Finite State Automata for Directed Acyclic Graphs

Yvo Meeres

University of Leipzig

December 16th, 2021 15 CET
@ Séminaire 68NQRT
de l'IRISA et d'Inria Rennes

Part I

Motivation & Intuition

Motivation: Are FSAs capable of recognizing graph languages?

Can FSAs read graphs?

Starting point from literature: a limited graph automaton model.

Type of Graph

- DAG
  - non-empty
  - connected
  - vertex-labeled
  - non edge-labeled

Automaton Model

- Regular
  - top-down
  - deterministic
- whose runs read
  - by labeling edges
  - with states

Intuition

For the next 15 minutes of the talk,
you are an automaton.
You start as an ordinary DAG automaton.

But ...

you will become an FSA.
You will turn into a
Finite
State
Automaton.

You are so ... cool!

Regular DAG Automata

Definition

A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Regular DAG Automata

Definition

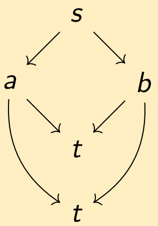
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Regular DAG Automata

Definition

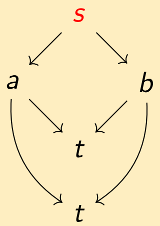
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Regular DAG Automata

Definition

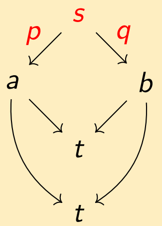
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Regular DAG Automata

Definition

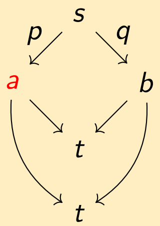
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Regular DAG Automata

Definition

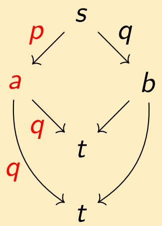
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Regular DAG Automata

Definition

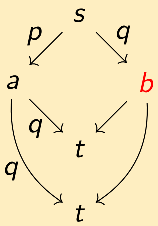
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Regular DAG Automata

Definition

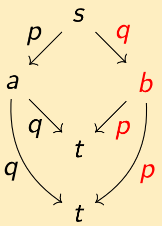
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Regular DAG Automata

Definition

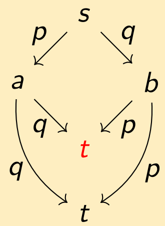
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Regular DAG Automata

Definition

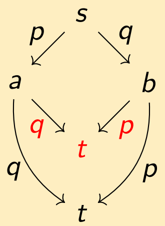
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Definition

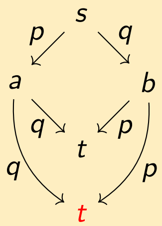
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



Regular DAG Automata

Definition

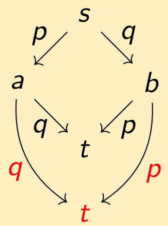
A *regular DAG automaton* is a triple $A = (Q, \Sigma, R)$ where

- Q is a finite set of states,
- Σ is a finite alphabet and
- R is a finite set of rules of the form $\alpha \twoheadrightarrow \sigma \twoheadrightarrow \beta$ where $\sigma \in \Sigma$ and $\alpha, \beta \in Q^*$.

Example

$$A = (\{p, q\}, \{s, a, b, t\}, R) \text{ where}$$

$$R = \{\lambda \twoheadrightarrow \mathtt{s} \twoheadrightarrow pq,\ p \twoheadrightarrow \mathtt{a} \twoheadrightarrow qq,\ q \twoheadrightarrow \mathtt{b} \twoheadrightarrow pp,\ qp \twoheadrightarrow \mathtt{t} \twoheadrightarrow \lambda\}$$



The Meta-state

A meta-state is the multiset of states assigned to edges with at least one unread neighbouring vertex.

Definition

A *meta-state* $\mathfrak{q}$ is an element of the multiset over Q, $\mathbb{N}^Q$. A derivation DAG G is a DAG with a (partial) run, thus (partially) labeled edges. We let $\underline{G}$ denote the meta-state of a derivation graph G. The set of all meta-states of $\mathcal{G}$ that occur in derivations of DAGs in $L(\mathcal{G})$ is denoted by $\mathcal{Q}(A)$.

Are FSAs capable of recognizing graph languages?

