Algebraic Characterizations of Classes of Regular Languages in DynFO

Corentin Barloy, Felix Tschirbs, Nils Vortmeier, Thomas Zeume

Incremental maintenance and DynFO

$$\begin{array}{|c|c|c|c|c|c|}\hline \mathsf{a} & \mathsf{a} & \mathsf{b} & \mathsf{a} & \mathsf{b} & \mathsf{a} \\ \hline \end{array} \in (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$$

Incremental maintenance and DynFO

a	b	b	a	b	a

$\notin (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$

Incremental maintenance and DynFO

$\begin{array}{|c|c|c|c|c|c|}\hline \mathsf{a} & \mathsf{b} & \mathsf{a} & \mathsf{a} & \mathsf{b} & \mathsf{a} \\ \hline \end{array} \notin (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$

Incremental maintenance and DynFO

a	a	a	a	b	a

$\in (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$

Incremental maintenance and DynFO

$$\begin{array}{|c|c|c|c|c|c|}\hline \mathsf{a} & \mathsf{a} & \mathsf{a} & \mathsf{a} & \mathsf{b} & \mathsf{a} \\ \hline \end{array} \in (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$$

→ Descriptive approach DynFO: use tables updated by first-order formulas

Incremental maintenance and DynFO

$$\boxed{\text{a}}\boxed{\text{a}}\boxed{\text{a}}\boxed{\text{a}}\boxed{\text{b}}\boxed{\text{a}} \in (\texttt{aa})^*\texttt{b}(\texttt{a}+\texttt{b})^*$$

→ Descriptive approach DynFO: use tables updated by first-order formulas

→ Table T: store whether of the parity of the number of *a*s before every position is even.

Incremental maintenance and DynFO

a	a	a	a	b	a	$\in (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$

$\rightarrow$ Descriptive approach DynFO: use tables updated by first-order formulas

$\rightarrow$ Table T: store whether of the parity of the number of as before every position is even.

0	1	0	1	1	0

Incremental maintenance and DynFO

$$\begin{array}{|c|c|c|c|c|c|}\hline \mathsf{a} & \mathsf{a} & \mathsf{a} & \mathsf{a} & \mathsf{b} & \mathsf{a} \\ \hline \end{array} \in (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$$

→ Descriptive approach DynFO: use tables updated by first-order formulas

→ Table T: store whether of the parity of the number of as before every position is even.

$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array}$$

→ Update of j on change at i: $(j \geq i \Leftrightarrow \neg T(j))$

Incremental maintenance and DynFO

a	a	a	a	b	a

$\in (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$

→ Descriptive approach DynFO: use tables updated by first-order formulas

→ Table T: store whether of the parity of the number of as before every position is even.

0	1	0	1	1	0

→ Update of j on change at i: $(j \geq i \Leftrightarrow \neg T(j))$

→ Output: $\exists j, \text{First-b}(j) \wedge T(j)$

Incremental maintenance and DynFO

$$\begin{array}{|c|c|c|c|c|c|}\hline \mathsf{a} & \mathsf{b} & \mathsf{a} & \mathsf{a} & \mathsf{b} & \mathsf{a} \\ \hline \end{array} \notin (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$$

→ Descriptive approach DynFO: use tables updated by first-order formulas

→ Table T: store whether of the parity of the number of as before every position is even.

$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 0 & 1 & 0 & 0 & 1 \\ \hline \end{array}$$

→ Update of j on change at i: $(j \geq i \Leftrightarrow \neg T(j))$

→ Output: $\exists j, \text{First-b}(j) \wedge T(j)$

Incremental maintenance and DynFO

a	b	a	a	b	a

$\notin (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$

→ Descriptive approach DynFO: use tables updated by first-order formulas

→ Table T: store whether of the parity of the number of as before every position is even.

0	0	1	0	0	1

→ Update of j on change at i: $(j \geq i \Leftrightarrow \neg T(j))$

→ Output: $\exists j, \text{First-b}(j) \wedge T(j)$

→ This language is in DynFO but not in FO.

Incremental maintenance and DynFO

a	b	a	a	b	a

$\notin (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$

→ Descriptive approach DynFO: use tables updated by first-order formulas
→ Table T: store whether of the parity of the number of as before every position is even.

0	0	1	0	0	1

→ Update of j on change at i: $(j \geq i \Leftrightarrow \neg T(j))$
→ Output: $\exists j, \text{First-b}(j) \wedge T(j)$
→ This language is in DynFO but not in FO.

Fine-grained analysis:

Incremental maintenance and DynFO

a	b	a	a	b	a

$\notin (\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$

→ Descriptive approach DynFO: use tables updated by first-order formulas

→ Table T: store whether of the parity of the number of as before every position is even.

0	0	1	0	0	1

→ Update of j on change at i: $(j \geq i \Leftrightarrow \neg T(j))$

→ Output: $\exists j, \text{First-b}(j) \wedge T(j)$

→ This language is in DynFO but not in FO.

Fine-grained analysis:

→ Restricting alternations: DynProp, DynΣ_1, DynΣ_2, . . .

Incremental maintenance and DynFO

a	b	a	a	b	a

$\notin \mathtt{(aa)^*b(a+b)^*}$

→ Descriptive approach DynFO: use tables updated by first-order formulas

→ Table T: store whether of the parity of the number of *a*s before every position is even.

0	0	1	0	0	1

→ Update of j on change at i: $(j \geq i \Leftrightarrow \neg T(j))$

→ Output: $\exists j, \text{First-b}(j) \wedge T(j)$

→ This language is in DynFO but not in FO.

Fine-grained analysis:

→ Restricting alternations: DynProp, DynΣ_1, DynΣ_2, ...

→ Restricting tables arity: UDynProp, UDynΣ_1, UDynΣ_2, ...

Regular languages

Already studied: DynProp = Reg [Gelade, Marquardt, Schwentick 2012]

Regular languages

Already studied: DynProp = Reg [Gelade, Marquardt, Schwentick 2012]

The proof only uses binary tables for a DFA A:

Regular languages

Already studied: DynProp = Reg [Gelade, Marquardt, Schwentick 2012]

The proof only uses binary tables for a DFA A:
→ for every pair of states p, q, $T_{p,q}$ stores all couples $j \leq k$ such that $w[j,k]$ ends in q starting from p.

Regular languages

Already studied: DynProp = Reg [Gelade, Marquardt, Schwentick 2012]

The proof only uses binary tables for a DFA A:
→ for every pair of states p, q, $T_{p,q}$ stores all couples $j \leq k$ such that $w[j,k]$ ends in q starting from p.
→ Output: disjunction of $T_{i,f}(\min, \max)$ for i initial and f final.

