAG automata with respect to picture language recognition is still an open problem, as two deterministic DAG automata accepting the same encodings of pictures need not be equivalent, because they can accept different sets of DAGs that do not encode pictures. Ad Meeres [5] provides a framework to port known efficient finite state automata algorithms and properties from string to DAG languages. By applying these techniques, we could obtain classes of picture languages that can be efficiently recognized by DAG automata.

DAG automata have many strong properties [4]: emptiness and finiteness are decidable, and accepted DAG classes are closed under union and intersection, including for picture language encodings. We can minimize deterministic DAG automata, and their equivalence is polynomial-time decidable, though equivalence for deterministic DAG automata with respect to picture language recognition is still open. A recent framework by Ad Meeres [5] enables adapting efficient finite automata algorithms from strings to DAG languages.

The paper is structured as follows. After establishing the basic notation, Sect. 2 provides an overview of common automata accepting two-dimensional inputs. Sect. 2.2 presents a model of a DAG automaton suitable for recognizing pictures due to its limited capabilities. Sect. 3 introduces various encodings of pictures into DAGs. Sect. 4 compares DAG automata first to classical string automata to show the capabilities of DAG automata in the 1D case and then to the picture automata for the 2D case. The concluding Sect. 5 summarizes the obtained results and indicates open problems for future research. The majority of the proofs have been placed in the Appendix.

The paper is structured as follows. After establishing the basic notation, Sect. 2 provides an overview of common automata accepting two-dimensional inputs. Sect. 2.2 presents a model of a DAG automaton suitable for recognizing pictures due to its limited capabilities. Sect. 3 introduces various encodings of pictures into DAGs. Sect. 4 compares DAG automata first to classical string automata to show the capabilities of DAG automata in the 1D case and then to the picture automata for the 2D case. The concluding Sect. 5 summarizes the obtained results and indicates open problems for future research. The majority of the proofs appear in the appendix of the full version, available at [6].

1. Notation

For the positive integers $\mathbb{N}$ with $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, let $[n] = \{1, \dots, n\}$ and $[n]_0 = \{0, \dots, n\}$ where $n \in \mathbb{N}$. The cardinality of a set S equals $|S|$. A finite set S of *symbols* forms an *alphabet*. Concatenated symbols yield a *string*. The concatenation of two strings u and v is written as $u \cdot v$ or as their juxtaposition uv. For a string $w = w_1 w_2 \dots w_i \dots w_n \in S^*$ with $i \in [n]$, symbol $w_i \in S$ occurs at *position* i where $|w|$ denotes the length n of it, λ represents the empty string, $|w|_\sigma$ gives the number of occurrences of the symbol σ in w and $[w]$ denotes the smallest subset $T \subseteq S$ such that $w \in T^*$. All finite strings *over* S form the set S^*, with any subset of S^* called a *(string) language*, and $S^n \subset S^*$ comprising all strings over S of length n. A doubly ranked alphabet Σ is defined as an alphabet in which each symbol is assigned a rank through a function $r : \Sigma \to \mathbb{N}_0^2$.

An *automaton model* M is a computational model that defines how a specific instance of M, an automaton A, *recognizes* a language $L(A)$. Two automata models M and M' are considered equivalent, denoted by $M \equiv M'$, if they recognize the same class of languages. Throughout the paper, in an automaton definition, the following symbols do not require further definition when they appear: Σ for an input alphabet excluding the border signal $\# \notin \Sigma$, Q for a finite set of states, $q_0 \in Q$ for an initial state, $F \subseteq Q$ for a set of final states. A finite state automaton (FSA), in its nondeterministic variant (NFA) (or its deterministic one (DFA)) is defined as a tuple $A = (Q, \Sigma, \delta, q_0, F)$, where $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation (or a function in the case of DFA). The reflexive transitive closure $\delta^* \subseteq Q \times \Sigma^* \times Q$ is defined by: (i) for all $q \in Q$, $(q, \lambda, q) \in \delta^*$, and (ii) if $(q, a, q') \in \delta$ and $(q', w, q'') \in \delta^*$ for $a \in \Sigma$, $w \in \Sigma^*$, then $(q, aw, q'') \in \delta^*$. If $(q_0, w, q') \in \delta^*$ and $q' \in F$, A *accepts* $w \in \Sigma^*$. The automaton A recognizes the language $L(A)$ which is the set of all strings that A accepts.

2. Computational Models for Two Dimensions

Both rectangular arrays of symbols and other binary, i.e. 2-ary, relations can be regarded as two-dimensional structures. For both notions of two-dimensionality, automata have been introduced in the literature. Picture automata process two-dimensional matrices over an alphabet (see Sect. 2.1), while DAG automata digest finite multisets of an alphabet with a binary relation defined on them (see Sect. 2.2).

2.1. Picture Automata

Inspired from natural languages, a one-dimensional string is called a word. Inspired from image processing, a two-dimensional string or a matrix is called a *picture*. Both terms are standard in automata theory. A *picture automaton* recognizes a *picture language*.

We consider the three picture automaton models *returning* and *boustrophedon finite automaton* (RFA and BFA, Def. 2.2) and *two-dimensional online tessellation automaton* (2DOTA and 2OTA, Def. 2.5). We consolidate these fundamentally different picture automata under a single DAG-automaton model, providing a complete characterization for each individual automaton while simultaneously unifying them within a common framework, thereby capturing the full expressive power of each characterized picture automaton.

Notably, when restricted to one-row input pictures (i.e. in the one-dimensional case), each of these picture automaton models recognizes exactly the regular string languages.

Definition 2.1 ((Boundary) Picture)**.** *A* two-dimensional string P *over a finite alphabet* Σ *is called a* picture. *The* dimensions *of a picture* P *with* m *rows and* n *columns are given by the ordered pair* (m, n). *The symbol* Λ *denotes the unique* empty picture *with dimensions* $(0, 0)$. $\Sigma^{m,n}$ *represents the set of all pictures with dimensions* (m, n) *over* Σ. *The notation* $P_{i,j}$ *refers to the symbol from* Σ *at the* position $P(i, j)$ *of the picture* P*, where* $i \in [m]$ *denotes the* row *counted from top to bottom and* $j \in [n]$ *the* column *counted from left to right:*

$$P = \begin{matrix} P_{1,1} & \dots & P_{1,n} \\ \vdots & \ddots & \vdots \\ P_{m,1} & \dots & P_{m,n} \end{matrix}.$$

Any subset of the set of all pictures over Σ, denoted $\Sigma^{*,*}$, is called a picture language. *Given a picture $P \in \Sigma^{m,n}$, its* boundary picture $\hat{P}$ *is the picture P itself surrounded by the border symbol #, thus a picture over $\Sigma \cup \{\#\}$ with dimensions $(m+2, n+2)$ where the rows and columns range over $[m+1]_0$ and $[n+1]_0$, respectively. Let $\hat{P}_{i,j} = P_{i,j}$, for all $i \in [m]$ and $j \in [n]$, and $\hat{P}_{i,j} = \#$, for all $i \in [m+1]_0$ and $j \in [n+1]_0$, where $i \in \{0, n+1\}$ or $j \in \{0, m+1\}$.*

$$\begin{array}{|c|ccc|c|}\hline \# & \# & \cdots & \# & \# \\ \hline \# & & & & \# \\ \vdots & & P & & \vdots \\ \# & & & & \# \\ \hline \# & \# & \cdots & \# & \# \\ \hline \end{array} \quad = \quad \hat{P} \quad = \quad \begin{array}{ccccc} \hat{P}_{0,0} & & \cdots & & \hat{P}_{0,n+1} \\ \hat{P}_{1,0} & P_{1,1} & \cdots & P_{1,n} & \hat{P}_{1,n+1} \\ \vdots & & \ddots & & \vdots \\ \hat{P}_{m,0} & P_{m,1} & \cdots & P_{m,n} & \hat{P}_{m,n+1} \\ \hat{P}_{m+1,0} & & \cdots & & \hat{P}_{m+1,n+1} \end{array}$$