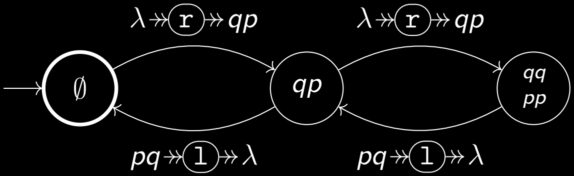
Can I, a DAG automaton,
turn myself into an FSA?

FSAs 4 DAGs ?

Yes, we can![1]

[1] https://en.wikipedia.org/wiki/File:Venezuelan_Sit_In_Si_Se_Puede.jpg

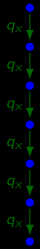
An FSA for a DAG Automaton



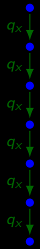
DAG languages

- Finite Induced Meta-States
  - STRING
  - STAR
  - GRASS
- Infinite Meta-States
  - SAW
  - RAINBOW
  - TREE
- Finite Meta-States by Limiting Meta-states
  - SAW
  - RAINBOW
  - TREE

STRING



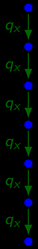
STRING



Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow q_1$$
$$q_2 \twoheadrightarrow \bigcirc \twoheadrightarrow q_x$$
$$q_x \twoheadrightarrow \bigcirc \twoheadrightarrow q_x$$
$$q_x \twoheadrightarrow \bigcirc \twoheadrightarrow q_x$$
$$q_x \twoheadrightarrow \bigcirc \twoheadrightarrow q_x$$
$$q_{|Q|} \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

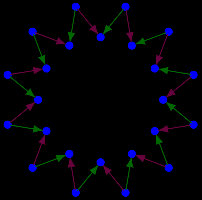
STRING

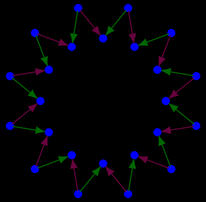


Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow q_1$$
$$q_2 \twoheadrightarrow \bigcirc \twoheadrightarrow q_x$$
$$q_x \twoheadrightarrow \bigcirc \twoheadrightarrow q_x$$
$$q_x \twoheadrightarrow \bigcirc \twoheadrightarrow q_x$$
$$q_x \twoheadrightarrow \bigcirc \twoheadrightarrow q_x$$
$$q_{|Q|} \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

STAR

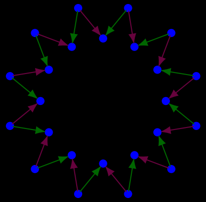




Rules R

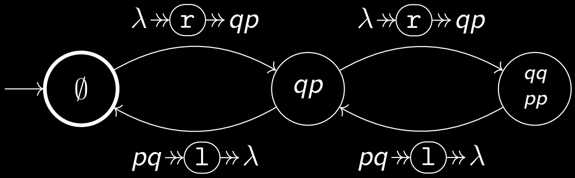
$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow qp$$
$$pq \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

STAR

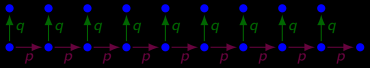


Rules R

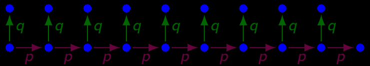
$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow qp$$
$$pq \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$



GRASS



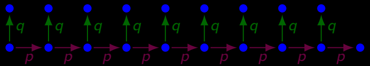
GRASS

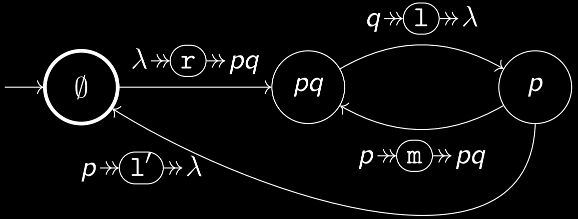


Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$q \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

GRASS

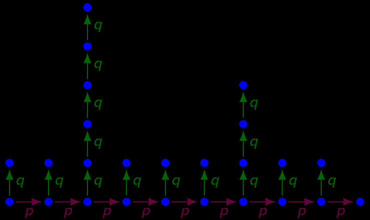


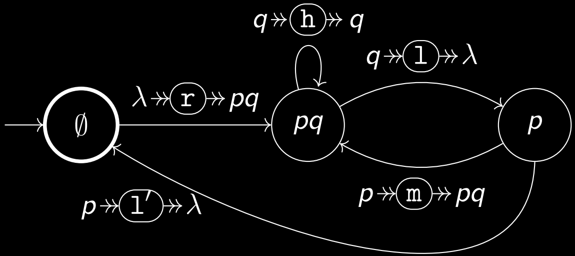


Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$q \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