Regular languages

Already studied: DynProp = Reg [Gelade, Marquardt, Schwentick 2012]

The proof only uses binary tables for a DFA A:

→ for every pair of states p, q, $T_{p,q}$ stores all couples $j \leq k$ such that $w[j, k]$ ends in q starting from p.

→ Output: disjunction of $T_{i,f}(\min, \max)$ for i initial and f final.

→ Update of (j, k) for change a at i:

→ if $j \leq i \leq k$: disjunction over $r \xrightarrow{a} r'$ of $T_{p,r} \wedge T_{r',q}$.

→ else: do nothing.

Regular languages

Already studied: DynProp = Reg [Gelade, Marquardt, Schwentick 2012]

The proof only uses binary tables for a DFA A:
→ for every pair of states p, q, $T_{p,q}$ stores all couples $j \leq k$ such that $w[j,k]$ ends in q starting from p.
→ Output: disjunction of $T_{i,f}(\min, \max)$ for i initial and f final.
→ Update of (j,k) for change a at i:
→ if $j \leq i \leq k$: disjunction over $r \xrightarrow{a} r'$ of $T_{p,r} \wedge T_{r',q}$.
→ else: do nothing.

We restrict our attention to unary DynFO

Regular languages

Already studied: DynProp = Reg [Gelade, Marquardt, Schwentick 2012]

The proof only uses binary tables for a DFA A:
$\rightarrow$ for every pair of states p, q, $T_{p,q}$ stores all couples $j \leq k$ such that $w[j, k]$ ends in q starting from p.
$\rightarrow$ Output: disjunction of $T_{i,f}(\min, \max)$ for i initial and f final.
$\rightarrow$ Update of (j, k) for change a at i:
$\rightarrow$ if $j \leq i \leq k$: disjunction over $r \xrightarrow{a} r'$ of $T_{p,r} \wedge T_{r',q}$.
$\rightarrow$ else: do nothing.

We restrict our attention to unary DynFO
Already known: Reg $\subseteq$ UDynFO [Hesse 2003]

Regular languages

Already studied: DynProp = Reg [Gelade, Marquardt, Schwentick 2012]

The proof only uses binary tables for a DFA A:
→ for every pair of states p, q, $T_{p,q}$ stores all couples $j \leq k$ such that $w[j,k]$ ends in q starting from p.
→ Output: disjunction of $T_{i,f}(\min, \max)$ for i initial and f final.
→ Update of (j, k) for change a at i:

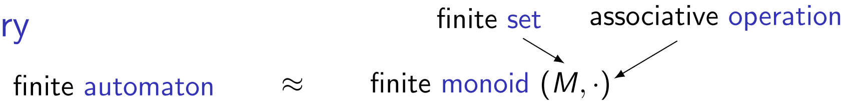
- → if $j \leq i \leq k$: disjunction over $r \xrightarrow{a} r'$ of $T_{p,r} \wedge T_{r',q}$.
- → else: do nothing.

We restrict our attention to unary DynFO
Already known: Reg ⊆ UDynFO [Hesse 2003]
→ We refine this with algebra!

The algebraic theory

finite automaton $\approx$ finite monoid $(M, \cdot)$

The algebraic theory



finite set

associative operation

finite automaton $\approx$ finite monoid $(M, \cdot)$

The algebraic theory

finite set

associative operation

finite automaton $\approx$ finite monoid $(M, \cdot)$

Take $\mu\colon \{a, b\} \to M$ extended to Σ^* by $\mu(a_1 \cdots a_n) = \mu(a_1) \cdots \mu(a_n)$
$\to$ L recognized by μ: $L = \mu^{-1}(P)$ for $P \subseteq M$

The algebraic theory

finite set

associative operation

finite automaton $\approx$ finite monoid $(M, \cdot)$

recognize same languages

Take $\mu\colon \{a, b\} \to M$ extended to Σ^* by $\mu(a_1 \cdots a_n) = \mu(a_1) \cdots \mu(a_n)$
$\to$ L recognized by μ: $L = \mu^{-1}(P)$ for $P \subseteq M$

The algebraic theory

finite set

associative operation

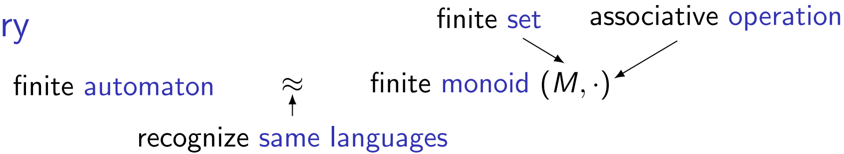
finite automaton $\approx$ finite monoid $(M, \cdot)$

recognize same languages

Take $\mu\colon \{a, b\} \to M$ extended to Σ^* by $\mu(a_1 \cdots a_n) = \mu(a_1) \cdots \mu(a_n)$
$\to$ L recognized by μ: $L = \mu^{-1}(P)$ for $P \subseteq M$

$(\mathtt{aa})^*\mathtt{b}(\mathtt{a} + \mathtt{b})^*$ is recognized by $M = \{a, aa, aab, ab\}$.

The algebraic theory



finite set

associative operation

finite automaton $\approx$ finite monoid $(M, \cdot)$

recognize same languages

Take $\mu\colon \{a, b\} \to M$ extended to Σ^* by $\mu(a_1 \cdots a_n) = \mu(a_1) \cdots \mu(a_n)$
$\to$ L recognized by μ: $L = \mu^{-1}(P)$ for $P \subseteq M$

$(\mathtt{aa})^*\mathtt{b}(\mathtt{a} + \mathtt{b})^*$ is recognized by $M = \{a, aa, aab, ab\}$.
$\to$ records if there is a b and the parity of the number of as before the first b.

The algebraic theory

finite set

associative operation

finite automaton $\approx$ finite monoid $(M, \cdot)$

recognize same languages

Take $\mu\colon \{a, b\} \to M$ extended to Σ^* by $\mu(a_1 \cdots a_n) = \mu(a_1) \cdots \mu(a_n)$

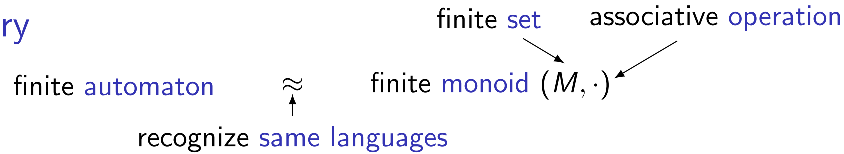
$\to$ L recognized by μ: $L = \mu^{-1}(P)$ for $P \subseteq M$

$(\mathtt{aa})^*\mathtt{b}(\mathtt{a} + \mathtt{b})^*$ is recognized by $M = \{a, aa, aab, ab\}$.