Both automata models defined below, RFA and BFA, were introduced by Fernau, Paramasivan, Schmid and Thomas [7]. They work as a finite state machine (as a DFA in the deterministic and an NFA in the nondeterministic case) equipped with a scanning strategy [8] which enables the automaton models to operate on two-dimensional input. The scanning strategy serializes a boundary picture $\hat{P}$ into a string $\hat{p}$ over $\Sigma \cup \{\#\}$. Afterward, the picture P is accepted if the corresponding FSA accepts this serialized picture $\hat{p}$. Both models scan their input picture horizontally line by line, according to their respective scanning strategy,but they differ in their scanning directions. An RFA proceeds uniformly from left to right, this corresponds to the reading of Western texts where the eye has to 'jump' from the end of a line to the beginning of the next one. A BFA, on the other hand, proceeds like an ox plowing a field, alternating its direction.

Definition 2.2 (RFA, BFA)**.** *Both a (deterministic)* returning finite automaton *(*RFA*) and a (deterministic)* boustrophedon finite automaton *(*BFA*) are given by a 6-tuple $A = (Q, \Sigma, \delta, q_0, F, \#)$, where $\delta \subseteq Q \times (\Sigma \cup \{\#\}) \times Q$ is the transition relation (transition function $\delta : Q \times (\Sigma \cup \{\#\}) \to Q$).*

Both automata models are equipped with a scanning strategy*, which defines a total order on the elements of a two-dimensional string. Formally, for a two-dimensional string S holds $S(i_1, j_1) \prec S(i_2, j_2)$ if and only if, in case of an*

- RFA : $(i_1 + j_1 < i_2 + j_2) \vee (j_1 = j_2 \wedge i_1 < i_2)$ *and in case of a*
- BFA : $(i_1 + j_1 < i_2 + j_2) \vee \big((i_1 + j_1 \bmod 2 = 0 \wedge j_1 > j_2) \vee (i_1 + j_2 \bmod 2 = 1 \wedge j_1 < j_2)\big)$.

With $M = (Q, \Sigma \cup \{\#\}, \delta, q_0, F)$ being an NFA *(*DFA*), the language L accepted by A is given as*

$$\begin{aligned} L(A) = \{\, P \in \Sigma^{m,n} \mid\ & \text{for all } i \in [m+1]_0 \text{ and } j \in [n+1]_0 \\ & s_1 s_2 \ldots s_k s_{k+1} \ldots s_{(m+2)(n+2)} \in L(M) \text{ with } \hat{P}_{i_1 j_1} \prec \hat{P}_{i_2 j_2} \text{ where } s_k = \hat{P}_{i_1 j_1} \text{ and } s_{k+1} = \hat{P}_{i_2 j_2} \,\} \end{aligned}$$

using the respective total order $\prec$ as specified above.

We presented an alternative, yet equivalent, definition of [7], which serves as a characterization emphasizing both the treatment of the boundary symbols as well as the scanning strategy. It is easy to see that our modified definition is correct since it does not change the expressive power of the automata models: every state computed within the top or bottom boundary could be computed within the first, resp. last row of the picture as well and two consecutive boundary symbols ## do not equip an automaton with expressiveness since it can do the same in one step while reading the squeezed single #, see below.

Here, we defined scanning strategies on a boundary picture with all four sides explicitly represented. The original definition, in contrast, delimits pictures only on the left and right and even then, it does not reference all those boundary symbols. Using boundary pictures with the full boundary, on all four sides, uniformly across all automaton models supports a coherent perspective and facilitates their systematic comparison in Sect. 4. By limiting the boundary symbols to acting as a 'fence' in the serialized string, we mimic the behavior of the original definition easily:

$$\begin{aligned} L(A) = \{\quad & P \in \Sigma^{m,n} \mid \text{ for all } i \in [m+1] \text{ and } j \in [n+1] \\ & s_1 s_2 \ldots s_k s_{k+1} \ldots s_{m\cdot(n+1)-1} \in L(M) \text{ with } \hat{P}_{i_1 j_1} \prec \hat{P}_{i_2 j_2} \text{ where } s_k = \hat{P}_{i_1 j_1} \text{ and } s_{k+1} = \hat{P}_{i_2 j_2} \quad \} \end{aligned}$$

Interpreting the strict order $\prec$ above as a partial order $\preceq$ by regarding consecutive positions labeled with # as equal, like $P(i, n+1) = P(i+1, 0)$, allows us to take canonical representatives of the resulting equivalence classes. We take the boundary symbols of the right border for both models as canonical representatives. In this way, in above language-defining formula, j no longer ranges over zero, excluding the left boundary. E.g., in case of an RFA, for every pair $P(i, n+1)$ and $P(i+1, 0)$, the canonical representative is $P(i, n+1)$ where $i \in [m+1]_0$. While the corresponding formalism may seem intricate at first glance, the underlying idea is entirely natural and aligns closely with the structure imposed by the scanning strategy, shown in Sect. 4.2.1. This partial order $\preceq$ mimics the original definition of RFA and BFA [7] where all vertices labeled # are present, but, only the canonical representative is referenced in a run which would be $P(i, n+1)$ for an RFA.

By making the scanning strategy explicit, our characterization reveals its effect on the family of languages recognized by the models. As a result of the explicit scanning strategy, Sect. 4.2.2 demonstrates that parametrizing it enables the recognition of a wider range of picture languages.

Theorem 2.3 (RFA ≡ BFA [7])**.** RFA *and* BFA *recognize the same picture language family.*

Theorem 2.4 (det. RFA ≡ nondet. RFA, det. BFA ≡ nondet. BFA [7, Lemma 14])**.** RFA *and* BFA *have the same expressive power in their deterministic and in their nondeterministic variant.*

The computation of an online tessellation automaton on a picture $P \in \Sigma^{*,*}$ consists of associating a state $q_{i,j} \in Q$ to each position $\hat{P}(i, j)$ depending only on the two states at the positions above and left from $\hat{P}(i, j)$.

Definition 2.5 (2OTA, 2DOTA [1])**.** *A nondeterministic (deterministic)* two-dimensional online tessellation automaton A *referred to as* 2OTA *(*2DOTA*), is defined by the five-tuple* $A = (\Sigma, Q, q_0, F, \delta)$ *where* $\delta : Q \times Q \times \Sigma \to 2^Q$ *(*$\delta : Q \times Q \times \Sigma \to Q$*) is a transition relation (function). A run of* A *on an input picture* $P \in \Sigma^{*,*}$ *is a function assigning a state* $q \in Q$ *to every position of its boundary picture* $\hat{P}(i, j)$*. The state of a run* ρ *at position* $\hat{P}(i, j)$ *is referenced as* $\rho(\hat{P}(i, j)) = q_{i,j}$*. A run* ρ *of* A *on a picture* P *assigns the initial state* q_0 *to all positions in the first column and the first row of its boundary picture* $\hat{P}$*. That is,* $q_{0,j} = q_{i,0} = q_0$*, for all* $i \in [m+1]_0$ *and* $j \in [n+1]_0$*. If* A *is deterministic,* ρ *assigns a state* $q_{i,j}$ *to each position* $P(i, j)$*, for* $i \in [m]$ *and* $j \in [n]$ *with* $q_{i,j} = \delta(q_{i-1,j}, q_{i,j-1}, P_{i,j})$*; if* A *is nondeterministic,* ρ *guesses all states such that* $q_{i,j} \in \delta(q_{i-1,j}, q_{i,j-1}, P_{i,j})$*. It accepts* P*, if there exists a run* ρ *such that* $q_{m,n} \in F$*.*

Here, we are interested in DAG automata. Blum and Drewes [4] showed that DAG automata have many desirable properties. The class of languages accepted by DAG automata is closed under union and intersection, but not under complementation. Emptiness and finiteness for languages accepted by DAG automata are decidable in polynomial time. Deterministic DAG automata can be minimized and tested for equivalence in polynomial time.