GRASS





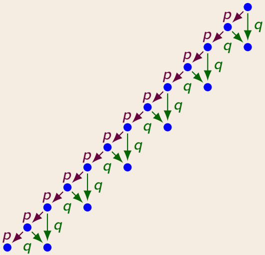
Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$q \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$
$$[\, q \twoheadrightarrow \bigcirc \twoheadrightarrow q \,]$$

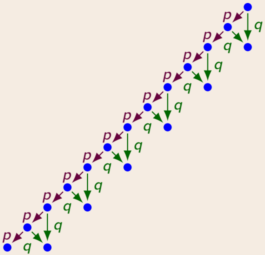
DAG languages

- Finite Induced Meta-States
  - STRING
  - STAR
  - GRASS
- Infinite Meta-States
  - SAW
  - RAINBOW
  - TREE
- Finite Meta-States by Limiting Meta-states
  - SAW
  - RAINBOW
  - TREE

SAW



SAW

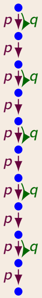


Rules R

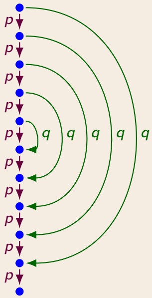
$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$qq \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

RAINBOW

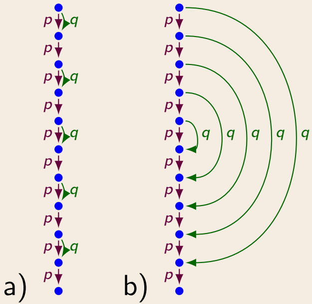
a)



b)



RAINBOW



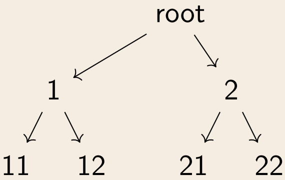
Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$pq \twoheadrightarrow \bigcirc \twoheadrightarrow p$$
$$pq \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

TREE

$$
\begin{tikzcd}
 & & & \text{root} \arrow[dll] \arrow[drr] & & & \\
 & 1 \arrow[dl] \arrow[dr] & & & & 2 \arrow[dl] \arrow[dr] & \\
11 & & 12 & & 21 & & 22
\end{tikzcd}
$$

TREE



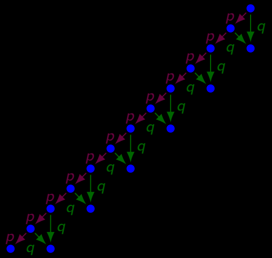
Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow qq$$
$$q \twoheadrightarrow \bigcirc \twoheadrightarrow qq$$
$$q \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

DAG languages

- Finite Induced Meta-States
  - STRING
  - STAR
  - GRASS
- Infinite Meta-States
  - SAW
  - RAINBOW
  - TREE
- Finite Meta-States by Limiting Meta-states
  - SAW
  - RAINBOW
  - TREE

SAW



Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$qq \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

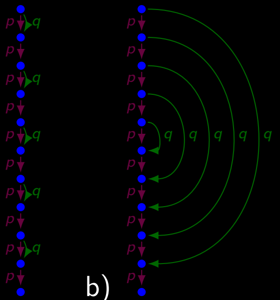
a) $p \twoheadrightarrow \bigcirc \twoheadrightarrow pq \qquad p \twoheadrightarrow \bigcirc \twoheadrightarrow pq$

b) $\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow pq \qquad p \twoheadrightarrow \bigcirc \twoheadrightarrow pq \qquad qq \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$

FSAs4DAGs
Y. Meeres
Finite
STRING
STAR
GRASS
Infinite
SAW
RAINBOW
TREE
Made Finite
SAW
RAINBOW
TREE

RAINBOW

a) b)



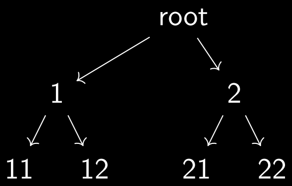
Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$p \twoheadrightarrow \bigcirc \twoheadrightarrow pq$$
$$pq \twoheadrightarrow \bigcirc \twoheadrightarrow p$$
$$pq \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$

a) $p \twoheadrightarrow \bigcirc \twoheadrightarrow pq \quad pq \twoheadrightarrow \bigcirc \twoheadrightarrow p \qquad p \twoheadrightarrow \bigcirc \twoheadrightarrow pq \quad pq \twoheadrightarrow \bigcirc \twoheadrightarrow p$

