Finite Automata on Unranked and Unordered DAGs
Extented Version

Lutz Priese

Nr. 22/2007

Arbeitsberichte aus dem
Fachbereich Informatik

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Kontaktdaten der Verfasser

Lutz Priese
Institut für Computervisualistik
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
EMail: priese@uni-koblenz.de

Finite Automata on Unranked and Unordered DAGs Extented Version[*]

Lutz Priese
Fachbereich Informatik
Universität Koblenz-Landau, Germany
priese@uni-koblenz.de

Abstract

We introduce linear expressions for unrestricted dags (directed acyclic graphs) and finite deterministic and nondeterministic automata operating on them. Those dag automata are a conservative extension of the $T_{u,u}$-automata of Courcelle on unranked, unordered trees and forests. Several examples of dag languages acceptable and not acceptable by dag automata and some closure properties are given.

Keywords: finite state automata, directed acyclic graphs, regular dag languages.

1 Introduction

Various equivalent concepts to finite automata operating on finite or infinite words and (ordered and ranked) trees and regularity are known. Those are recognizability with congruences, rationality with magmas, expressibility with regular expressions, definability in certain classes of monadic second order logic, generation by certain right-linear grammars. A good overview on the tree results is given in the TATA book [6]. However, only ranked and ordered trees are considered there, where a graph is called ranked if the degree of any node is determined by its label, leading to ranked alphabets. On the other hand, the degree in an infinite set of unranked graphs may be unbounded. Thus, a finite automaton recognizing languages of unranked trees has to operate on nodes with an unknown number of outgoing arcs (let us call this the "problem of unbounded degree"). An unpublished but well-known report [4] of Brüggemann-Klein, Murata and Wood researches ordered, unranked trees in some detail. Unranked, ordered and unordered trees have been investigated as an algebra by Courcelle in [8]. He presents a very elegant characterization of acceptability by $T_{u,u}$-magmas and frontier-to-root $T_{u,u}$-automata. Unranked and unordered trees with arc labels instead of node labels allow for a simpler algebraic approach and are found in Boneva and Talbot [2]. Brüggemann-Klein, Murata, Wood, and also Boneva, Talbot, solve the problem of unbounded degree for their unranked trees by allowing infinite but regular sets of transitions for their automata, while Courcelle uses an associative and commutative transition function on pairs of state that easily extends to unbounded multisets of states.
Some generalizations of automata to ranked graphs or to their sub-class of ranked directed acyclic graphs are known: Finite graph automata have been introduced by Thomas [15], automata over planar dags by Kamimura and Slutzki [10]. A Kleene theorem for planar dags has been

[*]A shorter version [14] of this paper without proofs and fewer examples was accepted to DLT 2007, Turku, and published in LNCS 4588.

presented by Bossut, Dauchet and Warin [3]. They describe planar dags by linear expression that follow a graphic lay-out and use seriell and parallel composition. Graph expressions have also been introduced by Courcelle [7] for hyper-graphs to define context-free graph grammars. Charatonik [5] has researched automata on t-dags where no isomorphic sub-trees are allowed. Anantharaman, Narendran and Rusinowitch [1] continue this work, where the dag automata are mainly tree automata that run on dags, and present several interesting properties of such recognizable dag languages.
However, there exists no satisfying concept of automata on unrestricted unranked, unordered graphs or dags. Kaminski and Pinter [11] avoid the problem of unbounded degree as their automata define a bound on the degree of acceptable graphs. However, the language of all graphs over a fixed alphabet now is not accepted any more. Fanchon and Morin [9] define regular pomsets languages over unranked alphabets with auto-concurrency via congruences of finite index. Those congruences mirror a serial-parallel composition of pomsets. They receive a concept of regularity that is closed under union but not under intersection or complement.
We will follow Courcelle's approach towards automata - but without using algebras as a semantics. We introduce linear dag expressions as a syntax and give a set-theoretical semantics as graphs. Finite automata operating as well on (congruence classes of) dag expressions as on abstract dags are introduced. In contrast to trees, dags possess incoming and outgoing arcs and all problems of root-to-frontiers and of frontiers-to-root automata must appear in dags. It is known that on trees deterministic root-to-frontier automata are a proper sub-class of nondeterministic ones which are equivalent to deterministic or nondeterministic frontier-to-root automata. As a consequence, the 'root-to-frontier' part of dag automata should be nondeterministic. Our dag automata will therefore contain aspects of deterministic frontier-to-root and nondeterministic root-to-frontier automata.

2 Graphs and DAGs

Set-Theoretic Approach

A set-theoretical approach to graphs is simple:
A *graph* γ over an alphabet Σ is a triple $\gamma = (N, E, \lambda)$ of two finite sets N of *nodes* and $E \subseteq N \times N$ of *edges* and a labelling mapping $\lambda : N \to \Sigma$. Two graphs $\gamma_i = (N_i, E_i.\lambda_i)$ are *isomorphic*, $\gamma_1 \sim_{iso} \gamma_2$, if there exists a bijective function $h : N_1 \to N_2$ with $(v, v') \in E_1 \Leftrightarrow (h(v), h(v')) \in E_2$ and $\lambda_2(h(v)) = \lambda_1(v)$ holds for all v, v' in N_1.
Thus, graphs in this paper are directed, unranked, unordered, finite and node labelled. We use the following rather standard notations.

$^\bullet v := \{v' \in V | (v', v) \in E\}$, $v^\bullet := \{v' \in V | (v, v') \in E\}$. For $V' \subseteq V$: $^\bullet V' := \bigcup_{v \in V'} {}^\bullet v$, $V'^\bullet := \bigcup_{v \in V'} v^\bullet$. Any node in $^\bullet v$ ($v^\bullet$) is a *father (son)* of v. $|^\bullet v|$ ($|v^\bullet|$) is the *in- (out-)degree* of v. A *root (leaf)* of a graph is a node with in-degree (out-degree) 0. A *connection* of length n between two nodes v, v' is a word $w = v_1 ... v_{n+1}$ s.t. $v = v_1, v' = v_{n+1}$ and $(v_i, v_{i+1}) \in E \cup E^{-1}$ holds for $1 \leq i \leq n$. If $(v_i, v_{i+1}) \in E$ holds for all i w is called a *directed path* from v to v'. A *cycle* is a directed path of some length > 0 from one node to itself.
A *dag* (directed acyclic graph) is a graph without cycles. A *forest* is a dag where any node possesses at most one connection to at most one root. A *tree* is a forest with exactly one root. Thus, the empty graph $\varepsilon := (\emptyset, \emptyset, \emptyset)$ is a forest but not a tree. We usually identify isomorphic graphs and thus deal with *abstract* graphs.

By Σ^t, Σ^f, $\Sigma^\dagger$, and Σ^g we denote the sets of all abstract trees, forests, dags, and graphs, respectively, over Σ.
A graph is *ranked* if all nodes with the same label must also possess the same out-degree, and *double ranked* if the label defines both the in- and out-degree. It is *ordered* if a specific order between all sons of any node is given.

Algebraic Specification for Unranked Trees

Courcelle [8] defines a theory $\mathcal{T}_{uu}$ for unranked, unordered trees that consists of a syntax of sorts $S_{uu} = \{l, t, f\}$ (for *letter, tree, forest*), operator symbols $Op_{uu} = \{p_{l\times f\to t}, r_{t\to f}, +_{f\times f\to f}, \theta_{\to f}\}$, and equations E_{uu} :

$$u + v = v + u$$
$$(u + v) + w = u + (v + w)$$
$$u + \theta = u,$$

for variables u, v, w of sort f.
Let Σ denote a set of 0-ary generators of sort l. Any unranked, unordered tree over Σ now simply becomes an element of sort t of $\mathcal{F}(\mathcal{T}_{uu}, \Sigma) = Term(\mathcal{S}_{uu}, \Sigma)/_{\equiv_{uu}}$, where $Term(\mathcal{S}_{uu}, \Sigma)$ are all terms generated by $Op_{uu} \cup \Sigma$ and $\equiv_{uu}$ is the $\mathcal{S}_{uu}$-algebra congruence induced by the equations E_{uu}. To get a theory $\mathcal{T}_{uo}$ for unranked, ordered trees just drop the equation for commutativity. We might try to follow this approach and define a theory $\mathcal{T}_d$ for dags by adding to $\mathcal{T}_{uu}$ a new sort s for *synchronization point* and a new operator symbol $q_{s\times t\to t}$ and study $Term(\mathcal{T}_d, \Sigma \cup \mathbb{N})$, where any integer $i \in \mathbb{N}$ is regarded as a 0-ary symbol of sort s. However, as we will not be able to use $\mathcal{T}_d$-algebras as a semantics for dags such an approach seems to be overloaded. In the following syntax of linear dag expression we mainly abbreviate $p(a, f)$ by af, $p(a, \theta)$ by a, $q(i, t)$ by it, and $r(t)$ by t and give a set-theoretic semantics.