$\to$ records if there is a b and the parity of the number of as before the first b.

$\to$ The operation is defined accordingly. (ex: $a \cdot aab = ab$ and $aab \cdot a = aab$)

The algebraic theory



finite set

associative operation

finite automaton $\approx$ finite monoid $(M, \cdot)$

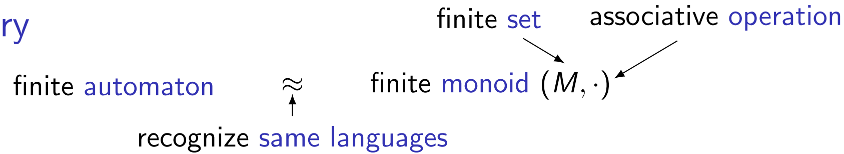
recognize same languages

Take $\mu\colon \{a, b\} \to M$ extended to Σ^* by $\mu(a_1 \cdots a_n) = \mu(a_1) \cdots \mu(a_n)$
$\to$ L recognized by μ: $L = \mu^{-1}(P)$ for $P \subseteq M$

$(\mathtt{aa})^*\mathtt{b}(\mathtt{a} + \mathtt{b})^*$ is recognized by $M = \{a, aa, aab, ab\}$.
$\to$ records if there is a b and the parity of the number of as before the first b.
$\to$ The operation is defined accordingly. (ex: $a \cdot aab = ab$ and $aab \cdot a = aab$)

Syntactic monoid of a language: smallest monoid recognizing it.

The algebraic theory

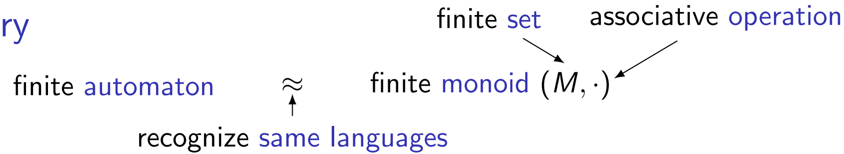


Take $\mu\colon \{a,b\} \to M$ extended to Σ^* by $\mu(a_1 \cdots a_n) = \mu(a_1) \cdots \mu(a_n)$
→ L recognized by μ: $L = \mu^{-1}(P)$ for $P \subseteq M$

$(\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$ is recognized by $M = \{a, aa, aab, ab\}$.
→ records if there is a b and the parity of the number of as before the first b.
→ The operation is defined accordingly. (ex: $a \cdot aab = ab$ and $aab \cdot a = aab$)

Syntactic monoid of a language: smallest monoid recognizing it.
→ Given its minimal automaton, it corresponds to the set of transition functions with composition.

The algebraic theory



finite set

associative operation

finite automaton $\approx$ finite monoid $(M, \cdot)$

recognize same languages

Take $\mu\colon \{a, b\} \to M$ extended to Σ^* by $\mu(a_1 \cdots a_n) = \mu(a_1) \cdots \mu(a_n)$
$\to$ L recognized by μ: $L = \mu^{-1}(P)$ for $P \subseteq M$

$(\mathtt{aa})^*\mathtt{b}(\mathtt{a} + \mathtt{b})^*$ is recognized by $M = \{a, aa, aab, ab\}$.
$\to$ records if there is a b and the parity of the number of as before the first b.
$\to$ The operation is defined accordingly. (ex: $a \cdot aab = ab$ and $aab \cdot a = aab$)

Syntactic monoid of a language: smallest monoid recognizing it.
$\to$ Given its minimal automaton, it corresponds to the set of transition functions with composition.
$\to$ Exhibit algebraic properties instead of combinatorial ones.

The algebraic theory

finite set

associative operation

finite automaton $\approx$ finite monoid $(M, \cdot)$

recognize same languages

Take $\mu\colon \{a, b\} \to M$ extended to Σ^* by $\mu(a_1 \cdots a_n) = \mu(a_1) \cdots \mu(a_n)$
$\to$ L recognized by μ: $L = \mu^{-1}(P)$ for $P \subseteq M$

$(\mathtt{aa})^*\mathtt{b}(\mathtt{a} + \mathtt{b})^*$ is recognized by $M = \{a, aa, aab, ab\}$.
$\to$ records if there is a b and the parity of the number of as before the first b.
$\to$ The operation is defined accordingly. (ex: $a \cdot aab = ab$ and $aab \cdot a = aab$)

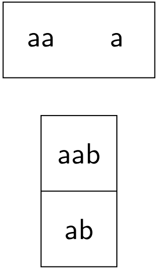
Syntactic monoid of a language: smallest monoid recognizing it.
$\to$ Given its minimal automaton, it corresponds to the set of transition functions with composition.
$\to$ Exhibit algebraic properties instead of combinatorial ones.

Previous proof of Reg $\subseteq$ DynFO: Maintain the evaluation of infixes in a monoid.

The regular languages of $\mathrm{UDyn}\Sigma_2$

Ideal structure

A division-based representation of the syntactic monoid of $(\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$:



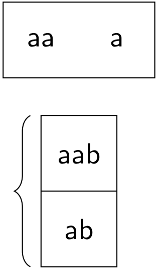
aa a

aab

ab

Ideal structure

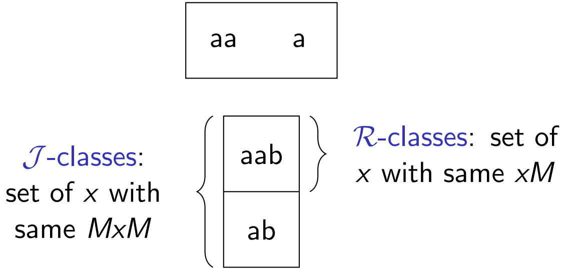
A division-based representation of the syntactic monoid of $(\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$:



$\mathcal{J}$-classes:
set of x with
same MxM

Ideal structure

A division-based representation of the syntactic monoid of $(\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$:



aa a

aab

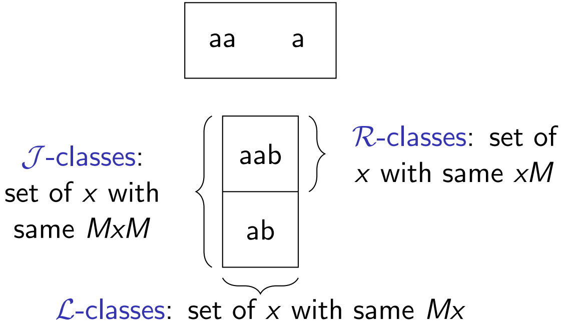
ab

$\mathcal{J}$-classes: set of x with same MxM

$\mathcal{R}$-classes: set of x with same xM

Ideal structure

A division-based representation of the syntactic monoid of $(\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$:



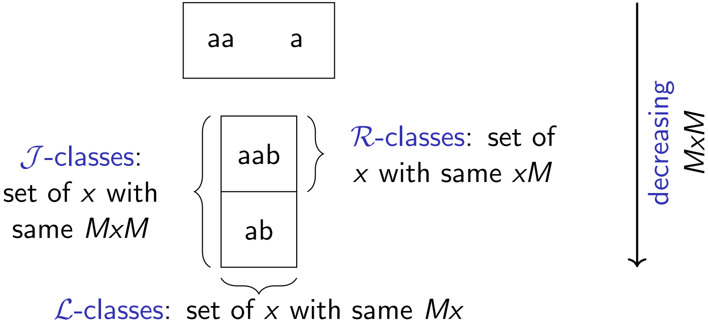
$\mathcal{J}$-classes: set of x with same MxM

$\mathcal{R}$-classes: set of x with same xM

$\mathcal{L}$-classes: set of x with same Mx

Ideal structure

A division-based representation of the syntactic monoid of $(\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$:



aa a

aab

ab

$\mathcal{J}$-classes: set of x with same MxM

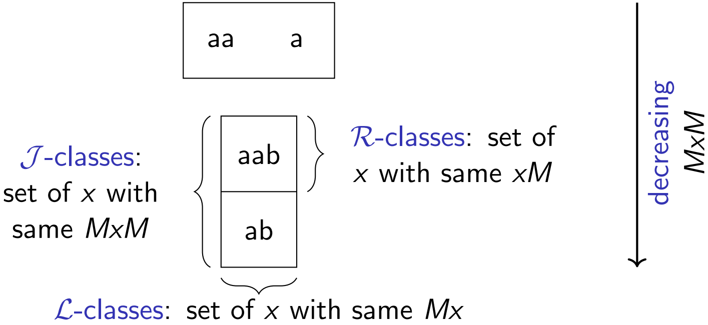
$\mathcal{R}$-classes: set of x with same xM

$\mathcal{L}$-classes: set of x with same Mx

decreasing MxM

Ideal structure

A division-based representation of the syntactic monoid of $(\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$:



aa a

aab

ab

$\mathcal{J}$-classes: set of x with same MxM

$\mathcal{R}$-classes: set of x with same xM

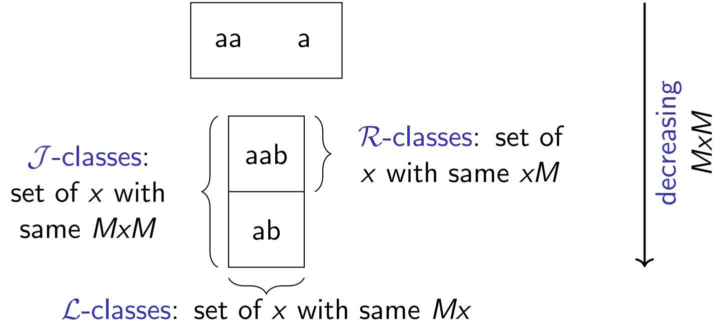
$\mathcal{L}$-classes: set of x with same Mx

decreasing MxM

→ If an infix evaluates in J, then the whole word evaluates $\geq J$.

Ideal structure

A division-based representation of the syntactic monoid of $(\mathtt{aa})^*\mathtt{b}(\mathtt{a}+\mathtt{b})^*$:



aa a

aab

ab

$\mathcal{J}$-classes: set of x with same MxM

$\mathcal{R}$-classes: set of x with same xM

$\mathcal{L}$-classes: set of x with same Mx

decreasing MxM

→ If an infix evaluates in J, then the whole word evaluates $\geq J$.

→ Every word has a decomposition of the form:

$$\underbrace{\underbrace{\underbrace{w_1 \cdots\cdots w_{l_1}}_{w_1 \;\; \mathcal{J} \;\; x_1 \;\; <_{\mathcal{J}} \;\; x_1 w_{l_1+1}} \; w_{l_1+1} \cdots\cdots w_{l_2}}_{\substack{\mathcal{J} \\ x_2 \quad <_{\mathcal{J}} \quad x_2 w_{l_2+1}}} \; w_{l_2+1} \cdots\cdots w_n}_{\substack{\vdots \\ x_m}}$$

The power of UDynΣ_2

Theorem

All regular languages are in UDynΣ_2.

The power of $\mathrm{UDyn}\Sigma_2$

Theorem

All regular languages are in $\mathrm{UDyn}\Sigma_2$.

Proof sketch: We show that we can compute the evaluation in any monoid M.

The power of $\mathrm{UDyn}\Sigma_2$

Theorem
All regular languages are in $\mathrm{UDyn}\Sigma_2$.

Proof sketch: We show that we can compute the evaluation in any monoid M.

A table $R_{J,x}$ for all $\mathcal{J}$-class J and $x \in M$.
→ contains i iff the greatest infix in J starting at i evaluates to x.
→ Similarly: tables $L_{J,x}$

The power of UDynΣ_2

Theorem
All regular languages are in UDynΣ_2.

Proof sketch: We show that we can compute the evaluation in any monoid M.

A table $R_{J,x}$ for all $\mathcal{J}$-class J and $x \in M$.
→ contains i iff the greatest infix in J starting at i evaluates to x.
→ Similarly: tables $L_{J,x}$

We can check if $w[i,j]$ evaluates to x in Σ_2:
$\exists i = l_1 < \cdots < l_m = j$,
→ $\forall l_k \leq j < l_{k+1}$, there is no jump in $\mathcal{J}$-class at j (thanks to L)
→ there is a jump in $\mathcal{J}$-class at each l_k (thanks to L)
→ the overall evaluation is x (thanks to R, and more work!)

The power of UDynΣ_2

Theorem
All regular languages are in UDynΣ_2.

Proof sketch: We show that we can compute the evaluation in any monoid M.

A table $R_{J,x}$ for all $\mathcal{J}$-class J and $x \in M$.
→ contains i iff the greatest infix in J starting at i evaluates to x.
→ Similarly: tables $L_{J,x}$

We can check if $w[i,j]$ evaluates to x in Σ_2:
$\exists i = l_1 < \cdots < l_m = j$,
→ $\forall l_k \leq j < l_{k+1}$, there is no jump in $\mathcal{J}$-class at j (thanks to L)
→ there is a jump in $\mathcal{J}$-class at each l_k (thanks to L)
→ the overall evaluation is x (thanks to R, and more work!)