A deterministic DAG automaton (DDA) for strings encoded as strings DAGs corresponds exactly to an NFA that can only guess that it is at the end of the string. A DAG automaton is aware of the termination of the input earlier than a string automaton because it can infer that a symbol is the last one when it detects no outgoing edges, unlike a string automaton, which only realizes the word has ended after the last symbol has already been consumed. Thus, apart from this nondeterministic guessing of the end of a string, a DDA acts on a string DAG like a DFA on a string. However, we will add start and end markers to simplify the proof. Then, a one-dimensional DDA works precisely the same way as a DFA.

Theorem 2.6 ([7, 1])**.** *The picture automata* RFA, BFA, 2DOTA, *and* 2OTA *run on one-dimensional pictures* $P \in \Sigma^{m,1}$*, thus on strings, recognize exactly the regular string languages.*

2.2. Graph Automata

A graph is a set with a 2-ary relation on it. Consequently, we introduce DAG-automata for parsing two-dimensional strings. We limit our focus to DAGs due to the inherent orientation of binary tuples,

and the acyclic restriction offers algorithmic benefits. We choose the most limited model of a DAG automaton defined in the literature [4] to ensure that the model is not overly powerful. It operates on multigraphs with ordered edges.

Definition 2.7 (DAG). *A* directed acyclic graph over Σ, *abbreviated as* DAG, *is a tuple* (V, E, ℓ, in, out) *with* Σ, V, *and* E *being disjoint finite sets,* alphabet of vertex labels, *the sets of* vertices *and* edges, *respectively. The vertices are labeled by* $\ell : V \to \Sigma$. *An* edge $e \in E$ *joins two distinct vertices* $v, w \in V$. *The* source v *is referenced by* $src(e)$ *and the* target w *by* $tar(e)$[1]. *By* $in, out : V \to E^*$, *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* $out(v)$. *For the graphical representation of a graph, each vertex is drawn as a circle. The ingoing edges are placed along the upper half of the circle and the outgoing edges along the lower half, arranged from left to right according to the strings* $in(v)$ *and* $out(v)$.[2] *For every sequence* $e_1 e_2 \cdots e_n$ *of edges* $e_1, \ldots, e_n \in E$ *in a DAG with* $n \in \mathbb{N}$, *it holds that* $v_0 \neq v_n$ *in* $\{src(e_i), tar(e_i)\} = \{v_{i-1}, v_i\}$ *for all* $i \in [n]$, *meaning that the DAG is* acyclic. *The* empty graph $\emptyset$ *is the graph with* $V = \emptyset$. *A vertex is called a* root *if* $in(v) = \lambda$ *and a* leaf *if* $out(v) = \lambda$. *A DAG* $S = (\{v_0, \ldots, v_n\}, \{e_1, \ldots, e_n\}, \ell, in, out)$ encodes *a string* $\sigma_0\sigma_1 \ldots \sigma_n$ *as a* string DAG *if* $\ell(v_i) = \sigma_i$ *for* $i \in [n]_0$, $in(v_i) = e_i$ *for* $i \in [n]$ *and* $out(v_{i-1}) = e_i$ *for* $i \in [n]$.

Definition 2.8 (DAG Automaton). *A* DAG automaton *is a triple* $A = (Q, \Sigma, R)$, *where* R *is a finite set of* rules. *A state* $q \in Q$ *may also be called an* edge label. *A rule* $r \in R$ *is either of the form* $(\alpha \twoheadrightarrow \textcircled{\sigma} \twoheadrightarrow \beta)$, *where* $\sigma \in \Sigma$, *or of the form* $\emptyset$[3] *and the* head α *and the* tail β *are elements of* Q^*. *A* run *of* A *on a DAG* $G = (V, E, \ell, in, out)$ *is a mapping* $\rho : E \to Q$, *extended to strings* $\rho : E^* \to Q^*$ *by applying* ρ *component-wise to every edge* $e \in E$, *such that, for every vertex* $v \in V$, $\big(\rho(in(v)) \twoheadrightarrow \textcircled{\ell(v)} \twoheadrightarrow \rho(out(v))\big) \in R$. *A* accepts *a nonempty DAG* G *if such a run exists, and* A *accepts the empty DAG* $\emptyset$ *if* $\emptyset$ *is in* R. *The* DAG language recognized by A *is* $L(A) = \{\, G \mid A \text{ accepts } G \text{ and } G \text{ is connected} \}$.

The DAG automaton A *is called* top-down deterministic *if for every fixed combination of* $\sigma \in \Sigma$, $\alpha \in Q^*$, *and* $b \in \mathbb{N}_0$ *there exists at most one* $\beta \in Q^b$ *such that* $(\alpha \twoheadrightarrow \textcircled{\sigma} \twoheadrightarrow \beta) \in R$.[4] *A* rule cycle[5] *is a ring of rules (a sequence where the first rule is adjacent to the last) where in each pair of adjacent rules a state* $q \in Q$ *is connected to the same state* q, *once in a head and once in a tail. We abbreviate a top-down deterministic DAG automaton as* DDA *and a nondeterministic one as* NDA.

For a DAG automaton A, let $L_{\text{onerow}}(A)$ denote the set of string DAGs accepted by the DAG automaton A, and $\mathcal{L}_{\text{onerow}}(DAG) = \{L_{\text{onerow}}(A) \mid A \text{ is a DAG automaton}\}$ is the class of string languages that encoded as strings DAGs are accepted by DAG automata.

Theorem 2.9. $\mathcal{L}_{\text{onerow}}(DAG) = Reg$, *where* Reg *is the class of regular languages.*

3. Encoding Strings and Pictures as DAGs

How to encode a picture as a graph? The obvious idea is to represent all positions in a picture as vertices of a DAG. We call this a picture DAG. However, this gives us an arbitrary, even possibly disconnected DAG.

Definition 3.1 (Picture DAG). *A* picture DAG $G = (V, E, \ell, in, out)$ *encodes a two-dimensional string* S *with positions* $S(i, j)$ *and* $i, j \in \mathbb{N}$ *by interpreting the positions within* S *as the uniquely referenced vertices*

[1] Note that E can contain multiple edges connecting the same pair of nodes. Such distinct edges with same source and same target can even cross since the sources and targets are ordered.

[2] Hence, the ingoing edges appear clockwise and the outgoing edges counterclockwise around the circle.

[3] Adding the empty graph to the rule set allows consistency with the string languages but differs from the definition of DAG automata in [4].

[4] Intuitively, this means that for a vertex with fixed vertex label, fixed degree, and known ingoing edge labels only one labeling of the outgoing edges is allowed by R. In such a case, an input DAG permits at most one run, and this run can be constructed deterministically from the roots downward (the leaves upward).

[5] See [5] for a more formal definition.

of G with $V = \{S(i,j) \mid i \in [m] \text{ and } j \in [n]\}$, labeled as $\ell(S(i,j)) = S_{i,j}$. The set of edges is arbitrary (i.e., neither in, out nor E, is specified) unless specified by a picture-to-DAG encoding. The set of all picture DAGs over an alphabet Σ given by the context is denoted as $\mathcal{P} = \{G \mid G \text{ encodes a picture from } P \in \Sigma^{*,*}\}$; the set of the boundary pictures is $\hat{\mathcal{P}} = \{G \mid G \text{ encodes a picture } \hat{P}, \text{ where } P \in \Sigma^{*,*}\}$.