Syntax of Graph Expressions

Let Σ denote a finite alphabet with $\Sigma \cap \mathbb{N} = \emptyset$. We define the sets $E^t_{\Sigma\cup\mathbb{N}}$ and $E^f_{\Sigma\cup\mathbb{N}}$ of *tree* and *forest expressions* over $\Sigma \cup \mathbb{N}$ as the smallest sets fulfilling the following requirements $\forall x \in \Sigma \cup \mathbb{N}$:

$$E^t_{\Sigma\cup\mathbb{N}} \subseteq E^f_{\Sigma\cup\mathbb{N}},\ \theta \in E^f_{\Sigma\cup\mathbb{N}},\ x \in E^t_{\Sigma\cup\mathbb{N}},$$
$$f, g \in E^f_{\Sigma\cup\mathbb{N}} \implies xf \in E^t_{\Sigma\cup\mathbb{N}},\ (f + g) \in E^f_{\Sigma\cup\mathbb{N}}.$$

The binary relation $\equiv$ on forest expressions is defined $\forall f, f', g, g', h, \in E^f_{\Sigma\cup\mathbb{N}}$, $x \in \Sigma \cup \mathbb{N}$ by

1) $f \equiv f$, $f \equiv g \implies g \equiv f$, $(f \equiv g \wedge g \equiv h) \implies f \equiv h$,
2) $f \equiv f' \implies xf \equiv xf'$, $(f \equiv f' \wedge g \equiv g') \implies (f + g) \equiv (f' + g')$,
3) $(f + g) \equiv (g + f)$, $(f + (g + h)) \equiv ((f + g) + h)$, $(f + \theta) \equiv f$.

By 1) $\equiv$ becomes an equivalence relation, by 2) a congruence on our expressions, and fulfills by 3) the equations E_{uu}. The congruence $\equiv_0$ is defined as above but without the requirement $f + \theta \equiv_0 f$ in 3). $(f_1 + ... + f_n)$ abbreviates $(...(f_1 + f_2) + ...) + f_n)$.

Semantics of Graph Expressions

In a first, intermediate step we interpret a graph expression as a forest over $\Sigma \cup \mathbb{N}$.

$\forall x \in \Sigma \cup \mathbb{N}, f, g \in E^g_\Sigma$:
$\Im^o(\theta) := (\emptyset, \emptyset, \emptyset)$, $\Im^o(x) := (\{1\}, \emptyset, \lambda(1) := x)$,
$\Im^o(f) = (V, E, \lambda) \Longrightarrow \Im^o(xf) := (V \cup \{v_{new}\}, E \cup \{(v_{new}, v) | v \in V \wedge^\bullet v = \emptyset\}, \lambda \cup \lambda(v_{new}) := x\}$,
$\Im^o(f+g) := \Im^o(f) + \Im^o(g)$, where $\alpha + \beta$ is the disjoint union of the two graphs α, β.
To get our intended interpretation as abstract graphs over Σ we regard all integers as synchronization points that must be synchronized (i.e., all occurrences of the same integer are identified) and deleted. Therefor we introduce the operation Syd (for "*Sy*nchronize and *d*elete"):

$$Syd_{i_1,\ldots,i_k}(\alpha) := Syd(_{i_1}(\ldots(Syd_{i_k}(\alpha)\ldots), \text{ with}$$

$$Syd_i(\alpha) := (V', E \cap (V' \times V') \cup E', \lambda_{|V'}), \text{ for}$$

$V' = \{v \in V | \lambda(v) \neq i\}$, $E' = \{(v, v') | \exists v_1, v_2 \in V : \lambda(v_1) = \lambda(v_2) = i \wedge (v, v_1), (v_2, v') \in E\}$, setting all nodes v as a father of all nodes v' if v possesses some son with label i and v' possesses some father with label i, deleting such all melted synchronization points.
The interpretation of an expression as an abstract graph is given as

$$\Im(f) := [Syd_{int(f)}(\Im^0(f))]_{\sim iso},$$

where $int(f)$ is the set of all integers appearing in f.

Example 2.1 *$\Im^o(f)$ and $\Im(f)$ for $f = a1b2a + a(1+2+a) + ab$ are shown in Figure 1. The abstract graph α is the interpretation of the expressions $1a2c(1+db2)$, $1c(a1+db1)$, and $1db2c(a2+1)$.*

□

$$\begin{tikzcd}[row sep=small]
& a \arrow[r] & 1 \arrow[r] & b \arrow[r] & 2 \arrow[r] & a \\
& & 1 & & & \\
\Im^o(f): & a \arrow[r] \arrow[ru] \arrow[rd] & 2 & & & \\
& & a & & & \\
& a \arrow[r] & b & & &
\end{tikzcd}$$

$$\begin{tikzcd}[row sep=small]
& a \arrow[r] & b \arrow[r] & a \\
\Im(f): & a \arrow[r] \arrow[ru] \arrow[rru] & a & \\
& a \arrow[r] & b &
\end{tikzcd}$$

$$\begin{tikzcd}[row sep=small]
& & c \arrow[r, shift left] \arrow[rd] & a \arrow[l, shift left] \\
\alpha: & b \arrow[ru] & & d \arrow[ll]
\end{tikzcd}$$

$$\begin{tikzcd}[row sep=small]
& a \arrow[r] \arrow[rd] & b \\
\beta: & c \arrow[r] \arrow[ru] & d \\
& e \arrow[ru] &
\end{tikzcd}$$

Figure 1: Some graphs.

Of course, $\equiv$- or $\equiv_0$-congruent expressions describe the same abstract graph and $[f]_{\equiv_0}$ is always finite.

DAG Expressions

We start with an example.

Example 2.2 *Regard the dag β of Figure 1. There are various different expressions d representing β, i.e. with $\Im(d) = \beta$. We will discuss a few of them.*

- $d_1 := a1 + c1 + 1(2+3) + 2b + 3d + e3$, *where β is cut into "atomar" pieces, its nodes a, b, c, d, e, that are glued together by the synchronization points* 1,2 *and* 3,
- $d_2 := a1b + c12d + e2$, *gluing together the pathes ab, cd and e,*
- $d_3 := a1b + c12 + e2d$, *gluing together the pathes ab, c and ed,*

- $d_4 := a1(b + 2d) + c1 + e2$, *gluing together the tree* $a(b + d)$ *and the pathes* c *and* e, *using a trick - a synchronization point preceding a forest - to give two sons to* c,
- $d_5 := a(1b + 2d) + c(1 + 2) + e2$, *gluing together the trees* $(a(b + d)$, *and the pathes* c, e, *presenting clearly the son structure of* a, c, *and* e,
- $d_6 := c(1d + 2b) + a(1 + 2) + e1$, *an alternative to the idea in* d_5,
- $d_7 := 3c(1d + 2b4 + 5) + a(1 + 1 + 2)4 + 3e1$, *similar to* d_6 *but using superfluous synchronization points* 3,4 *and* 5 *that have no influence on* $\Im$ *but destroy the clear son structure as presented in* d_5 *or* d_6.

$\square$

If we allow all those different graph expression to represent dags we will run into difficulties in defining automata operating on all of them. Only some of those possible expression representing dags are "smooth" and will be allowed as dag expressions. These are d_5 and d_6 in the above example.