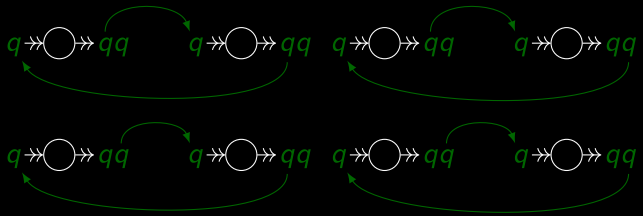
b) $p \twoheadrightarrow \bigcirc \twoheadrightarrow pq \quad p \twoheadrightarrow \bigcirc \twoheadrightarrow pq \qquad pq \twoheadrightarrow \bigcirc \twoheadrightarrow p \quad pq \twoheadrightarrow \bigcirc \twoheadrightarrow p$

TREE

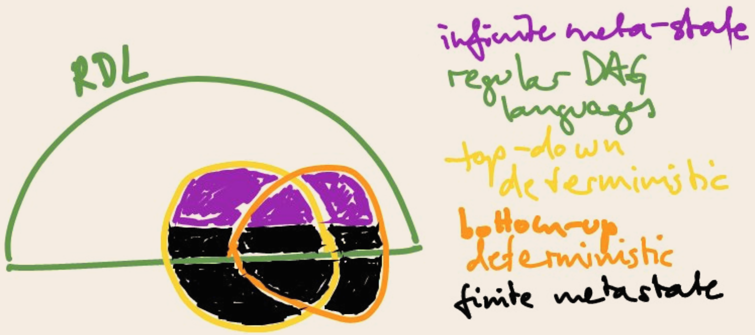


Rules R

$$\lambda \twoheadrightarrow \bigcirc \twoheadrightarrow qq$$
$$q \twoheadrightarrow \bigcirc \twoheadrightarrow qq$$
$$q \twoheadrightarrow \bigcirc \twoheadrightarrow \lambda$$



Language Hierarchy



Part II

The Formal Part

The Meta-states $\mathcal{Q}_{\text{min}}$ for the FSA

The smallest set of meta-states with which A can read all DAGs $L(A)$:

Definition

For a DAG automaton $A = (Q, \Sigma, R)$, we denote by $\mathcal{Q}_{\text{min}}(A)$ any set of meta-states such that

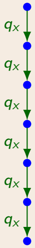
1. every DAG $G \in L(A)$ has a run including G_n, such that $\underline{G_0}, \ldots, \underline{G_n} \in \mathcal{Q}_{\text{min}}(A)$, and
2. there is no meta-state of smaller cardinality with this property.

Finite $\mathcal{Q}_{\min}$, finite $\mathcal{Q}_0$

$\mathcal{Q}(A)$ is the set-of all meta-states that can occur in a run for a DAG in $L(A)$.

Lemma

There exist DAG automata A for which both $\mathcal{Q}_{min}$ and $\mathcal{Q}(A)$ are finite.

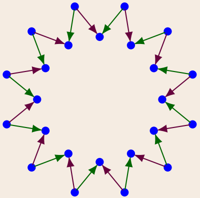


Finite $\mathcal{Q}_{\text{min}}$, infinite $\mathcal{Q}_0$

$\mathcal{Q}(A)$ is the set-of all meta-states that can occur in a run for a DAG in $L(A)$.

Lemma

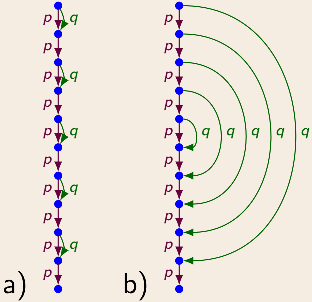
There exist DAG automata A for which $\mathcal{Q}_{min}(A)$ is finite whereas $\mathcal{Q}(A)$ is infinite.