Thus we can answer membership in Σ_2

The power of UDynΣ_2

Theorem
All regular languages are in UDynΣ_2.

Proof sketch: We show that we can compute the evaluation in any monoid M.

A table $R_{J,x}$ for all $\mathcal{J}$-class J and $x \in M$.
→ contains i iff the greatest infix in J starting at i evaluates to x.
→ Similarly: tables $L_{J,x}$

We can check if $w[i,j]$ evaluates to x in Σ_2:
$\exists i = l_1 < \cdots < l_m = j$,
→ $\forall l_k \leq j < l_{k+1}$, there is no jump in $\mathcal{J}$-class at j (thanks to L)
→ there is a jump in $\mathcal{J}$-class at each l_k (thanks to L)
→ the overall evaluation is x (thanks to R, and more work!)

Thus we can answer membership in Σ_2
Updates of $R_{J,x}$ at i: there is an index j such that $w[i,j]$ evaluates to x and $w[i,j+1]$ is $> J$.

The regular languages of UDynProp

Varieties

We used: maintain monoids $\Rightarrow$ maintain all recognized languages

Varieties

We used: maintain monoids $\Rightarrow$ maintain all recognized languages
$\rightarrow$ We want the converse.

Varieties

We used: maintain monoids $\Rightarrow$ maintain all recognized languages
$\rightarrow$ We want the converse.

Definition (Variety)

A set of languages is a variety if it is closed under:

- Boolean operations ($\cup$, $\cap$ and L^c),
- quotients ($a^{-1}L$ and La^{-1}),
- inverse morphisms ($\mu^{-1}(L)$).

Varieties

We used: maintain monoids $\Rightarrow$ maintain all recognized languages
$\rightarrow$ We want the converse.

Definition (Variety)

A set of languages is a variety if it is closed under:

- Boolean operations ($\cup$, $\cap$ and L^c),
- quotients ($a^{-1}L$ and La^{-1}),
- inverse morphisms ($\mu^{-1}(L)$).

UDynProp is a variety.

Varieties

We used: maintain monoids $\Rightarrow$ maintain all recognized languages
$\rightarrow$ We want the converse.

Definition (Variety)

A set of languages is a variety if it is closed under:

- Boolean operations ($\cup$, $\cap$ and L^c),
- quotients ($a^{-1}L$ and La^{-1}),
- inverse morphisms ($\mu^{-1}(L)$).

UDynProp is a variety.

Theorem

Membership of a language in UDynFO only depends on its syntactic monoid.

Varieties

We used: maintain monoids ⇒ maintain all recognized languages
→ We want the converse.

Definition (Variety)

A set of languages is a variety if it is closed under:

- Boolean operations ($\cup$, $\cap$ and L^{c}),
- quotients ($a^{-1}L$ and La^{-1}),
- inverse morphisms ($\mu^{-1}(L)$).

UDynProp is a variety.

Theorem

Membership of a language in UDynFO only depends on its syntactic monoid.

Usefull to define a class of languages by their syntactic monoids

Varieties

We used: maintain monoids ⇒ maintain all recognized languages
→ We want the converse.

Definition (Variety)

A set of languages is a variety if it is closed under:

- Boolean operations ($\cup$, $\cap$ and L^c),
- quotients ($a^{-1}L$ and La^{-1}),
- inverse morphisms ($\mu^{-1}(L)$).

UDynProp is a variety.

Theorem
Membership of a language in UDynFO only depends on its syntactic monoid.

Usefull to define a class of languages by their syntactic monoids
→ **G** is the class of languages whose syntactic monoid is a group

The power of UDynProp

Theorem

$\mathsf{UDynProp} \cap \mathsf{Reg} = \mathbf{G}$

The power of UDynProp

Theorem

$\text{UDynProp} \cap \text{Reg} = \mathbf{G}$

Upper bound: In a group, the evaluation of $w[i,j]$ only depends on $w[1,i]$ and $w[1,j]$.

The power of UDynProp

Theorem
$\mathsf{UDynProp} \cap \mathsf{Reg} = \mathbf{G}$

Upper bound: In a group, the evaluation of $w[i,j]$ only depends on $w[1,i]$ and $w[1,j]$.
→ Only maintain evaluation prefixes is enough to retrieve all infixes

The power of UDynProp

Theorem

$\text{UDynProp} \cap \text{Reg} = \mathbf{G}$

Upper bound: In a group, the evaluation of $w[i,j]$ only depends on $w[1,i]$ and $w[1,j]$.
→ Only maintain evaluation prefixes is enough to retrieve all infixes
→ the naive algorithm can be improved with unary tables.

The power of UDynProp

Theorem
UDynProp $\cap$ Reg $=$ **G**

Upper bound: In a group, the evaluation of $w[i, j]$ only depends on $w[1, i]$ and $w[1, j]$.
→ Only maintain evaluation prefixes is enough to retrieve all infixes
→ the naive algorithm can be improved with unary tables.

Lower bound: How are the monoids that are not groups?

The power of UDynProp

Theorem

UDynProp $\cap$ Reg $=$ **G**

Upper bound: In a group, the evaluation of $w[i, j]$ only depends on $w[1, i]$ and $w[1, j]$.
→ Only maintain evaluation prefixes is enough to retrieve all infixes
→ the naive algorithm can be improved with unary tables.

Lower bound: How are the monoids that are not groups?
→ M is a group $\Leftrightarrow$ the identity is the only x such that $x^2 = x$

The power of UDynProp

Theorem
UDynProp $\cap$ Reg $=$ $\mathbf{G}$

Upper bound: In a group, the evaluation of $w[i,j]$ only depends on $w[1,i]$ and $w[1,j]$.
$\rightarrow$ Only maintain evaluation prefixes is enough to retrieve all infixes
$\rightarrow$ the naive algorithm can be improved with unary tables.

Lower bound: How are the monoids that are not groups?
$\rightarrow$ M is a group $\Leftrightarrow$ the identity is the only x such that $x^2 = x$
$\rightarrow$ Take $M \notin \mathbf{G}$ and $x \neq 1$ such that $x^2 = x$

The power of UDynProp

Theorem
UDynProp $\cap$ Reg $=$ $\mathbf{G}$

Upper bound: In a group, the evaluation of $w[i,j]$ only depends on $w[1,i]$ and $w[1,j]$.
$\rightarrow$ Only maintain evaluation prefixes is enough to retrieve all infixes
$\rightarrow$ the naive algorithm can be improved with unary tables.