We will give a syntactic definition of dag expressions as a sub-class of graph expressions. Therefor we have to identify some interesting parts of the intermediate graph $\Im^o(f)$ in the interpretation for an expression $f \in E^f_{\Sigma\cup\mathbb{N}}$ already in the syntax of f. We operate with multisets. A multiset m over a set M is a mapping $m : M \to \mathbb{N}$ where $m(a)$ denotes the multiplicity of an element a in M. We also write $m \cup m'$ for $m + m'$ and sometimes regard a multiset m as the set $\{a \in M | m(a) > 0\}$. A relation $\rho \subseteq M \times M$ may also be regarded as a multiset where $a\,\rho\,b$ holds for $a, b \in M$ if $\rho(a, b) > 0$. We define

- the multiset $int(f)$ of all integers appearing in f,
- the multiset $first(f)$ of all elements in f that become roots in $\Im^o(f)$,
- the multiset $last(f)$ of all elements in f that become leaves in $\Im^o(f)$, as well as
- $\#_j nol(f)$ the number of occurrences of integer j in f that do not become leaves in $\Im^o(f)$, and
- $\lhd(f) \subseteq int(f) \times int(f)$ a relation that tells the order of integers in $\Im^o(f)$

inductively for $a \in \Sigma$, $i, j \in \mathbb{N}$, $x \in \Sigma \cup \mathbb{N}$, $f, g \in E^f_{\Sigma\cup\mathbb{N}}$ as:

1. $int(\theta) := first(\theta) := last(\theta) := \#_j nol(\theta) := \lhd(\theta) := 0$,
2. $int(a) := 0$, $int(i) := 1{\cdot}i$, $first(x) := last(x) := 1{\cdot}x$, $\#_j nol(x) := \lhd(\theta) := 0$,
3. $int(af) := int(f)$, $int(if) := int(f) + 1{\cdot}i$, $first(xf) := 1{\cdot}x$, $last(xf) := last(f)$, $\#_j nol(af) := \#_j nol(f)$, $\#_j nol(if) := \#_j nol(f) + \delta_{ij}$, for $last(f) \neq 0$ with $\delta_{ij} = 1$ for $i = j$ and 0 elsewhere, $\lhd(af) := \lhd(f)$, $\lhd(if) := \lhd(f) \cup \{(i, j) | j \in int(f)\}$,
4. $X(f + g) := X(f) + X(g)$, for $X \in \{int, first, last, \#_j nol, \lhd\}$.

Obviously, the interpretation $\Im(f)$ of an expression f is an abstract dag if the transitive closure $\lhd(f)^+$ of $\lhd(f)$ is an irreflexive partial order. We now present a formal definition for graph expression admissible as dag expressions:

An expression $f \in E^f_{\Sigma\cup\mathbb{N}}$ is called *admissible* if $f = \theta$ or

1. f contains no sub-expressions $\theta,\ ij,\ ig$ with $i,j \in \mathbb{N}, g \in E^f_{\Sigma\cup\mathbb{N}} - E^t_{\Sigma\cup\mathbb{N}}$,
2. no f' with $f \equiv_0 f$ contains a sub-expression $(t_1 + t_2)$ with two tree expressions t_1, t_2 with $first(t_1) = first(t_2) \in \mathbb{N}$,
3. $first(f) \cap \mathbb{N} = \emptyset$, and
4. $j \in int(f) \implies \#_j nol(f) = 1$ and $int(f)(j) \neq 1$.

In an admissible expression we drop the redundant symbol θ (with the exception of $f = \theta$ to express the empty forest). Further, no two synchronization points must follow each other (1) (as this must lead to a complicated synchronization of synchronization points with different names). Each occurring synchronization point j must occur exactly once not as a last element (4). In this case j must precede some tree expression (1). It makes no sense to synchronize the roots of two brother trees (2). If they are both sons of a common father node this father does the synchronization itself. If not, we will not synchronize a root of f with another node (3) to avoid situations as indicated in d_1. It makes no sense to synchronize only leaves. However, life will become more easy if we synchronize only leaves with exactly one inner node that is no leaf. We also claim (4) that no synchronization points appears exactly once in f as in this case i has no synchronization partner and is without meaning. It should be noted (4) that at most one occurrence of an integer j can be followed by a tree expression. Thus, for admissible expressions 2. is equivalent to

2'. no f' with $f \equiv_0 f$ contains a sub-expression $(jt + j)$ for any $t \in E^t_{\Sigma\cup\mathbb{N}}$, $j \in \mathbb{N}$,

and 4. is equivalent to

4'. $j \in int(f) \implies \#_j nol(f) = 1$ and $last(f)(j) > 0$.

Tree and forest expressions over Σ are simply expressions without integers, a *graph expression* over Σ is a forest expression over $\Sigma \cup \mathbb{N}$, and a *dag expression* over Σ is an admissible graph expression where $\lhd^+$ is an irreflexive partial order:

$$E^t_\Sigma := \{t \in E^t_{\Sigma\cup\mathbb{N}} \mid int(t) = \emptyset\},\ E^f_\Sigma := \{f \in E^f_{\Sigma\cup\mathbb{N}} \mid int(f) = \emptyset\},\ E^g_\Sigma := E^f_{\Sigma\cup\mathbb{N}},$$
$$E^\dagger_\Sigma := \{f \in E^g_\Sigma \mid f \text{ is admissible and } \nexists i,j \in int(f) : (i \lhd(f)^+ j \wedge j \lhd(f)^+ i)\}.$$

All abstract trees, forest, dags, and graphs can be expressed:

Lemma 2.1 $\Im(E^t_\Sigma) = \Sigma^t$, $\Im(E^f_\Sigma) = \Sigma^f$, $\Im(E^\dagger_\Sigma) = \Sigma^\dagger$, $\Im(E^g_\Sigma) = \Sigma^g$.

3 DAG-Automata

Let M be a set. A function $f : M \times M \to M$ is *associative* and *commutative* if $f(x,y) = f(y,x)$ and $f(x, f(y,z)) = f(f(x,y),z)$ holds for all $x,y,z \in M$.
Such an associative and commutative function f is easily extended to

$$f^* : (\mathbb{N}^M - \{0\}) \to M$$

operating on nonempty multisets over M: $\forall a \in M : \forall m \in \mathbb{N}^M - \{0\}$:

$$f^*(1 \cdot a) := a, f^*(1 \cdot a + m) := f(a, f^*(m)).$$

A (root-to-frontier) *dag automaton*

$$A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$$

consists of
- a finite set Q of states with $Q \cap \mathbb{N} = \emptyset$,
- a finite alphabet Σ with $Q \cap \Sigma = \emptyset = \Sigma \cap \mathbb{N}$,
- a function $\delta : Q \times \Sigma \to 2^Q$,
- four associative and commutative functions

$$\delta_i, \delta_o, \delta_I, \delta_F : Q \times Q \to Q,$$

- a set $I \subseteq Q$ of initial states, and
- a set $F \subseteq Q$ of final states.

We now introduce the concept of configurations and computations of dag automata. A *configuration* C for A is a graph expression over $\Sigma \cup Q$ where exactly the first elements of C are states. A configuration C_d for a dag expression d results by putting one state before any first element of d. C_d is start configuration for d if for the multiset $first(C_d)$ of state $\delta_I^*(first(C_d)) \in I$ holds. $\mathcal{C}_d$ denotes the set of all start configurations for d. A configuration that consists solely of states - i.e., a multiset of states - is called a *final configuration.* As a configuration is a dag expression itself the congruences $\equiv$ and $\equiv_0$ may also be applied to configurations.

A configuration C' is a *direct successor* of a configuration C, $C \vdash_A C'$, if there exists a configuration $\hat{C}$ with $C \equiv_0 \hat{C}$ and C' is the result of replacing in $\hat{C}$ a sub-expression

1. sa by s', with $s' \in \delta(s, a)$, or
2. $s(f_1 + f_2)$ by $(s_1 f_1 + s_2 f_2)$, with $\delta_o(s_1, s_2) = s$, or
3. $(s_1 it + s_2 i)$ by $s' it$, with $\delta_i(s_1, s_2) = s'$, or
4. si by s, if i occurs exactly once in $\hat{C}$,

for $a \in \Sigma, i \in \mathbb{N}, t \in \Sigma^t_{\Sigma \cup \mathbb{N}}, f_1, f_2 \in \Sigma^g_\Sigma, s, s_1, s_2 \in Q$.

In addition, we write $d \vdash_A C$ for a dag expression d and a start configuration $C \in \mathcal{C}_d$ and $C_f \vdash_A s$ for a final configuration C_f with $\delta_F^*(C_f) = s$. Only here δ_I and δ_F play a rôle.
δ_o handles the transport of a state to outgoing arcs and δ_i of states from incoming arcs.
$eval^A(d) := \{s \in Q | d \vdash_A^* s\}$ is the set of all states into which a dag expression d may evaluate under A.
$D(A) := \{d \in E^\dagger_\Sigma | eval^A(d) \cap F \neq \emptyset\}$ is the dag language *accepted* by A. A dag language is *regular* if it is accepted by some dag automaton.