This definition implies that a picture DAG G encodes a picture P with dimensions (m,n) as a DAG with the set of vertices $V = \{P(i,j) \mid i \in [m] \text{ and } j \in [n]\}$, and its boundary picture $\hat{P}$ through $V = \{\hat{P}(i,j) \mid i \in [m+1]_0 \text{ and } j \in [n+1]_0\}$.

A DAG encoding specifies the set of edges for a picture DAG. One possible DAG encoding of a picture is illustrated in Figure 1. A DAG encoding determines the dependencies the automata can use while processing a picture.

Definition 3.2 (Picture-to-DAG Encoding)**.** *A* picture-to-DAG encoding *specifies the edges for a picture DAG. It is a function $\square : (\Sigma \cup \{\#\})^{*,*} \cup \mathcal{P} \cup \hat{\mathcal{P}} \to \mathcal{P} \cup \hat{\mathcal{P}}$ that maps its input to a picture DAG $G = (V, E, \ell, in, out)$ by specifying the set of edges E and the mappings in and out for G. The argument of the function $\square$ can be a (boundary) picture (the resulting DAG will have nodes corresponding to its positions) or a picture DAG (its original set of edges will be replaced with a new set of edges).*

The picture-to-DAG encoding specifies the edges either uniquely, resulting in one specific DAG G, or partially, which allows for several distinct DAGs, but the encoding maps the picture (DAG) to an arbitrary but fixed DAG G in this case. The symbol $\square$ denotes the abstract DAG encoding without edges specified. It stands for a specific DAG encoding defined below.

Note, this definition means that the picture-to-DAG encoding does not affect the presence of a boundary. A picture P is mapped to a picture DAG $G \in \mathcal{P}$ without boundary, whereas a boundary picture $\hat{P}$ is mapped to a picture DAG $\hat{G} \in \hat{\mathcal{P}}$ encoding additionally the boundary. The same holds for picture DAG input: the encoding does preserve the membership in $\mathcal{P}$ or $\hat{\mathcal{P}}$.

Picture automata use different scanning strategies, like row-by-row or diagonal-by-diagonal. These scanning strategies correspond roughly to the 'wiring' of the vertices in a DAG. However, not exactly, since scanning strategies impose a total order on the pixels of a picture, whereas a DAG imposes only a partial order on its vertices. All of the following picture-to-DAG encodings specify the sets of edges that depend only on the dimensions of input pictures; therefore, we call them *input-agnostic*.

Definition 3.3 (Input-agnostic picture-to-DAG encodings)**.** *The* input-agnostic DAG encoding *maps its input to a unique DAG $G = (V, E, \ell, in, out)$. In case the input is a picture DAG $G' = (V, E', \ell, in', out')$, the edges E', as well as their ordering and wiring specified by in' and out' are replaced by the DAG encoding. This allows for reencoding with another DAG encoding. The following sets of* input-agnostic edges *depend only on the dimensions (m,n) of the picture P to be encoded, but not on its symbols.*
For $i \in [m]$, $j \in [n]$, $i_0 \in [m]_0$, $j_0 \in [n]_0$, $\hat{i}, \hat{k} \in [m+1]_0$ and $\hat{j}, \hat{l} \in [n+1]_0$, we define a set of edges.

- $\hat{E}_{\downarrow\square} = \{e \mid src(e) = \hat{P}(i_0, 0), tar(e) = \hat{P}(i_0+1, 0)\}$ *are downward along the left border.*
- $\hat{E}_{\square\downarrow} = \{e \mid src(e) = \hat{P}(i_0, n+1), tar(e) = \hat{P}(i_0+1, n+1)\}$ *are downward the right border.*
- $\hat{E}_{\downdownarrows} = \{e \mid src(e) = \hat{P}(i_0, \hat{j}), tar(e) = \hat{P}(i_0+1, \hat{j})\}$ *are the vertical edges oriented downwards.*
- $\hat{E}_{\rightrightarrows} = \{e \mid src(e) = \hat{P}(\hat{i}, j_0), tar(e) = \hat{P}(\hat{i}, j_0+1)\}$ *are horizontal oriented to the right.*
- $\hat{E}_{rfa\swarrow} = \{e \mid src(e) = \hat{P}(i_0, n+1), tar(e) = \hat{P}(i_0+1, 0)\}$ *are the 'return' edges for the* RFA.
- *The horizontal edges oriented alternatingly to the left and to the right for the* BFA *are*
  $\hat{E}_{\rightleftarrows} = \{e \mid src(e) = \hat{P}(\hat{i}, j_0), tar(e) = \hat{P}(\hat{i}, j_0+1), i \text{ is odd}\}$
  $\quad \cup \{e \mid tar(e) = \hat{P}(\hat{i}, j_0), src(e) = \hat{P}(\hat{i}, j_0+1), i \text{ is even}\}$
  $\hat{E}_{\downarrow bfa\downarrow} = \{e \mid src(e) = \hat{P}(i_0, 0), tar(e) = \hat{P}(i_0+1, 0), i \text{ is odd}\}$
  $\quad \cup \{e \mid src(e) = \hat{P}(i_0, n+1), tar(e) = \hat{P}(i_0+1, n+1), i \text{ is even}\}$ *are vertical* BFA *edges.*
- $\hat{E}_{\searrow} = \{e \mid src(e) = \hat{P}(i_0, j_0), tar(e) = \hat{P}(i_0+1, j_0+1)\}$ *are the diagonals to the south east.*

Now we define several picture-to-DAG encodings by specifying their set of edges. We assume the corresponding in and out functions are defined in a natural way such that the edges do not cross. Furthermore, we

define $E_{all} = \{e \mid src(e) = u, tar(e) = v, u \neq v \text{ for all } u, v \in V\}$ *as the set of all possible edges for a picture, in order to get rid of the edges connected to the boundary in case the picture-to-DAG encoding maps a picture without a boundary to a DAG.*

- $\downarrow\!\!\rightrightarrows$ *defines* $E = \hat{E}_{\downarrow\square} \cup \hat{E}_{\rightrightarrows} \cap E_{all}$ *joining the leftmost vertices and all vertices in each row.*
- $\rightrightarrows\!\!\downarrow$ *defines* $E = \hat{E}_{\rightrightarrows} \cup \hat{E}_{\square\downarrow} \cap E_{all}$ *joining the rightmost vertices and all vertices in each row.*
- ▤ *defines* $E = \hat{E}_{\downarrow\square} \cup \hat{E}_{\rightrightarrows} \cup \hat{E}_{\square\downarrow} \cap E_{all}$ *joining the left and the right border and the rows.*
- $\rightleftarrows$ *defines* $E = \hat{E}_{\rightrightarrows} \cup \hat{E}_{rfa\swarrow} \cap E_{all}$ *as the* RFA *scanning strategy.*
- $\circlearrowleft$ *defines* $E = \hat{E}_{\rightleftarrows} \cup \hat{E}_{\downarrow bfa\downarrow} \cap E_{all}$ *as the* BFA *scanning strategy.*
- ▦ *defines* $E = \hat{E}_{\rightrightarrows} \cup \hat{E}_{\downdownarrows} \cap E_{all}$ *as a vertex pointing to two of its neighbours: below and right.*
- ▧ *defines* $E = \hat{E}_{\searrow} \cap E_{all}$ *as every vertex points to one neighbour, the one right below.*

Instead of using the total order of a scanning strategy or partial orders to define input-agnostic edges, we introduce edges depending on the input symbol at the picture's positions, called *input-driven.*