Lower bound: How are the monoids that are not groups?
$\rightarrow$ M is a group $\Leftrightarrow$ the identity is the only x such that $x^2 = x$
$\rightarrow$ Take $M \notin \mathbf{G}$ and $x \neq 1$ such that $x^2 = x$
$\rightarrow$ Consider $\mu : \{a, b\}^* \rightarrow M$ such that $\mu(b) = 1$ and $\mu(a) = x$

The power of UDynProp

Theorem
UDynProp $\cap$ Reg $= \mathbf{G}$

Upper bound: In a group, the evaluation of $w[i,j]$ only depends on $w[1,i]$ and $w[1,j]$.
$\rightarrow$ Only maintain evaluation prefixes is enough to retrieve all infixes
$\rightarrow$ the naive algorithm can be improved with unary tables.

Lower bound: How are the monoids that are not groups?
$\rightarrow$ M is a group $\Leftrightarrow$ the identity is the only x such that $x^2 = x$
$\rightarrow$ Take $M \notin \mathbf{G}$ and $x \neq 1$ such that $x^2 = x$
$\rightarrow$ Consider $\mu : \{a, b\}^* \rightarrow M$ such that $\mu(b) = 1$ and $\mu(a) = x$
$\rightarrow$ It recognizes $(\mathtt{a} + \mathtt{b})^*\mathtt{a}(\mathtt{a} + \mathtt{b})^*$

The power of UDynProp

Theorem
UDynProp $\cap$ Reg $=$ $\mathbf{G}$

Upper bound: In a group, the evaluation of $w[i,j]$ only depends on $w[1,i]$ and $w[1,j]$.
→ Only maintain evaluation prefixes is enough to retrieve all infixes
→ the naive algorithm can be improved with unary tables.

Lower bound: How are the monoids that are not groups?
→ M is a group $\Leftrightarrow$ the identity is the only x such that $x^2 = x$
→ Take $M \notin \mathbf{G}$ and $x \neq 1$ such that $x^2 = x$
→ Consider $\mu : \{a,b\}^* \to M$ such that $\mu(b) = 1$ and $\mu(a) = x$
→ It recognizes $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$
→ This language in not in UDynProp [Schwentick, Zeume 2015]

The regular languages of UDynΣ_1^+

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.
→ All formulas used so far had no negations.

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.
→ All formulas used so far had no negations.

UDynΣ_1^+ (and UDynΣ_1) are not closed under complement.

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.
→ All formulas used so far had no negations.

UDynΣ_1^+ (and UDynΣ_1) are not closed under complement.

Definition
A set of languages is a positive variety if it is closed under: Union, intersection, quotients and inverse morphisms.

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.
→ All formulas used so far had no negations.

UDynΣ_1^+ (and UDynΣ_1) are not closed under complement.

Definition
A set of languages is a positive variety if it is closed under: Union, intersection, quotients and inverse morphisms.

→ UDynΣ_1^+ is a positive variety

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.
→ All formulas used so far had no negations.

$\mathrm{UDyn}\Sigma_1^+$ (and $\mathrm{UDyn}\Sigma_1$) are not closed under complement.

Definition
A set of languages is a positive variety if it is closed under: Union, intersection, quotients and inverse morphisms.

→ $\mathrm{UDyn}\Sigma_1^+$ is a positive variety

Membership does not depend only on the syntactic monoid.

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.
→ All formulas used so far had no negations.

$\mathrm{UDyn}\Sigma_1^+$ (and $\mathrm{UDyn}\Sigma_1$) are not closed under complement.

Definition
A set of languages is a positive variety if it is closed under: Union, intersection, quotients and inverse morphisms.

→ $\mathrm{UDyn}\Sigma_1^+$ is a positive variety

Membership does not depend only on the syntactic monoid.

Definition
An ordered monoid is a monoid equiped with an order $\leq$.

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.
→ All formulas used so far had no negations.

$\mathrm{UDyn}\Sigma_1^+$ (and $\mathrm{UDyn}\Sigma_1$) are not closed under complement.

Definition
A set of languages is a positive variety if it is closed under: Union, intersection, quotients and inverse morphisms.

→ $\mathrm{UDyn}\Sigma_1^+$ is a positive variety

Membership does not depend only on the syntactic monoid.

Definition
An ordered monoid is a monoid equiped with an order $\leq$.

→ L is recognized by $(M, \leq)$ if there is an upset P and a morphism μ st $L = \mu^{-1}(P)$

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.
→ All formulas used so far had no negations.

$\mathrm{UDyn}\Sigma_1^+$ (and $\mathrm{UDyn}\Sigma_1$) are not closed under complement.

Definition
A set of languages is a positive variety if it is closed under: Union, intersection, quotients and inverse morphisms.

→ $\mathrm{UDyn}\Sigma_1^+$ is a positive variety

Membership does not depend only on the syntactic monoid.

Definition
An ordered monoid is a monoid equiped with an order $\leq$.

→ L is recognized by $(M, \leq)$ if there is an upset P and a morphism μ st $L = \mu^{-1}(P)$
→ There is a notion of syntactic ordered monoid

Positive varieties

Σ_1^+: formulas of the form $\exists x_1, \cdots, x_k,\ \varphi$ where φ has no negations.
→ All formulas used so far had no negations.

UDynΣ_1^+ (and UDynΣ_1) are not closed under complement.

Definition
A set of languages is a positive variety if it is closed under: Union, intersection, quotients and inverse morphisms.

→ UDynΣ_1^+ is a positive variety

Membership does not depend only on the syntactic monoid.

Definition
An ordered monoid is a monoid equiped with an order $\leq$.

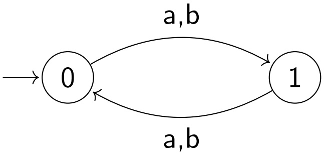
→ L is recognized by $(M, \leq)$ if there is an upset P and a morphism μ st $L = \mu^{-1}(P)$
→ There is a notion of syntactic ordered monoid
→ Membership of a language in a positive variety only depends on its syntactic ordered monoid

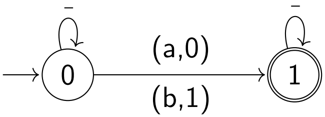
Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

Wreath products

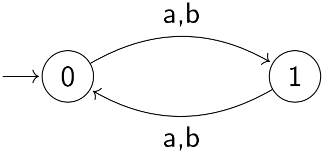
Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

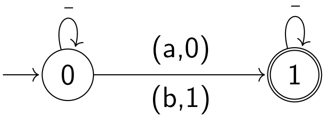




Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

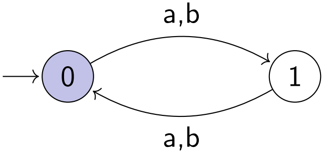


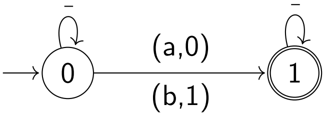


a	b	a	b	b	b

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

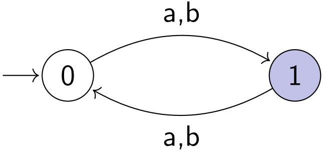


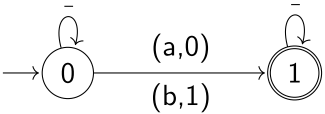


a	b	a	b	b	b

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.