Example 3.1 *Let $\Sigma^\dagger{}_{even}$ denote the language of all dags over Σ with an even number of nodes. Σ_{even} is accepted by A_{even} with $Q := \{\mathbf{0}, \mathbf{1}\}$, where we use boldface integers as states, $I := F := \{\mathbf{0}\}$, $\delta(s, a) := s + \mathbf{1} \bmod 2$ for all $a \in \Sigma$, and $\delta_F := \delta_I := \delta_i := \delta_o := + \bmod 2$.*
A nondeterministic computation with the dag expression f of example 2.1 for the dag $\Im(f)$ of Figure 1 is, e.g.

$$f = a1b2a + \Big(a\big((1+2)+a\big)+ab\Big) \vdash \mathbf{1}a1b2a + \Big(\mathbf{0}a\big((1+2)+a\big)+\mathbf{1}ab\Big)\ (\in \mathcal{C}_f)$$
$$\vdash^*_{(\delta)} \mathbf{0}1b2a + \mathbf{1}\big((1+2)+a\big)+\mathbf{0}b \vdash^*_{(\delta_o,\delta)} \mathbf{0}1b2a + \big(\mathbf{1}(1+2)+\mathbf{0}a\big)+\mathbf{1} \vdash_{(\delta_o)}$$
$$\mathbf{0}1b2a + \big((\mathbf{1}1+\mathbf{0}2)+\mathbf{0}a\big)+\mathbf{1} \equiv_0 \Big(\big((\mathbf{0}1b2a+\mathbf{1}1)+\mathbf{0}2\big)+\mathbf{0}a\Big)+\mathbf{1} \vdash_{(\delta_i)}$$
$$\big((\mathbf{1}1b2a+\mathbf{0}2)+\mathbf{0}a\big)+\mathbf{1} \vdash^* (\mathbf{1}b2a+\mathbf{0}2)+(\mathbf{1}+\mathbf{1}) \vdash^* (\mathbf{0}2a+\mathbf{0}2)+\mathbf{0} \vdash \mathbf{0}2a+\mathbf{0} \vdash^* \mathbf{1}.$$

Any computation for f leads to $\mathbf{1}$*, thus* $f \notin D(A_{even})$.

□

Any start configuration is a dag expression, but a computation may lead to configurations that are graph expression but no dag expressions, i.e., if exactly one synchronization point is left that has to be eliminated by rule 4.

Example 3.2 *Let* ${\Sigma^\dagger}_{dis}$ *denote the language of all disconnected dags over* Σ *and* ${\Sigma^\dagger}_{con}$ *those of all connected dags. We present an automaton* A_d *accepting* ${\Sigma^\dagger}_{dis}$*. When an abstract dag* α *consists of two disjunct dags* α_1, α_2 *an accepting computation of* A_d *guesses a state* s_1 *to be attached to all roots of* α_1 *and a different state* s_2 *to all roots of* α_2*.* A_d *passes* s_i *through* α_i*. If* α_1 *and* α_2 *should have a common node the states* s_1 *and* s_2 *will meet and pass an error message to some leaf thats forbids acceptance. Thus simply choose*
$Q := \{s_0, s_1, s_2, \checkmark, \bot\}$ *with a sink state* $\bot$ *(s.t.* $\delta_\bullet(x,y) = \bot$ *if* $x = \bot$ *or* $y = \bot$ *for all transition functions),* $I := \{s_0\}, F := \{\checkmark\}$,
$\delta_I(s_1,s_2) := s_0, \delta_I(s_1,s_1) := s_1, \delta_I(s_2,s_2) := s_2$,
$\delta(s_1,x) := s_1, \delta(s_2,x) := s_2$, *for* $x \in \Sigma$,
$\delta_o(s_1,s_1) := s_1, \delta_o(s_2,s_2) := s_2$,
$\delta_i(s_1,s_1) := s_1, \delta_i(s_2,s_2) := s_2, \delta_i(s_1,s_2) := \bot$,
$\delta_F(s_1,s_1) := s_1, \delta_F(s_2,s_2) := s_2, \delta_F(s_1,s_2) := \checkmark, \delta_F(\checkmark,s_i) := \checkmark$ *for* $i = 1,2$*, plus all required transitions to get commutative and associative mappings and make* $\bot$ *a sink state. A nonaccepting and an accepting computation for the dag expression* f *in example 3.1 are shown in Figure 2. Thus,* $f \in {\Sigma^\dagger}_{dis}$.

□

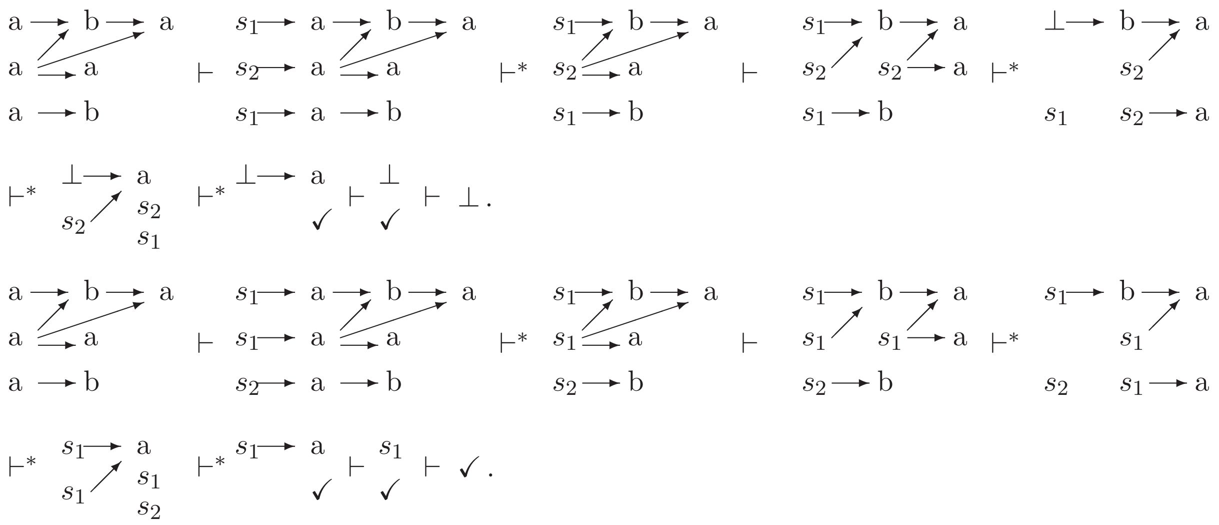

Figure 2: A nonaccepting and an accepting computation of A_{dis}

We may regard a dag automaton as operating on dag expressions, $\equiv_0$-congruence classes of them, or on abstract dags, as any dag automaton operates identically on different dag expressions of the same abstract dag:

Theorem 1 *For all dag expressions d_1, d_2 and dag automata A over the same alphabet:*

$$\Im(d_1) = \Im(d_2) \implies eval^A(d_1) = eval^A(d_2).$$

Proof. Let $\alpha = [G]_{\sim_{iso}}$ be a graph represented by a dag expressions d, i.e. $\Im(d) = \alpha$. We firstly note that the number of roots (leaves) of α coincides with the size of $first(d)$ ($last(d)$): By 1. (no θ allowed) and 3. of admissibility roots cannot result from synchronization (thus, $|first(d)| \leq |root(\alpha)|$) or as last elements of some tree sub-expression (thus, $|first(d)| \geq |root(\alpha)|$). Any leaf v of G with $\lambda(v) = a \in \Sigma$ must be expressed by at least one occurrence $\mathbf{a}$ of a in $last(d)$. $a\,i$ is not allowed in d_1 as in this case i must find a proper synchronization partner (4.) s.t. $\mathbf{a}$ would not present a leaf in G. Thus, $|leaves(\alpha)| \leq |last(d) \cap \Sigma|$. On the other hand, by the definition of $\Im^o$ one occurrence $\mathbf{a}$ of a in $last(d)$ cannot represent two different leaves in G, thus $|last(d) \cap \Sigma| \geq |leaves(G) \cap \Sigma| = |leaves(\alpha)|$.
Let v be a node in G with $\lambda(v) = a \in \Sigma$. There must be at least one occurrence $\mathbf{a}$ of a in d representing v. However, as no synchronization can melt two occurrences $\mathbf{a}_1, \mathbf{a}_2$ into the single node v of G there is exactly one occurrence $\mathbf{a}$ in d that is mapped into v via $\Im^o$.
Let $indegree(v) = n > 1$. Thus, we need one synchronization point, i, to express the incoming arcs into v. Two different synchronization points i and j cannot do the job as $i\,j$ (1.) cannot be a sub-expression of d. Thus $\geq n$ occurrences of i are required. However, as no occurrence of i is a first element of d or follows a θ, any such occurrence give reason to one incoming arc. Thus there are exactly n occurrences of i in d. By 4. $n-1$ are last elements and one occurrence is with in the sub-expression $i\,\mathbf{a}\,f$ for some forest sub-expression f or empty sub-expression f.
For $indegree(v) = 1$ the same arguments show that no integer may precede $\mathbf{a}$ as single occurrences of integers are forbidden (4.).
Let $outdegree(v) = o > 0$. In this case the sub-expression $i\,\mathbf{a}\,f$ possesses a forest sub-expression f with at least o elements in $first(f)$. However, no first elements of f can be synchronized (by 2.) or ignored (1., θ as well as $i\,j$ is forbidden), yielding $o = |first(f)|$.