Definition 3.4 (Input-Driven picture-to-DAG encoding)**.** *Let* $G' = (V, E', \ell, in', out')$ *be a picture DAG over* Σ*, either encoding the input picture* P *or* $\hat{P}$*, with* $E' = \emptyset$ *and* $in' = out' = \lambda$*, or given as input itself. Then, the* input-driven DAG encoding *maps* G' *to an arbitrary but fixed DAG* $G = (V, E \cup E_t, \ell, in, out)$ over the doubly ranked alphabet (Σ, r)*. The edge set* E_t *denotes the set of* input-driven edges*. The ranks* $r(\ell(v)) = (r(\ell(v))_1, r(\ell(v))_2)$ *of the labels' vertices, with the vertices* $v \in V$*, determine the input-driven edges:*

- $in(v) = in'(v) \cdot e$ *with* $e \in E_t^*$ *and* $|e| = r(\sigma)_1$ *and all edges in* e *are pairwise distinct and*
- $out(v) = out'(v) \cdot e$ *with* $e \in E_t^*$ *and* $|e| = r(\sigma)_2$ *and all edges in* e *are pairwise distinct.*

The input-driven picture-to-DAG encoding is undefined if the above construction does not yield a DAG.[6]

Each edge in E_t lacks explicit designation of its source and target vertices, allowing for all combinations of joining the vertices according to the input-driven in- and outgoing specifications, as long as the combination yields a valid DAG. As such, this DAG encoding accommodates a variety of wiring combinations.

A swap operation is a fundamental operation on DAGs [4]. 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 from $tar(e_i)$ to $src(e_{1-i})$ for $i \in \{0, 1\}$. Then, the *edge swap* of e_0 and e_1 yields the DAG $G' = (V, E, \ell, h \circ in, out)$, where h is a bijection defined as follows:

$$h(e) = \begin{cases} e_{1-i} & \text{if } e = e_i \text{ for some } i \in \{0, 1\}, \\ e & \text{otherwise.} \end{cases}$$

A DAG automaton cannot be restricted to picture DAGs by its rule set. The rules allow accepting DAGs that do not represent pictures and are obtained from DAGs encoding pictures by the swap operation. For using DAG automata as a device recognizing picture languages, it is necessary to always have an intersection with the picture languages $\Sigma^{*,*}$.

Definition 3.5 (Equivalence $\equiv_{\square}$)**.** *Let* A *be a DAG automaton and* A' *be an automaton accepting a picture language* $L(A)$*. The automata* A *and* A' *are* picture equivalent *with respect to a picture-to-DAG encoding* $\square$*, denoted as* $A \equiv_{\square} A'$*, if* $L(A) \cap \square(\Sigma^{*,*}) = \square(L(A'))$*. If* $\mathcal{A}$ *is a family of DAG automata and* $\mathcal{A}'$ *is a family of picture automata, the families are equivalent with respect to* $\square$*, denoted as* $\mathcal{A} \equiv_{\square} \mathcal{A}'$ *if, for each DAG language* L*, it holds* $L \in \mathcal{L}(\mathcal{A}) \cap \square(\Sigma^{*,*}) \Leftrightarrow L \in \square(\mathcal{L}(\mathcal{A}'))$*.*

4. Comparison

This section provides a comparative analysis of string automata, designed for one-dimensional input (see Sect. 4.1), in relation to their capabilities vis-à-vis DAG automata, and picture automata, which operate in two dimensions (c.f. Sect. 4.2).

[6] It is thus defined if and only if the in- and outgoing edges are balanced: $\sum_{v \in V} r(\ell(v))_1 = \sum_{v \in V} r(\ell(v))_2$ and the newly added input derived edges do not introduce a cycle.

4.1. 1D: Comparison of String Automata with DAG Automata

Just as DAG automata recognize the regular string languages if the strings are encoded as string DAGs (Theorem 2.9), so do picture automata RFA, BFA, 2OTA, and 2DOTA when operating in one dimension only (Theorem 2.6). String DAG (Def. 2.7) joins vertices corresponding to the letters of a string in a simple path. But obviously, a node in a graph can have more than one incoming edge and one outgoing edge.

String DAGs and picture-to-DAG encodings from Def. 3.3 define edges connecting nodes representing symbols in a string or in a picture in an input-agnostic fashion. However, based on Def. 3.4, we can also add edges connecting symbols in an input-driven way. Since we use DAG automata for one- and two-dimensional strings, extra edges need to be added.

Edges are added either in an *input-agnostic* fashion [9] or an *input-driven* way [10], meaning that the edges depend either only on the dimensions of the input or also on the input itself. DAG encodings as defined in Def. 3.3 are input-agnostic.

Adding to a string DAG input-agnostic edges computed by FSAs does not seem to add power to DAG automata. An input-agnostic edge e corresponds to the length n of the substring $v_1 v_2 \dots v_n$ with $src(e) = v_1$ and $tar(e) = v_n$, because if an FSA is input-agnostic, it sees a unary language where the strings can only differ in their length. But a DDA on string DAGs accepts the same strings as a FSA. Every additional input-agnostic edge requires an FSA for a unary language. Since the DAG automata recognize graphs of bounded degree, only a constant number of input-agnostic edges are allowed, which in turn means that in each vertex, only a constant number of FSAs can start the computation of the spanning length of the input-agnostic edge. A vertex with both in and outgoing input-driven edges would use the final state for the unary FSA as starting state for the new unary FSA. This indicates that an FSA can, by power set construction, keep track of all FSAs computing the input-agnostic edges.

Thus, only a device more powerful than a FSA can exceed the regular languages by computing input-agnostic edges for the string. One example would be to add edges spanning n vertices where n is a power of two, a square, or a prime, requiring a stack, a linear bounded tape, or an infinite tape, respectively. A DAG automaton processing strings encoded with input-agnostic edges computed by machines more powerful than a mere FSA exceeds thus the power of the regular languages. Because input-agnostic edges require this trick of preprocessing with a more powerful device, we take a look at input-driven edges in the hope of avoiding devices more powerful than FSAs.

Input-driven edges do not require edges to be computed by a more powerful device than an FSA. Nevertheless, the power of DAG automata recognizing strings encoded with input-driven edges exceeds the class of regular languages. Such automata can accept, e.g., languages $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ and $L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ when input-driven edges are added so that the double rank r of the symbols is $r(a) = (0,1)$, $r(b) = (1,0)$, in case of L_1, and $r(b) = (1,1)$, in case of L_2, and $r(c) = (1,0)$.

Theorem 4.1. *DAG automata can recognize both certain context-free as well as context-sensitive languages if strings in the string languages are encoded as a string DAG with additional input-driven edges determined by a ranked alphabet.*

4.2. 2D: Comparison of Picture Automata with DAG Automata

The picture automata RFA and BFA are compared to DAG automata in Sect. 4.2.1, and based on that comparison, these two models are modified in Sect. 4.2.2. This modification in turn leads to comparing online tessellation automata to DAG automata in Sect. 4.2.3.

4.2.1. Comparison of RFA and BFA to DAG automata

Returning Finite Automata (RFA) and Boustrophedon Automata (BFA) are the most limited picture automata in literature and recognize the same family of picture languages (Theorem 2.3).

How can a DAG automaton model these two picture automata? Scanning strategies of RFA and BFA are total orders on the positions of the picture. Both scanning strategies are drawn as oriented graphs

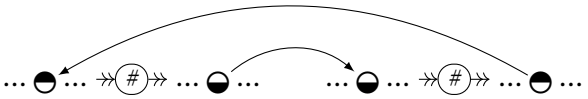
in Figure 2a. For the RFA the big jump back from right to left was viewed critically. Therefore, the BFA uses the alternating directions. Let us look at an example of how DAG automata process pictures.