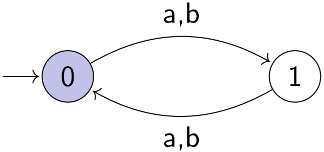


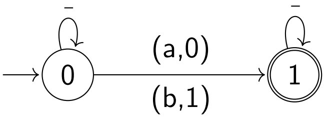
↓

a	b	a	b	b	b
1					

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.



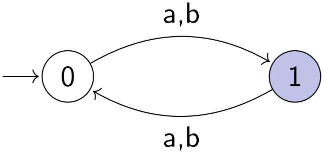


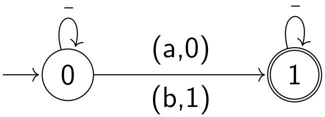
↓

a	b	a	b	b	b
1	0				

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.



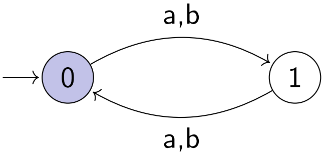


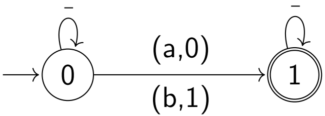
↓

a	b	a	b	b	b
1	0	1			

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.



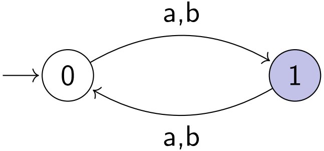


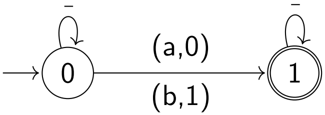
↓

a	b	a	b	b	b
1	0	1	0		

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

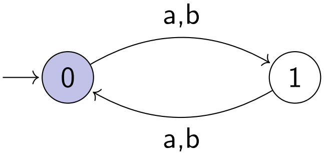


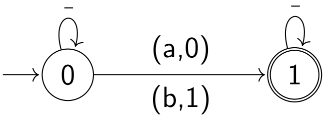


				↓	
a	b	a	b	b	b
1	0	1	0	1	

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

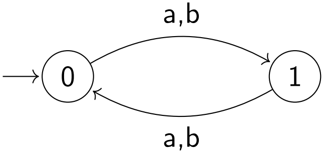


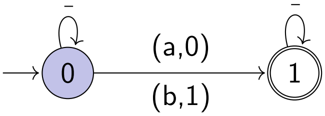


a	b	a	b	b	b
1	0	1	0	1	0

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

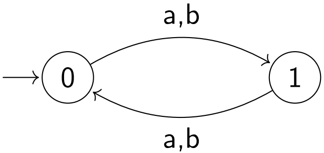


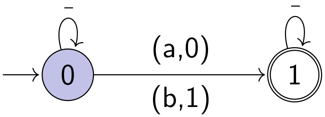


a	b	a	b	b	b
1	0	1	0	1	0

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.



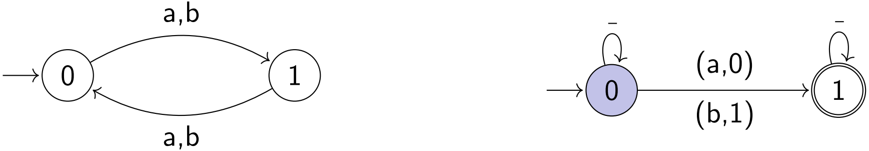


↓

a	b	a	b	b	b
1	0	1	0	1	0

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

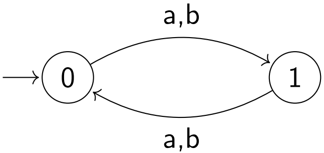


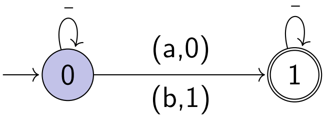
↓

a	b	a	b	b	b
1	0	1	0	1	0

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.



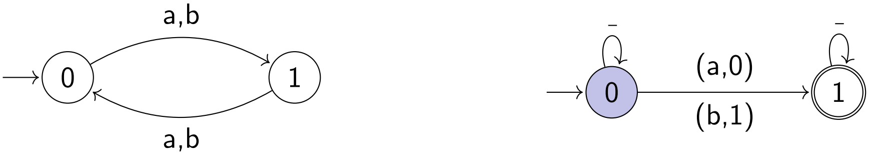


↓

a	b	a	b	b	b
1	0	1	0	1	0

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

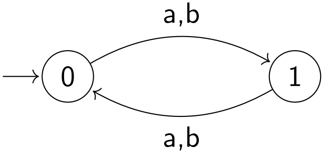


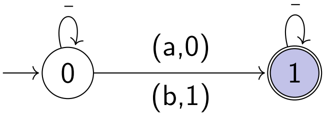
↓

a	b	a	b	b	b
1	0	1	0	1	0

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.

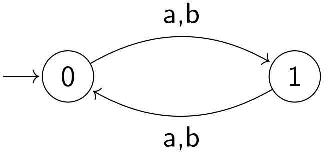


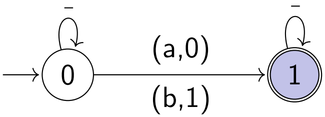


				↓	
a	b	a	b	b	b
1	0	1	0	1	0

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.



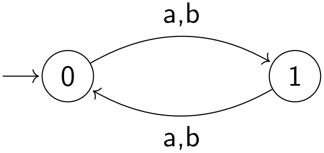


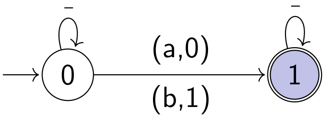
↓

a	b	a	b	b	b
1	0	1	0	1	0

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.





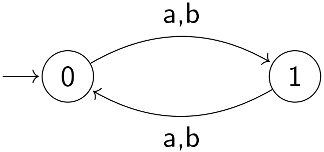
↓

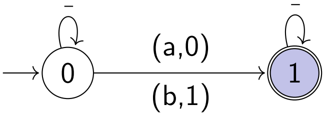
a	b	a	b	b	b
1	0	1	0	1	0

Wreath product: Algebraic counterpart of the sequential composition, denoted $*$

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.