Now, let d_1, d_2 be two different dag expressions representing α. Let C_{d_1} be a start configuration for a computation from d_1. This implies that $m := first(C_{d_1})$ is a multiset of states with $\delta_I^*(m) \in I$. However, as $|first(d_1)| = |first(d_2)| = |roots(\alpha)|$ there exists also an admissible start configuration C_{d_2} with also $m = first(D_{d_2})$.

Let $\mathbf{a}_j$ be the occurrence of a in d_j, $j = 1, 2$, representing the same occurrence $\mathbf{a}$ of a node labelled with a in α. For $indegree(\mathbf{a}) = n > 1$ and $outdegree(\mathbf{a}) = o \geq 1$ we have sub-expressions $i_j\,\mathbf{a}_j\,f_j$ in d_j with exactly n occurrences of i_j in d_j and with $|first(d_j)| = o$. Suppose we have a computation in d_1 from C_{d_1} to a configuration C_1' where all occurrences of i_1 are preceded by a multiset m' of states. By induction hypothesis we can find a computation from C_{d_2} to some configuration C_2' where also all occurrences of i_2 are preceded by the same multiset m'. If one can compute from C_1' by applying δ_i^* to i_1 (i.e., applying rules 3. and 4. of a direct successor in any order) resulting in C_1'' with a sub-expression $s\,\mathbf{a}_1$ with $s = \delta_i^*(m')$ we can do the same rule application to C_2' resulting in C_2'' with the sub-expression $s\,\mathbf{a}_2$ for the same s.
Suppose we have a computation in d_1 from C_{d_1} to a configuration C_1' with a sub-expression $s\,f_1$. By induction hypothesis we can find a computation from C_{d_2} to some configuration C_2' with the subexpression $s\,f_2$ with the same state s. Thus, any series of application of rule 2. to C_1' - to

simulate δ_o^* - can also be applied to C_2' as $|first(f_1)| = |first(f_2)| = o$.
Thus, as all $\delta_\bullet$ functions are commutative and associative, any computation from d_1 can be simulated in a straight forward way started from d_2.

■

Example 3.3 *The dag expressions*

$$d_1 = a(b(e+3f)+1c3+2d3)+b(1+2), \;\; d_2 = b(2d3f+1c3)+a(1+2+b(3+e))$$

describe the same dag. We present a computation of some dag automaton A on both. For simplicity we assume a deterministic automaton where $I := \{s_0\}$, $\delta_o(s,s) = \delta_I(s,s) = s$, $\delta(s,a) := s_a$ holds and abbreviate $s_{x\|y} := \delta_i(s_x, s_y)$, $s_{xa} := \delta(s_x, a)$ and $s_{x+y} := \delta_F(s_x, s_y)$.

$s_0 a(b(e+3f)+1c3+2d3)+s_0 b(1+2)$
$\vdash^* s_a(b(e+3f)+1c3+2d3)+s_0 b(1+2)$
$\vdash^* s_a b(e+3f)+s_a 1c3+s_a 2d3+s_b 1+s_b 2$
$\vdash^* s_{ab}(e+3f)+s_{a\|b}1c3+s_{a\|b}2d3$
$\vdash^* s_{ab}(e+3f)+s_{a\|b}c3+s_{a\|b}d3$
$\vdash^* s_{abe}+s_{ab}3f+s_{(a\|b)c}3+s_{(a\|b)d}3$
$\vdash^* s_{abe}+s_{(ab\|(a\|b)c\|(a\|b)d)}3f \vdash^* s_{abe+(ab\|(a\|b)c\|(a\|b)d)f}$

$s_0 b(2d3f+1c3)+s_0 a(1+2+b(3+e))$
$\vdash^* s_b 2d3f+s_b 1c3+s_a 1+s_a 2+s_{ab}3+s_{ab}e$
$\vdash^* s_{b\|a}d3f+s_{b\|a}c3+s_{ab}3+s_{abe}$
$\vdash^* s_{((b\|a)d\|(b\|a)c\|ab)}3f+s_{abe} \vdash^* s_{((b\|a)d\|(b\|a)c\|ab)f+abe}$

Obviously, $s_{abe+(ab\|(a\|b)c\|(a\|b)d)f}$ and $s_{((b\|a)d\|(b\|a)c\|ab)f+abe}$ coincide by commutativity and associativity of the functions.

□

Although the treatment of incoming and outgoing arcs in dag expressions is completely different (using a simple + for outgoing arcs but an alphabet of infinitely many synchronization points for incoming arcs) they are treated symmetrically in dag automata. This is easily seen with reverse dags.
The *reverse* A^{rev} of an automaton $A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$ is

$$A^{rev} = (Q, \Sigma, \delta^{rev}, \delta_i^{rev}, \delta_o^{rev}, \delta_I^{rev}, \delta_F^{rev}, I^{rev}, F^{rev}),$$

with

$$I^{rev} := F, F^{rev} := I, \delta^{rev}(s,a) := \{s' | s \in \delta(s',a)\},$$
$$\delta_i^{rev} := \delta_o, \delta_o^{rev} := \delta_i, \delta_I^{rev} := \delta_F, \delta_F^{rev} := \delta_I.$$

Any successful computation in A for some α defines in reverse order immediately a successful computation for α^{rev} in A^{rev}. This implies $D(A)^{rev} \subseteq D(A^{rev})$. Further

$$D(A) = (D(A)^{rev})^{rev} \subseteq D(A^{rev})^{rev} \subseteq D((A^{rev})^{rev}) = D(A), \text{ thus}$$

Lemma 3.1 $D(A^{rev}) = D(A)^{rev}$

4 Regular and Nonregular DAG Languages

Simple Regular DAG Languages and Closure Properties

Some typical examples of regular dag languages are - as expected - the language of all dags
- where no node possesses several sons with the same labels,
- where no node possesses several fathers with the same labels,
- where no node possesses several sons (fathers, respectively) with different labels,
- with exactly j roots (j leaves or j nodes, respectively),
- with 0 roots (0 leaves or 0 nodes, respectively) modulo some constant,
- where all nodes have the in-degree i or 0 (out-degree i or 0, in-degree mod i is 0, out-degree mod i is 0, respectively).

Constructing dag automata that accept those languages is just a simple exercise.

A *maximal path* in a dag is a directed path from some root to some leaf. A path is identified with the word of labels of its nodes. $path(\alpha)$ is the set of all maximal paths in a dag α and $path(D) = \bigcup_{\alpha \in D} path(\alpha)$ for $D \subseteq \Sigma^\dagger$ defines a projection

$$path : 2^{\Sigma^\dagger} \to 2^{\Sigma^*}$$

from languages over dags into languages over words. In the opposite direction there are two canonical ways to embed languages over words into languages over dags:

- the *skinny* embedding of $L \subseteq \Sigma^*$ regards any word $w \in \Sigma^*$ as a path $w \in \Sigma^\dagger$ and is also denoted as $L\,(\subseteq \Sigma^\dagger)$,
- the *fat* embedding D_L of $L \subseteq \Sigma^*$ is the dag language

$$D_L := \{\alpha \in \Sigma^\dagger | path(\alpha) \subseteq L\}.$$

Lemma 4.1 *The skinny embedding L and the fat embedding D_L of a regular word language L are regular dag languages.*

Proof. This is obvious for the skinny embedding. For the fat embedding let the deterministic automaton $A_L = (Q_L, \Sigma, \delta_L, s_L, F_L)$ accept $L \subseteq \Sigma^*$. To accept D_L by $A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$ we set for $M, N \subseteq Q_L, x \in \Sigma$:
$Q := 2^{Q_L} \cup \{\bot\}$ with a sink state $\bot$, $I := \{s_L\}, F := \{M | M \subseteq F_L\}$,
$\delta(M, x) := \{\delta_L(s, x) | s \in M\}$, $\delta_i(M, N) := M \cup N, \delta_o(M, M) := M$,
$$\delta_F(M, N) := \begin{cases} F_L & ; \; if \;\; M, N \subseteq F_L, \\ \bot & ; \; elsewhere \end{cases}$$
■

The opposite statement holds for projections of a fat embedding but not for projections of general dag languages.