Example 4.2. *Let $L_{\equiv}$ be the picture language of horizontal stripes where adjacent rows are of different colors. Formally, over a binary alphabet $\Sigma = \{○, ●\}$, this would be*

$$L_{\equiv} = \left\{ \begin{array}{l} a^n \\ b^n \\ a^n \\ b^n \\ \vdots \end{array} \;\middle|\; \{a, b\} = \{○, ●\} \text{ and } n \in \mathbb{N} \right\}.$$

Both RFA *and* BFA *scan the input horizontally, to recognize horizontal lines. Also, they can recognize stripes, as well as the above-defined stripe language $L_{\equiv}$, illustrated in Fig. 2a. And so can the DAG automaton, which simulates those two models by applying the encodings $\rightleftarrows$ and $\circlearrowleft$. This can be accomplished deterministically, since the scanning strategy is a serialization. The serialization of a picture yields a string, or, in the case of a DAG automaton, a string DAG, and apart from the scanning strategy, both an* RFA *and a* BFA *act like an* FSA *for which the nondeterminism adds no power, see Theorem 2.4.*

Abandoning the total order, we now consider encodings based on partial orders. We examine the encodings $\downarrow\rightrightarrows(\hat{P})$ and $\rightrightarrows\downarrow(\hat{P})$, which provide structured representations of an input picture P under relaxed ordering. We begin with $\downarrow\rightrightarrows(\hat{P})$. Consider a DAG automaton for the language $L_{\equiv}$ which uses the states ◓ and ◒ for encoding the alternation between two row colors in the edge labels. A rule cycle for the vertical edges ensures the alternating colors:



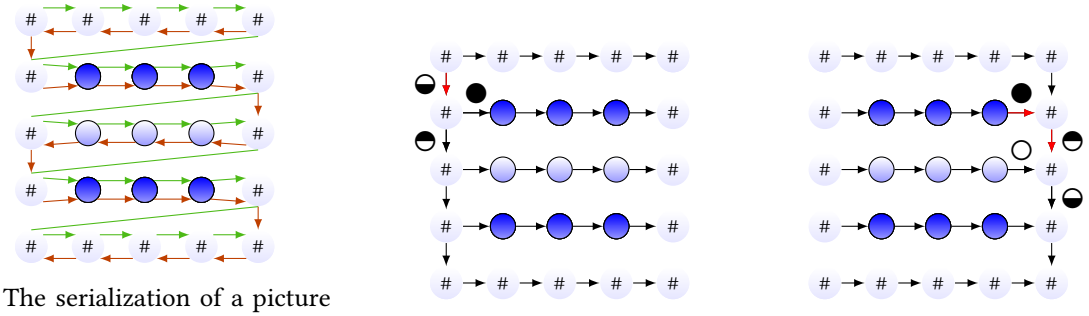
A DAG automaton $(\{◓, ◒, ●, ○, \#\}, \{●, ○\}, R_{root} \cup R_{⬓} \cup R_{\equiv} \cup R_{\#})$ for the encoding $\downarrow\rightrightarrows$ shown in Fig. 2b could label the edge path on the left border with the states ◒ and ◓ by a rule cycle alternating the rules $R_{⬓} = \{(◒ \twoheadrightarrow (\#) \twoheadrightarrow ◓○), (◓ \twoheadrightarrow (\#) \twoheadrightarrow ◒●)\}$. In this way, it can alternate between the colors on the left border and accept the lines in case they comply with the pattern memorized with the help of the states, vertical edge labels, at the left border. Additionally, it needs the rules $R_{\equiv} = \{(● \twoheadrightarrow (●) \twoheadrightarrow ●), (○ \twoheadrightarrow (○) \twoheadrightarrow ○), (● \twoheadrightarrow (●) \twoheadrightarrow \lambda), (○ \twoheadrightarrow (○) \twoheadrightarrow \lambda)\}$ for accepting the colored lines themselves, and the rules $R_{\#} = \{(◒ \twoheadrightarrow (\#) \twoheadrightarrow \#), (◓ \twoheadrightarrow (\#) \twoheadrightarrow \#), (\# \twoheadrightarrow (\#) \twoheadrightarrow \#), (\# \twoheadrightarrow (\#) \twoheadrightarrow \lambda)\}$ for $\hat{P}$'s lower border. The rules specified so far are top-down deterministic, but lack a specification for a root. The rule set of a nondeterministic DAG automaton may contain both rules in $R_{root} = \{(\lambda \twoheadrightarrow (\#) \twoheadrightarrow ◓\#), (\lambda \twoheadrightarrow (\#) \twoheadrightarrow ◒\#)\}$, which allows the automaton to recognize $L_{\equiv}$. In contrast, a DDA *lacks the ability to guess the initial color. Since, due to its determinism, a* DDA *is restricted to choosing one of the two root rules in R_{root} for its rule set. It cannot include both of them. But this hardcodes the color. Consequently, a deterministic DAG automaton using encoding $\downarrow\rightrightarrows$ can only recognize the stripe language starting with a fixed color. This is due to the left border being blind to the colors of the stripes when processing the picture top-down deterministically. The stripes depend on the left border but not vice versa. A* DDA *cannot recognize $L_{\equiv}$ encoded as $\downarrow\rightrightarrows$.*

However, with the right border vertically connected as in Fig. 2c, the vertical edges depend on the colors of the lines in a top-down deterministic run. Consequently, a DDA *can recognize all pictures $\rightrightarrows\downarrow(\hat{P}) \in L_{\equiv}$ rather than only those pictures P with a fixed initial color. The rules $(●◒ \twoheadrightarrow (\#) \twoheadrightarrow ◓)$ and $(○◓ \twoheadrightarrow (\#) \twoheadrightarrow ◒)$ assign the edge labels deterministically downward, thereby propagating the color information.*

Let us now formalize and generalize the observations that we have made on $L_{\equiv}$.

Theorem 4.3 (DDA $\equiv_{\rightleftarrows}$ RFA, DDA $\equiv_{\circlearrowleft}$ BFA)**.** RFA*s and* BFA*s recognize the same picture languages as* DDA*s run on pictures encoded by the scanning strategy of* RFA *and* BFA*, $\rightleftarrows$ and $\circlearrowleft$, respectively.*

Contrary to the total order on the pictures' pixels (resulting in ordinary string languages), when we use DAG encodings inducing a proper partial order (*proper* meaning no total order), nondeterminism can equip a DAG automaton with more power.



(a) The serialization of a picture as a string DAG for the RFA (green) and BFA (brown) enables deterministic recognition of the language $L_{\equiv}$.

(b) For $\downarrow\rightrightarrows$, with vertical edges only at the left border, deterministically cannot be detected which stripe color occurs first; the first color is hard-coded …

(c) … however this is possible with the right border connected, instead. For $\rightrightarrows\downarrow$, all stripe DAGs are accepted, not only those with the fixed first color.

Figure 2: The DAG encodings $\circlearrowright$, $\rightleftarrows$, $\downarrow\rightrightarrows$ and $\rightrightarrows\downarrow$ recognize horizontal stripes *deterministically*: The above figures illustrate with their red edges those edges where the first row's color is determined. (a) While the total orders on the vertices imposed by the RFA and the BFA scanning strategies with the DAG encodings $\circlearrowright$ and $\rightleftarrows$ can detect the first row's color, (b) the DAG encoding $\downarrow\rightrightarrows$, even though permitting the same number of edges, is blind to the row colors in the input. Therefore, even though a DAG automaton can detect striped pictures with the encoding $\downarrow\rightrightarrows$, it cannot recognize $L_{\equiv}$, but only a subset of it – the set of those pictures whose first line exhibits the hard-coded color. (c) However, sees the first row's color and can alternate the colors subsequently.