Infinite $\mathcal{Q}_{\min}$, infinite $\mathcal{Q}_0$

Lemma

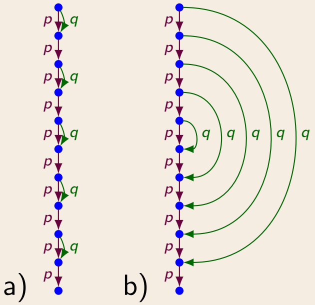
There exist DAG automata A for which both $\mathcal{Q}_{min}$ and $\mathcal{Q}(A)$ are infinite.



FD $\not\subseteq$ RDL^{det}

Lemma

Given is a minimal deterministic DAG automaton $A = (Q, \Sigma, R)$ and a finite set of meta-states $\mathcal{Q}$. Let $L^{\mathcal{Q}}(A)$ be the language generated by A if in a derivation step $G_1 \Rightarrow G_2$ is only allowed if the meta-state $\underline{G_2} \in \mathcal{Q}$. There exists a DAG language $L^{\mathcal{Q}}(A)$ that is not in the class of RDL^{det}.



Rule Cycle

Definition

A *rule cycle* is a nonempty sequence of marked rules $\hat{r}_1, \ldots, \hat{r}_k$ of A Such that, for all $i \in [k]$,

1. the exit state of $\hat{r}_i$ is equal to the entry state of $\hat{r}_{i \bmod k}$ and
2. $\hat{r}_i$ is tail exited if and only if $\hat{r}_{i \bmod k}$ is head entered.

The intuition is that a cycle is a sequence of rules in which each rule overlaps with the succeeding one in a cyclic fashion, i.e. modulo k.

Theorem (Theorem 6.4 of [1])

The DAG language generated by a DAG automaton $A = (Q, \Sigma, R)$ without useless rules is infinite iff R contains a rule cycle.

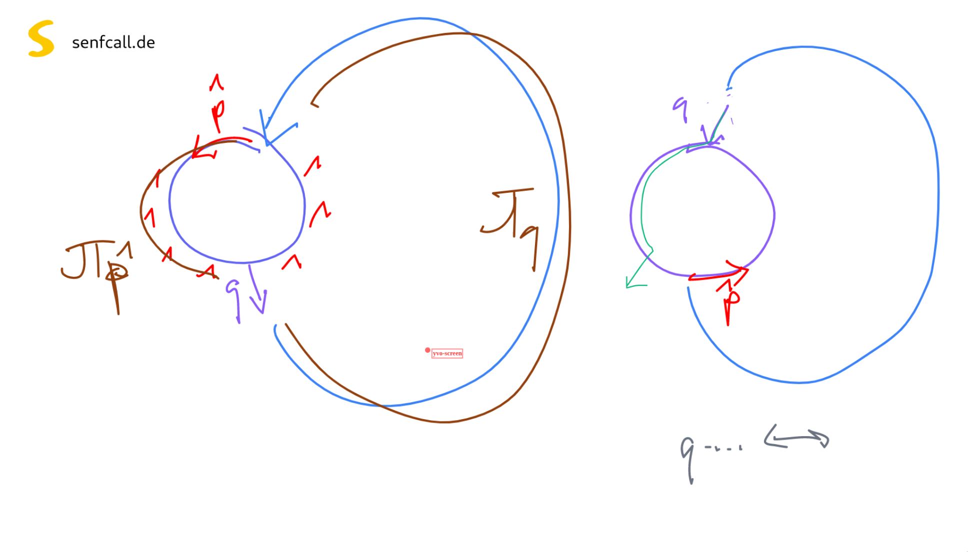
Characterization

Lemma

For a minimal deterministic DAG automaton $A = (Q, \Sigma, R)$ *its set of metastates* $\mathcal{Q}_{min}(A)$ *is infinite iff there exists a rule cycle* c *which satisfies the following conditions:*

1. *States* $q \in Q$ *and* $\hat{p} \in \hat{Q}$ *occur in* c *as unmarked and marked, resp.*
2. *A derivation DAG* D *exists with a rule path beteween* q *and* p *with* $\lfloor D \rfloor \in L(A)$.
3. *The path is from* q *to* p *iff* q *occurs in the tail* $\hat{\beta}$ *of one of* c*'s marked rules.*

Characterization Proof Sketch



MERCI

References

[1] Johannes Blum and Frank Drewes. "Language theoretic properties of regular DAG languages". In: *Inf. Comput.* 265 (2019), pp. 57–76. DOI: 10.1016/j.ic.2017.07.011. URL: https://doi.org/10.1016/j.ic.2017.07.011.