Lemma 4.2 *If D_L is a regular dag language then L is a regular word language. But there exist regular dag languages whose path projection is not even a context-free word language.*

Proof. Let $A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$ accept the fat embedding D_L of some word language L. For $w \in L$ we regard w as a path with $path(w) = w$, thus w must be also in D_L and A accepts w without using $\delta_i, \delta_o, \delta_I$, or δ_F. Thus, the non-deterministic automaton $A_L := (Q, \Sigma, \delta, I, F)$ accepts L.

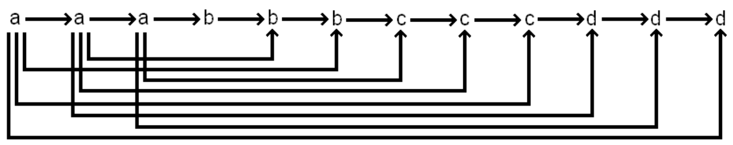

Figure 3: A dag accepted by A_D.

For a counterexample of a regular dag language with a noncontext-free path projection we construct a dag automaton A_D that accepts only dags of the form as shown in Figure 3 with an equal number of labels a, b, c and d where the order a before b before c before d must be respected. This can be achieved by forcing all nodes with a label a to have four sons, labelled with a, b, c, d, but one who has three sons labelled with b, c and d. All nodes labelled with $x \in \{b, c, d\}$ are forced to have two fathers, one labelled with a and a second labelled with x with an exception for the first label x as shown. Formally, define A_D with
$Q := (\{s_a, s_a^o, s_b, s_c, s_d, s_b^i, s_c^i, s_d^i, s_{a,b}^i, s_{b,c}^i, s_{c,d}^i, s_1, s_1', s_2, \perp\}$ with a sink state $\perp$,
$\Sigma := \{a, b, c, d\}$, $I := \{s_a\}, F := \{s_d^i\}$,
$\delta(s_a, a) = s_a^o, \delta(s_x, x) := s_x^i$ for $x \in \{b, c, d\}$, $\delta(.,.) := \perp$ elsewhere,
$\delta_i(s_b^i, s_{a,b}^i) := s_b, \delta_i(s_c^i, s_{a,c}^i) := \delta(s_b^i, s_{a,c}^i) := s_c, \delta(s_c^i, s_{a,d}^i) := \delta(s_d^i, s_{a,d}^i) := s_d$,
$\delta_o(s_a, s_b^i) := s_1, \delta_o(s_1, s_c^i) := s_2, \delta(s_b, s_c^i) := s_1', \delta_o(s_2, s_d^i) := \delta(s_1', s_d^i) := s_a^o$,
$\delta_I(q, q') := \delta_F(q, q') := \perp$ for $q, q' \in Q$.

There holds $\delta_I^*(m) = s \in Q - \{\perp\} \Leftrightarrow m = 1 \cdot s$ which forces one root that achieves the state s_a in any start configuration. Analogously, δ_F forces exactly one leaf in state s_d^i in the final configuration of an accepting computation. Note, $\delta_o^*(m) = s_a \Leftrightarrow (m = 1 \cdot s_a + 1 \cdot s_b^i + 1 \cdot s_c^i + 1 \cdot s_d^i$ or $m = 1 \cdot s_b^i + 1 \cdot s_c^i + 1 \cdot s_d^i)$. Thus, δ_o forces any a-node to possess exactly four sons, labeled with a, b, c, d, or three sons labelled with b, c, d. δ_i forces each node labelled with b, c or d to possess exactly two fathers, one of them an a-node. Thus, A_D accepts only dags with an equal number of labels a, b, c and d where the order a before b before c before d must be respected. I.e.,

$$path(D(A_D)) = \{a^i d^j | 1 \leq i, j\} \cup \{a^i c^j d^n | 1 \leq i, j \leq n\} \cup \{a^i b^j c^n d^n | 1 \leq i, j \leq n\},$$

a noncontext-free language.

■

Lemma 4.3 *The class of regular dag languages is closed under union and intersection.*

Proof. The standard product of finite automata is simply generalized to dag automata:
For $A_j = (Q_j, \Sigma, \delta_j, \delta_{i,j}, \delta_{o,j}, \delta_{t,j}, s_j, F_j), j = 1, 2$, set

$$A_1 \times A_2 := (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, \delta_{i,1} \times \delta_{i,2}, \delta_{o,1} \times \delta_{o,2}, \delta_{t,1} \times \delta_{t,2}, (s_1, s_2), F'),$$

with

$$\delta_{F,1} \times \delta_{F,2}((x_1, x_2), (y_1, y_2)) := (\delta_{F,1}(x_1, y_1), \delta_{F,2}(x_2, y_2)), \text{ etc.}$$

To get the union $D(A_1) \cup D(A_2)$ choose $F' := (F_1 \times Q_2) \cup (Q_1 \times F_2)$. For the intersection set $F' := F_1 \times F_2$.

■

Some Gaps between Regular and Nonregular DAG Languages

There are simple example of nonregular dag languages as finite dag automata cannot count above some boundary: the language $D_{r=l}$ of all dags with the same number of roots and leaves, or $D_{n=}$ ($D_{r=}$, $D_{l=}$) over $\{a, b\}$ where equally many nodes (roots, leaves, respectively) are labelled with a and b. However, if one would change the concept of nondeterministic dag automata in such a way that also partial, not associative functions δ_I and δ_F are allowed then $D_{r=}$ and $D_{l=}$ (in contrast to $D_{n=}$) become regular. To accept $D_{l=}$ choose $Q = \{s_0, s_a, s_b, \checkmark\}$, $I = \{s_0\}$, $F = \{\checkmark\}$, $\delta_I(s_0, s_0) = s_0$, $\delta(s_0, x) = s_x$ and $\delta_i(s_x, s_y) = s_0$ and $\delta_o(s_x, s_x) = s_x$ for $x, y \in \{a, b\}$ such that s_x tells that the last node visited has been labelled with x. Now, simply set $\delta_F(s_a, s_b) = \checkmark$, $\delta_F(\checkmark, \checkmark) = \checkmark$ and $\delta_F(.,.)$ undefined elsewhere to accept $D_{l=}$. For $D_{r=}$ use the reverse automaton. However, such a trick is impossible with total associative and commutative functions δ_I, δ_F.

Theorem 2 *$\Sigma^\dagger_{dis}$ is regular but $\Sigma^\dagger_{con}$ is not. Thus, regular dag languages are not closed under complement.*

Proof. Regularity of ${\Sigma^\dagger}_{dis}$ was shown in example 3.2. Non-regularity of ${\Sigma^\dagger}_{con}$ is seen as follows: Suppose there exists an automaton A that accepts $\Sigma^\dagger_{con}$. Set

$$d := \sum_{1 \le i \le n} a(ia + i')$$

with $i' := i + 1$ for $1 \le i < n$ and $n' := 1$, compare Figure 4 with $\alpha := \Im(d)$. α is connected.

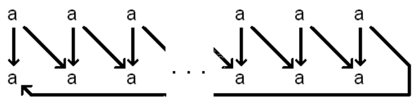

Figure 4: $\Im(d)$.

Thus, A accepts α. For n large enough (i.e., longer than $(|Q|+1)^2$) there must exist two different occurrences o'_1, o'_2 of labels a on the lower row and o_1, o_2 of their right fathers in the upper row where an accepting computation $\mathcal{C}$ of A reaches in o_1 and o_2 the same state, say s, and in o'_1 and o'_2 a same state, say s'. Now, let β result from re-pointing the arc originally from o_1 to o'_1 now to o'_2 and the arc pointing originally from o_2 to o'_2 now to o'_1. This doesn't introduce cycles and the same computation $\mathcal{C}$ will still accept β - but β is disconnected (and still planar).

■

However, "bounded" connectivity becomes regular:

Lemma 4.4 *The languages of all connected dags with a fixed or bounded number of roots or leaves, respectively, are regular.*

Proof. If the number of roots is bounded by l we can construct a finite automaton that guesses the number $k \le l$ of roots and attaches to any root a different state $s_1, ..., s_k$ in an initial configuration. For $M, N \subseteq \{1, ..., k\}$ a state s_M is passed along until it meets another state s_N , switching into $s_{M \cup N}$. s_M tells that up to now all roots in M can be connected by an undirected path. δ_F has to check that all roots are connected. Therefore we need further states with subscripts $\mathcal{M}, \mathcal{N}$ that are sets of subsets of $\{1, ..., k\}$ and set $\delta_F(s_M, s_M) := s_{\{M\}*\{N\}}$,

$\delta_F(s_{\mathcal{M}}, s_{\mathcal{N}}) := s_{\mathcal{M}*\mathcal{N}}$ with $A \in \mathcal{M} * \mathcal{N} \;:\Longleftrightarrow (\exists B \in \mathcal{M} : \exists C \in \mathcal{N} : (B \cap C \neq \emptyset \wedge A = B \cup C)$ or $(A \in \mathcal{M} \cup \mathcal{N} \wedge \nexists B \in \mathcal{M} \cup \mathcal{N} : (A \neq B \wedge A \cap B \neq \emptyset))$. Now, with $F := \{s_{\{\{1,\ldots,k\}\}}\}$ connectivity is detected. By symmetry the same holds for leaves.