Lemma 4.4 ($\mathrm{DDA}_{\rightrightarrows\downarrow}$ simulates $\mathrm{DDA}_{\downarrow\rightrightarrows}$)**.** *A deterministic DAG automaton can simulate using the DAG-encoding $\downarrow\rightrightarrows$ for a picture P by encoding it as $\rightrightarrows\downarrow(\hat{P})$, thus $\mathcal{L}_{\downarrow\rightrightarrows}(\mathrm{DDA}) \subseteq \mathcal{L}_{\rightrightarrows\downarrow}(\mathrm{DDA})$.*

Theorem 4.5 ($\mathcal{L}_{\downarrow\rightrightarrows}(\mathrm{DDA}) \subsetneq \mathcal{L}_{\rightrightarrows\downarrow}(\mathrm{DDA})$)**.** *The class of picture languages recognized by deterministic DAG automata with input pictures P encoded as $\downarrow\rightrightarrows(\hat{P})$ is a proper subclass of those encoded as $\rightrightarrows\downarrow(\hat{P})$.*

Corollary 4.6. DDA *for the DAG-encoding $\downarrow\rightrightarrows$ is not closed under union.*

The scanning strategy seems to determine, which languages an RFA or, equivalently, a BFA can recognize, which would mean that the language is partly encoded within the scanning strategy itself. This leads us to the discussion on the modification of RFAs and BFAs, which allows for greater flexibility and capability in recognizing a wider range of languages.

4.2.2. Adapted Scanning Direction for RFA and BFA

As hinted in the previous section, an RFA can recognize different languages by altering its scanning strategy. More precisely, it need not adapt the strategy; changing direction suffices.

Example 4.7 (DAG language of diagonals)**.** *Let $L_{\setminus}$ be the language where on every diagonal line oriented southeast all pixels have the same color. Fig. 3 illustrates this language with one diagonal only.*

$$L_{\setminus} = \{\, P \in \Sigma^{m,n} \mid P_{i,j} = P_{i+k,j+k} \text{ for } i, i+k \in [m] \text{ and } j, j+k \in [n] \,\}$$

The language $L_{\setminus}$ is not recognizable by an RFA [7, Lemma 47][7].

A DAG automaton can recognize a picture $P_{\setminus} \in L_{\setminus}$ deterministically by encoding $P_{\setminus}$ as $\blacksquare(\hat{P}_{\setminus})$. The rules $\lambda \twoheadrightarrow (\#) \twoheadrightarrow \#$ and $\# \twoheadrightarrow (\#) \twoheadrightarrow \lambda$ detect the border. For all colors $\sigma \in \Sigma$, the rules $\# \twoheadrightarrow (\sigma) \twoheadrightarrow \sigma$, $\sigma \twoheadrightarrow (\sigma) \twoheadrightarrow \sigma$ and $\sigma \twoheadrightarrow (\sigma) \twoheadrightarrow \#$ ensure one fixed color for each diagonal. Like that, a DDA *accepts a DAG $\blacksquare(\hat{P}_{\setminus})$. By Def. 3.2, $\blacksquare(\hat{P}_{\setminus})$ is a disconnected DAG. The DAG encoding $\blacksquare$ uses the edges illustrated in green in Fig. 3a. We could add, as illustrated with black arrows in Fig. 3a, the edge sets $\hat{E}_{\square\downarrow}$ and edges from right to left at the*

[7] The language in the cited proof is restricted to a specific diagonal but that does not affect the validity of the pumping argument for our language.

bottom border to the DAG encoding, in order to obtain a connected *DAG. But we do not need these edges for recognizing the language $L_\backslash$. The automaton would need it however, for recognizing diagonal stripes. Recall that the border ensured the stripes in Example 4.2. Analogously, with these additional edges, a DAG automaton recognizes diagonal stripes deterministically.*

An RFA scans row by row. In order for an RFA to recognize $L_\backslash$, we could either rotate the pictures by the angle of 45° clockwise, or we could adapt the direction of the scanning strategy. If an RFA uses its scanning strategy in the direction southeast instead of east, thus ↘, it will scan diagonals instead of rows. Similarly, a BFA can alternate its scanning direction between northwest and southeast instead of west and east. The *total order* for the diagonal scanning of an RFA of a two-dimensional string S in the direction southeast is given by:

$$S(i_1 j_1) \prec S(i_2 j_2) \text{ if } (i_1 + i_2 < j_1 + j_2) \vee ((i_1 + i_2 = j_1 + j_2) \wedge (j_1 < j_2))$$

and similarly for a BFA.

This shows that an RFA can detect lines in the scanning direction. With a DAG encoding ▥ or a scanning direction downwards, DDA or RFA, respectively, could recognize a language with vertical lines $L_|$. With a DAG encoding ▨ or a scanning direction southwest, DDA or RFA could recognize a language with diagonal lines oriented to the southwest $L_/$. Adapting the scanning strategy to the language is kind of 'cheating' because we should not know which language we will parse. Otherwise, we could always encode a part of the language into the scanning strategy. This would outsource the complexity of the membership problem for a language into its automaton's description via the scanning strategy. So we would hide a part of the language in the scanning strategy. This can be appropriate for certain use cases. However, if we do not know which angle for lines to expect in advance, we can use a grid. A grid has the advantage that it is not tailored to the language. The DAG encoding ▦ serves as one option for implementing a grid. Given a picture DAG ▦($\hat{R}_\backslash$) with one diagonal of the color ● ∈ Σ, the rule for a pixel of the diagonal $tq \twoheadrightarrow$ ◉ $\twoheadrightarrow tp$ and the rule for the pixel on its right-hand side $pt \twoheadrightarrow$ ○ $\twoheadrightarrow qt$ form the rule cycle in Fig. 3b. This rule cycle yields the diagonal in the picture shown in Fig. 3c. It is easy to verify that the DAG encoding ▦ can detect deterministically horizontal, vertical, and diagonal lines.

The DAG encoding ▦ enables the detection of horizontal, vertical, and diagonal patterns within a single language. In contrast, an RFA or BFA must be tailored with a specific scanning direction and, as a result, can recognize only one such pattern per language. Consequently, a DAG automaton operating on pictures encoded via ▦ exhibits strictly greater expressive power than both RFA and BFA. The following section identifies the classical picture automaton model to which the picture-to-DAG encoding ▦ corresponds.

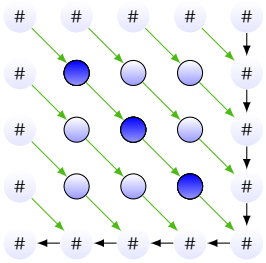
4.2.3. Comparison of online tessellation automata to DAG automata

The DAG encoding ▦ provides the capability to recognize 2D patterns that an RFA cannot. This leads us to the question: What capabilities does a DAG automaton possess when processing pictures encoded via the picture-to-DAG encoding ▦? In order to answer this question, we first show that the boundary of a picture does not increase the expressiveness of a DAG automaton. This observation will ease the proof of expressiveness for the encoding ▦.

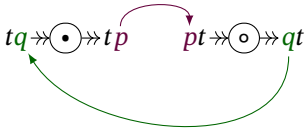
The concept of encoding a picture P as a boundary picture $\hat{P}$ raises the question of whether the presence of this boundary influences expressive power. The following lemma asserts that, for a picture language L, padding pictures with sentinels #, when encoding pictures using the picture-to-DAG encoding ▦, does not affect whether an NDA can accept the picture language L.

Lemma 4.8 (Boundary invariance of NDA). *Let $\hat{A} = (\hat{N}, \Sigma \cup \{\#\}, \hat{R})$ be an* NDA *for recognizing boundary pictures $\hat{P} \in \hat{\mathcal{P}}$ encoded as ▦($\hat{P}$) with $P \in \mathcal{P}$. Then there exists an equivalent* NDA *$A = (N, \Sigma, R)$ recognizing the same set of pictures encoded without the boundary as ▦(P), but restricted to* proper *pictures, meaning pictures with more than one row and column:*

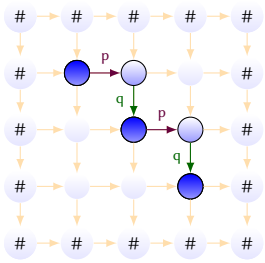
$$\forall m, n > 1 \; \forall P \in \Sigma^{m,n} : ▦(\hat{P}) \in L(\hat{A}) \iff ▦(P) \in L(A).$$



(a) RFA, BFA with adjusted scanning direction and DDA can detect diagonal lines.



(b) This rule cycle detects the diagonal for pictures encoded with ▦.



(c) The rule cycle in b yields this path in the picture, comprising the diagonal.

Figure 3: (a) An RFA or BFA can accept a picture R with diagonal lines if the angle for the scanning strategy is decremented by 45° (the direction of the scanning strategy is shown in the picture, but not the strategy itself), just as a DDA can for the picture being encoded as ▧$(\hat{R})$ (see picture). For the DAG encoding ▦, which has the advantage that this encoding is not tailored to the language, the rule cycled (b) yields the path which includes the diagonal (c).

In fact, the use of the encoding ▦ corresponds directly with an online tessellation automaton:

Theorem 4.9 (2OTA $\equiv_{▦}$ NDA)**.** *Online tessellation automata and DAG automata are equivalent with respect to the picture-to-DAG encoding* ▦*.*

If we use input-driven DAG encoding instead of input-agnostic DAG encoding, a DAG automaton can recognize more picture languages than online tessellation automata.

Theorem 4.10. *There exists a picture language that a* DDA *recognizes with the help of an input-driven DAG encoding, which neither an online tessellation nor a sgraffito*[8]*[11, 12] automaton can recognize.*

The DAG vertex and edge labels do not reflect the topology of the picture. We can retrieve the picture by the unique names of their vertices $v \in \{P(i,j) \text{ for } i \in [m], j \in [n]\}$, but if we prefer a topology preserved also by the wiring of the DAG, we can combine an input-agnostic with an input-driven DAG encoding. Encoding a picture $P \in L_{\text{balance}}$ as ▦$(\hat{P})$ with additional input-driven edges, as specified in the Proof of Theorem 4.10 above, yields proper DAGs for encoding this picture language while preserving the picture not only via the unique vertex naming but also via the topology given by the DAG encoding ▦. Note that L_{balance} includes the picture P', where $\rightleftarrows(\hat{P}') = b^n a^n$. Using the rule set R from the theorem above does not suffice since it would introduce cycles for P'. This can be fixed easily by adding $(\lambda \twoheadrightarrow (\text{b}) \twoheadrightarrow p')$ and $(p' \twoheadrightarrow (\text{a}) \twoheadrightarrow \lambda)$ to the rules. However, these additional rules turn the DDA into an NDA since they introduce nondeterminism. Hence, the topology ▦ within the edge wiring comes at the price of losing determinism.

5. Conclusions

We have proposed a unifying framework for recognizing picture languages via DAG automata by encoding two-dimensional inputs into directed acyclic graphs (DAGs). Using different input-agnostic encodings of pictures into DAGs, we obtained DAG automata equivalent to two-dimensional returning finite automata, boustrophedon automata, and online tessellation automata.

Encoding pictures as DAGs using input-driven edges increases the power of DAG automata for recognizing both string and picture languages. Input-driven edges enable recognition of some context-sensitive string languages. In two dimensions, input-driven DAG automata accept a proper super-class of the class of recognizable languages.

[8]We do not provide the definition of a sgraffito automaton here, since we only refer to the result that L_{balance} cannot be recognized by a sgraffito automaton.

DAG automata have many appealing properties. Blum and Drewes [4] proved that emptiness and finiteness are decidable for DAG automata, and the class of DAGs accepted by DAG automata is closed under union and intersection. These properties carry over to the case of accepting encodings of picture languages.

Deterministic DAG automata can be reduced and their equivalence is decidable in polynomial time [4]. Equivalence for deterministic DAG automata with respect to picture language recognition is still an open problem, as two deterministic DAG automata accepting the same encodings of pictures need not be equivalent, as they can accept different sets of DAGs that do not encode pictures.

Ad Meeres [5] provides a framework to port known efficient finite state automata algorithms and properties from strings to DAG languages. By applying the techniques, we could obtain classes of picture languages that can be efficiently recognized by finite state automata for DAG languages.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] D. Giammarresi, A. Restivo, Two-dimensional languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Volume 3: Beyond Words, Springer, 1997, pp. 215–267. doi:10.1007/978-3-642-59126-6_4.

[2] T. Kamimura, G. Slutzki, Parallel and two-way automata on directed ordered acyclic graphs, Information and Control 49 (1981) 10–51. doi:10.1016/S0019-9958(81)90438-1.

[3] F. Drewes, K. Knight, M. Kuhlmann, Formal Models of Graph Transformation in Natural Language Processing (Dagstuhl Seminar 15122), Dagstuhl Reports 5 (2015) 143–161. doi:10.4230/DagRep.5.3.143.

[4] J. Blum, F. Drewes, Language theoretic properties of regular DAG languages, Inf. Comput. 265 (2019) 57–76. doi:10.1016/j.ic.2017.07.011.

[5] Y. A. Meeres, A new notion of regularity: Finite state automata accepting graphs, in: F. Manea, G. Pighizzini (Eds.), Proceedings 14th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2024), NCMA 2024, Göttingen, Germany, 12–13 August 2024, volume 407 of *EPTCS*, 2024, pp. 5–26. doi:10.4204/EPTCS.407.2.

[6] Y. A. Meeres, F. Mráz, A unifying approach to picture automata, 2025. URL: https://arxiv.org/abs/2509.12077. arXiv:2509.12077.

[7] H. Fernau, M. Paramasivan, M. L. Schmid, D. G. Thomas, Simple picture processing based on finite automata and regular grammars, J. Comput. Syst. Sci. 95 (2018) 232–258. doi:10.1016/J.JCSS.2017.07.011.

[8] D. Průša, F. Mráz, Restarting tiling automata, Int. J. Found. Comput. Sci. 24 (2013) 863–878. doi:10.1142/S0129054113400236.

[9] D. Kuboň, F. Mráz, I. Rychtera, Learning automata using dimensional reduction, in: A. C. B. Garcia, M. Ferro, J. C. R. Ribón (Eds.), Advances in Artificial Intelligence – IBERAMIA 2022 – 17th Ibero-American Conference on AI, Cartagena de Indias, Colombia, November 23–25, 2022, Proceedings, volume 13788 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 41–52. doi:10.1007/978-3-031-22419-5_4.

[10] M. Kutrib, A. Malcher, M. Wendlandt, Input-driven multi-counter automata, Theor. Comput. Sci. 870 (2021) 121–136. doi:10.1016/J.TCS.2021.01.032.

[11] F. Otto, F. Mráz, Cyclically extended variants of sgraffito and restarting automata for picture languages, in: H. Bordihn, R. Freund, B. Nagy, G. Vaszil (Eds.), Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016, Debrecen, Hungary, August 29–30, 2016. Proceedings, volume 321 of *books@ocg.at*, Österreichische Computer Gesellschaft, 2016, pp. 259–273.

[12] D. Průša, F. Mráz, F. Otto, Two-dimensional sgraffito automata, RAIRO Theor. Informatics Appl. 48 (2014) 505–539. doi:10.1051/ITA/2014023.