■

Ladders of type 1 or 2 and *beams* are dags as presented in Figure 5. $D_{1-ladder}$, $D_{2-ladder}$, D_{beam}

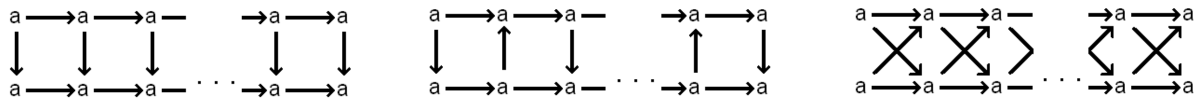

Figure 5: A type 1 ladder (left), type 2 ladder (middle) and beam (right)

denote the languages of all type 1 ladders, all type 2 ladders, and all beams, respectively, over a.

Theorem 3 *D_{beam} and $D_{2-ladder}$ are regular, $D_{1-ladder}$ is not.*

Proof. Assume there exists an automaton A_l accepting exactly the type 1 ladders. For a type 1 ladder α long enough (i.e., longer than $(|Q|+1)^2$) there must exist two occurrences o_1, o_2 of a in the upper line such that some accepting computation attaches to o_1 and to o_2 a same state, say s, and to their two sons in the lower row also a same state, say s'. Let β result from α by interchanging the lower sons of o_1, o_2. β stays acyclic and the same computation accepts also β, but β is no type 1 ladder any more. A contradiction. It should be noted for the following corollary that β is not planar any more.

Such an argument fails for beams or type 2 ladders as interchanging those two sons in a beam or type 2 ladder will introduce cycles. However, the behavior of a dag automaton on a graph that is no dag is irrelevant. To prove that D_{beam} is regular we need the following topological property of beams: Beams are exactly those dags that
- possess exactly two roots and two leaves,
- all non-roots possess exactly two fathers,
- all non-leaves possess exactly two sons,
- there exist no three nodes v, v_1, v_2 such that v is the father of both v_1 and v_2 where v_1 is the second father of v_2, i.e. no node is son and grandson of the same node.

All those properties are acceptable by a dag automaton A_b with
$Q := \{s_0, s_1, s_1^i, s_1^o, s_2, s_2^i, s_2^o, \checkmark, \perp\}$ with a sink state $\perp$, $I := \{s_0\}, F := \{\checkmark\}$,
$\delta_I(s_1, s_1) := s_0, \delta(.,.) := \perp$, elsewhere,
$\delta(s_j, a) := s_j^o, \delta(s_j^i, a) := \delta(s_j^o, a) := \perp$ for $j = 1, 2$,
$\delta_o(s_j^i, s_j^i) := s_j^o$ for $j = 1, 2$ and $\delta_o(.,.) := \perp$ elsewhere,
$\delta_i(s_1^i, s_1^i) := s_2, \delta_i(s_2^i, s_2^i) := s_1$, $\delta_i(.,.) := \perp$ elsewhere,
$\delta_F(s_1^o, s_1^o) := \delta_F(s_2^o, s_2^o) := \checkmark$ and $\delta_F(.,.) := \perp$ elsewhere.
δ_I forces two roots with an attached state s_1 in a start configuration. The change from s_j to s_j^o to s_j^i forces each node to possess exactly two sons and two fathers (or none). The change from sub-script 1 to 2 to 1 forces that no node is a son and grandson of one node.

A proof for the regularity of type 2 ladders is similar. Let an *i-j-node* be a node with i in-coming and j out-going arcs. Then type-2-ladders are uniquely described as those dags with

- there exits exactly one root that is a 0-2-node with its both sons being a 1-2-node and a 1-1-node that possesses itself one 2-1-node as son,
- each 1-2-node possesses exactly two 2-1-nodes or one 2-0-node and one 2-1-node as sons,
- each 2-1-node has one 1-2-node or 1-1-node as its son.
Those properties are acceptable by a dag automaton.

■

At a first sight, theorem 3 seems to point to a disadvantage of our concept of dag automata: type 1 ladders and type 2 ladders seem to be so similar that one might think that one type of ladders could result from the other by some "regular transformation"(and automata should preserve regular transformations). However, this is not the case. Type 1 and type 2 ladders have very distinct synchronization properties: An automaton may evaluate the upper row of a type-1-ladder ignoring the evaluation of the lower row, which is impossible for type 2 ladders. The situation is similar for Petri nets: There is a (rather simple) Petri net with $D_{2-ladder}$ as it true-concurrency dag semantics, but no Petri net can possess $D_{1-ladder}$ as its dag semantics, see [12].

In the above proof of theorem 3 it was shown that any dag automaton accepting $D_{1-ladder}$ must also accept some connected but not planar dag (that is not in $D_{1-ladder}$). Thus, $\mathcal{D}_{1-ladder}$ can't even be recognized relative to connected dags, i.e., if input dags are restricted to be connected. Both automata α, β of the proof of theorem 2 are planar. Thus, even if all input dags must be planar ${\Sigma^\dagger}_{con}$ is not regular. This implies

Corollary 4.1 *The language ${\Sigma^\dagger}_{plan}$ of all planar dags over Σ is not regular. ${\Sigma^\dagger}_{con}$ is not regular relative to ${\Sigma^\dagger}_{plan}$ and ${\Sigma^\dagger}_{plan}$ is not regular relative to ${\Sigma^\dagger}_{con}$.*

The *shuffle*

$$D_1 \parallel D_2 := \{\alpha_1 + \alpha_2 | \alpha_i \in D_i\}$$

of two dag languages consists of disjoint unions of one dag from D_1 with one from D_2. $\|^i$ and the big shuffle $\|^*$ are defined as

$$\|^0 D := \{\Im(\theta)\}, \ \ \|^{n+1} := (\|^n D) \parallel D, \ \ \|^* D := \bigcup_{i \geq 0} \|^n D.$$

It turns out that the big shuffle of even a single dag may be no regular language. Let $\Box$ be the beam of length 1, consisting of four nodes and being described by $a(1a + 2a) + a(1 + 2)$. $d := a(a1a + a1)$ describes a dag $\Diamond := \Im(d)$ with also only four nodes and four arcs as $\Box$ but only one root and leaf.

Theorem 4 *$\|^* \Box$ and $\|^* \Diamond$ are nonregular.*

Proof. $\|^* \Box$ is not regular: Assume some dag automaton A accepts $\|^* \Box$. Let $\alpha \in \|^* \Box$ be a union of a number of $\Box$ large enough that an accepting computation for α must attach the same multiset of states to the roots of two different occurrences o_1, o_2 of $\Box$. Let α' result from exchanging one son of o_1 with one son of o_2. α' is still acyclic but not in $\|^* \Box$. However, the mentioned accepting computation for α cannot sense this change and still accepts α_1. The argument for $\|^* \Diamond$ is similar.

■

This immediately implies:

Lemma 4.5 *The class of regular dag languages is closed under $\|^n$ for any n but not under $\|^*$.*

5 Deterministic DAG Automata

A dag automaton $A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$ is called *deterministic* if $|\delta(s, a)| = 1$ holds for $s \in Q, a \in \Sigma$, $I = \{s_0\}$ for one initial state s_0, there exists a sink state $\perp \in Q$ and $\delta_I(s_0, s_0) = s_0$, $\delta_I(.,.) = \perp$ elsewhere, $\delta_o(s, s) = s$ for some states $s \in Q$ and $\delta_o(.,.) = \perp$ elsewhere. Thus, to ensure a deterministic computation a start configuration for a dag α receives by δ_I the same state s_0 attached to all roots and the same state must be prolonged by δ_o from a father to all sons. As in a parse tree for a context-free derivation, the order of where to apply a transition in a configuration is still free, but $|eval^A(d)| = 1$ will hold. A regular dag language is called *deterministic regular* if it is accepted by some deterministic dag automaton.
If one applies a deterministic (root-to-frontier) dag automata to the reverse α^{rev} of an unranked, unordered tree α it behaves exactly as Courcelle's (frontiers-to-root) $\mathcal{T}_{uu}$-automata applied to α. Tree languages accepted by deterministic root-to-frontier tree automata are a proper subclass of those accepted by deterministic or nondeterministic frontier-to-root automata. Let $\triangleright = \Im(d)$ for $d = a1a + a1$. The trivial language $\{\triangleright\}$ is deterministic regular but $\{\triangleright\} \parallel \{\triangleright\}$ is not, as any deterministic automaton accepting $\triangleright + \triangleright = \Im(a1a + a1 + a2a + a2)$ must also accept $\square = \Im(a(1a + 2a) + a(1 + 2))$. Also, the regular languages $\Sigma^{\dagger}{}_{even}$ and $\Sigma^{\dagger}{}_{dis}$ are no longer deterministic regular. It is easily seen that the class of deterministic regular dag languages is closed under union, intersection and complement.
When a deterministic dag automaton passes a state from a father node to its sons it cannot react on the possibly different labels of the sons. Thus, deterministic dag automata are *forward blind*. One easily can define with the help of commutative and associative mappings a concept of not forward-blind deterministic root-to-frontier dag automata where the state passed to a son may depend on the state of the father and the multiset of labels of all sons. An *nfb regular* dag language is a regular dag language accepted by a deterministic not forward-blind dag automaton.
Table 1 presents some properties of the classes of regular, deterministic regular, nfb regular and semi rational dag languages. A dag language is semi rational if it is the dag semantics of some Petri net, see [13]. In contrast to word languages, dag languages accepted by finite automata must not necessarily be Petri net dag languages, see the last two lines of table 1.

6 Comparison to Further Automata Concepts

There are several concepts in the literature of finite automata analyzing graphs or dags with "local conditions". Kaminski and Pinter [11] operate on rooted directed graphs over a double ranked alphabet, and, thus, with a global bound for the in- and out-degree. With their automata $D_{1-ladder}$ is also not recognizable, see Thomas [16]. (One easily may regard $D_{1-ladder}$ as a language over a double ranked alphabet by using different labels for the upper and lower row). Thomas introduces "acceptors" on ranked graphs equivalent to existential monadic second-order logic (EMSL) on those graphs. Those acceptors simulate a tiling of a ranked graph with elementary graphs of a finite set of types plus some nonlocal constraints. $D_{1-ladder}$ becomes now acceptable relative to connected graphs. However, connectedness is expressible in MSL but not in EMSL, and thus not acceptable by those graph acceptors. It is hard to imagine how to generalize Thomas' acceptor concept to unranked graphs and languages with no fixed bound of the in- and out-degree.
It is also known that $D_{1-ladder}$ is acceptable relativ to a class of planar dags (pdags) of Bossut, Dauchet and Warin [3]. They introduce algebraic pdag expressions built from two-sorted letters and operations (iterated) *parallel* and *serial* composition. They can present an automaton that accepts all connected pdag expressions, in contrast to our theorem 2, corollary 4.1 and

Closed under:	$Reg^\dagger_{det}$	$Reg^\dagger_{nfb}$	$Reg^\dagger$	$SemiRat^\dagger$
union	✓	✓	✓	✓
intersection	✓	✓	✓	✓
complement	✓	✓	no	no
reverse	no	no	✓	✓
shuffle	no	no	✓	✓
big shuffle	no	no	no	no
finite sets	no	no	✓	✓
fat embedding of $\mathcal{L}_3$	✓	✓	✓	✓
contains: $\Sigma^\dagger_{dis}$	no	no	✓	✓
$D_{r=l}$	no	no	no	✓
$D_{n=}$	no	no	no	✓
can count modulo i the:				
nodes	no	no	✓	✓
roots	no	no	✓	✓
leaves	✓	✓	✓	✓
incoming arcs	✓	✓	✓	(no)
outgoing arcs	no	✓	✓	(no)

Table 1: Closure Properties, () is a Conjecture.

inexpressibility of connectedness in EMSL. This contradiction is resolved if one notes that their pdag expression cannot describe all planar dags, especially not all planar dags of Figure 4 that have been required for violating regularity of connectedness.

Résumé

We have introduced linear expressions that can describe all unranked and unordered abstract dags and finite automata on those unrestricted dags defining a class of regular dag languages. According to theorem 1, these dag automata operate on abstract dags as well as on dag expressions. The closure properties and examples of regular dag languages are just what should be expected from a reasonable concept of nondeterministic and deterministic dag automata. Although our approach is similar to the $\mathcal{T}_{u,u}$ approach of Courcelle we don't have a concept of rationality of dag languages by finite algebras, even not in the deterministic case. The reason is that in our concept of a computation the treatment of sorts l, t and f fits perfectly into the evaluation schema of algebras - such as to replace sa ba $\delta(s, a)$ or $s(f_1 + f_2)$ by $sf_1 + sf_2$. But for synchronization points a global view is involved as we replace si by s only if i occurs exactly once in the overall term. However, example 3.3 shows how close we are to algebras. The examples of regular and nonregular dag languages are very similar to those of semi-rational and not semi-rational ones in [13] although the concepts of regularity (acceptance by dag automata) and of semi-rationality (dag semantics of Petri nets) are rather different. This may be a hint that regularity and semi-rationality of dag languages indeed point more to inherent properties of dags than of the chosen concepts of automata and Petri nets.

References

[1] S. Anantharaman, P. Narendran, and M. Rusinowitch. Closure properties and decision problems of dag automata. *Information Processing Letters*, 94:231–240, 2005.

[2] I. Boneva and J.-M. Talbot. Automata and logic for unranked and unordered trees. *LNCS*, 3467:500–515, 2005.

[3] F. Bossut, M. Dauchet, and B. Warin. A Kleene theorem for a class of planar acyclic graphs. *Theor. Comp. Science Center Report HKUST-TCSC 2001-5*, 117:251–265, 1995.

[4] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and hedge languages of unranked alphabets. Theor. Comp. Science Center Report HKUST-TCSC 2001-5,pp29, 2001.

[5] W. Charatonik. Automata on dag representations of finite trees. Technical Report MPI-I-1999-2-001, MPI, Univ. SaarbrÃ$\frac{1}{4}$cken, 1999.

[6] H. Comon, M. Daucher, R. Gilleron, S. Tison, and M. Tommasi. Tree automata techniques and application. Available on the Web from 13ux02.univ-lille.fr in directoty tata, 1998.

[7] B. Courcelle. A representation of graphs by algebraic expressions and its use for graph rewriting systems. In *Proc. 3rd Internat. Workshop on Graph-Grammars, LNCS*, pages 112–132. Springer Verlag, 1988.

[8] B. Courcelle. On recognizable sets and tree automata. In H. Aït-Kaci, M. Nivat, editor, *Resolution of Equations in Algebraic Structures*, volume 1, pages 93–126. Academic Press, 1989.

[9] J. Fanchon and R. Morin. Regular sets of pomsets with auitoconcurrency. In *CONCUR 2002, LNCS 2421*, pages 402–417, 2002.

[10] T. Kamimura and G. Slutzki. Parallel and two-way automata on directed ordered acyclic graphs. *Inf. Control*, 49:10–51, 1981.

[11] M. Kaminski and S. Pinter. Finite automata on directed graphs. *J. Comp. Sys. Sci.*, 44:425–446, 1992.

[12] J.R. Menzel, L. Priese, and M. Schuth. Some examples of semi-rational dag languages. In *Developments in Language Theory: 10th International Conference, DLT 2006, St.Barbara, USA*, LNCS 4036, pages 351–362. Springer Verlag, 2006.

[13] L. Priese. Semi-rational sets of dags. In *Developments in Language Theory: 9th International Conference, DLT 2005, Palermo, Italy*, LNCS 3572, pages 385–396. Springer Verlag, 2005.

[14] L. Priese. Finite automata on unranked and unordered dags. In *Developments in Language Theory: 11th International Conference, DLT 2007, Turku, Finland*, LNCS 4588, pages 346–360. Springer Verlag, 2007.

[15] W. Thomas. Finite-state recognizability of graph properties. In D. Krob, editor, *Theorie des Automates et Applications*, volume 172, pages 147–159. l'Universite de Rouen, France, 1992.

[16] W. Thomas. Automata theory on trees and partial orders. In *Proc. 7th Intern. Joint Conference CAAP/FALSE: TAPSOFT'97, LNCS*, volume 1214, pages 20–34. Springer Verlag, 1997.