↓

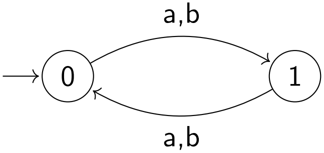
a	b	a	b	b	b
1	0	1	0	1	0

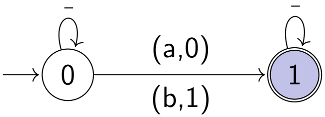
Wreath product: Algebraic counterpart of the sequential composition, denoted $*$

$\mathbf{J}^{+}$ is the class of languages whose syntactic ordered monoid satisfy $1 \leq x$, for all x

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.





↓

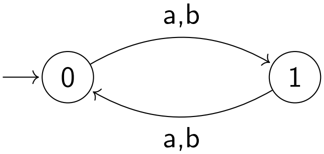
a	b	a	b	b	b
1	0	1	0	1	0

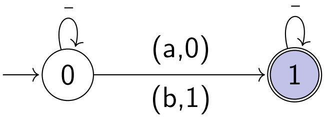
Wreath product: Algebraic counterpart of the sequential composition, denoted $*$

$\mathbf{J}^+$ is the class of languages whose syntactic ordered monoid satisfy $1 \leq x$, for all x
→ Captures Σ_1

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.





↓

a	b	a	b	b	b
1	0	1	0	1	0

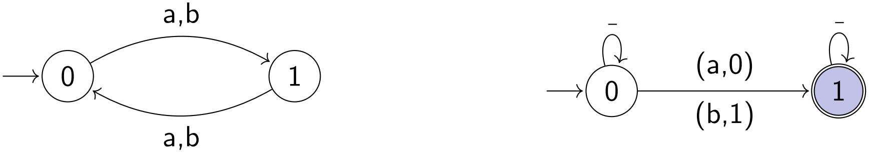
Wreath product: Algebraic counterpart of the sequential composition, denoted $*$

$\mathbf{J}^+$ is the class of languages whose syntactic ordered monoid satisfy $1 \leq x$, for all x
$\rightarrow$ Captures Σ_1

We will look at $\mathbf{J}^+ * \mathbf{G}$

Wreath products

Sequential composition of automata $\mathcal{A}_1$ and $\mathcal{A}_2$: on input w, label w by the states it reaches in $\mathcal{A}_1$ and feed it to $\mathcal{A}_2$.



↓

a	b	a	b	b	b
1	0	1	0	1	0

Wreath product: Algebraic counterpart of the sequential composition, denoted $*$

$\mathbf{J}^+$ is the class of languages whose syntactic ordered monoid satisfy $1 \leq x$, for all x
→ Captures Σ_1

We will look at $\mathbf{J}^+ * \mathbf{G}$
→ The complement of $(\mathtt{ab})^*$ is inside.

The power of $\mathsf{UDyn}\Sigma_1^+$

Theorem

$\mathsf{UDyn}\Sigma_1^+ \cap \mathsf{Reg} = \mathbf{J}^+ * \mathbf{G}$

The power of $\mathsf{UDyn}\Sigma_1^+$

Theorem

$\mathsf{UDyn}\Sigma_1^+ \cap \mathsf{Reg} = \mathbf{J}^+ * \mathbf{G}$

Upper bound: we can maintain with Prop the evaluation in a group of all prefixes

The power of $\mathrm{UDyn}\Sigma_1^+$

Theorem

$\mathrm{UDyn}\Sigma_1^+ \cap \mathrm{Reg} = \mathbf{J}^+ * \mathbf{G}$

Upper bound: we can maintain with Prop the evaluation in a group of all prefixes
→ we can maintain the word labeled by a group

The power of $\mathrm{UDyn}\Sigma_1^+$

Theorem

$\mathrm{UDyn}\Sigma_1^+ \cap \mathrm{Reg} = \mathbf{J}^+ * \mathbf{G}$

Upper bound: we can maintain with Prop the evaluation in a group of all prefixes

→ we can maintain the word labeled by a group

→ a Σ_1^+ formula can take care of the $\mathbf{J}^+$ part

The power of UDynΣ_1^+

Theorem

UDyn$\Sigma_1^+ \cap \mathrm{Reg} = \mathbf{J}^+ * \mathbf{G}$

Upper bound: we can maintain with Prop the evaluation in a group of all prefixes
→ we can maintain the word labeled by a group
→ a Σ_1^+ formula can take care of the $\mathbf{J}^+$ part

Lower bound: lot of work on wreath product by $\mathbf{G}$ [Almeida, Escada 2002] [Pin, Weil 2002]

The power of $\mathrm{UDyn}\Sigma_1^+$

Theorem
$\mathrm{UDyn}\Sigma_1^+ \cap \mathrm{Reg} = \mathbf{J}^+ * \mathbf{G}$

Upper bound: we can maintain with Prop the evaluation in a group of all prefixes
→ we can maintain the word labeled by a group
→ a Σ_1^+ formula can take care of the $\mathbf{J}^+$ part

Lower bound: lot of work on wreath product by $\mathbf{G}$ [Almeida, Escada 2002] [Pin, Weil 2002]
→ $(M, \leq)$ is in $\mathbf{J}^+ * \mathbf{G} \Leftrightarrow$ for all x such that $x^2 = x$, we have $x \geq 1$

The power of $\mathrm{UDyn}\Sigma_1^+$

Theorem

$\mathrm{UDyn}\Sigma_1^+ \cap \mathrm{Reg} = \mathbf{J}^+ * \mathbf{G}$

Upper bound: we can maintain with Prop the evaluation in a group of all prefixes
→ we can maintain the word labeled by a group
→ a Σ_1^+ formula can take care of the $\mathbf{J}^+$ part

Lower bound: lot of work on wreath product by $\mathbf{G}$ [Almeida, Escada 2002] [Pin, Weil 2002]
→ $(M, \leq)$ is in $\mathbf{J}^+ * \mathbf{G}$ $\Leftrightarrow$ for all x such that $x^2 = x$, we have $x \geq 1$
→ Take $(M, \leq) \notin \mathbf{J}^+ * \mathbf{G}$ and x such that $x \not\geq 1$

The power of $\mathrm{UDyn}\Sigma_1^+$

Theorem

$\mathrm{UDyn}\Sigma_1^+ \cap \mathrm{Reg} = \mathbf{J}^+ * \mathbf{G}$

Upper bound: we can maintain with Prop the evaluation in a group of all prefixes

→ we can maintain the word labeled by a group

→ a Σ_1^+ formula can take care of the $\mathbf{J}^+$ part

Lower bound: lot of work on wreath product by $\mathbf{G}$ [Almeida, Escada 2002] [Pin, Weil 2002]

→ $(M, \leq)$ is in $\mathbf{J}^+ * \mathbf{G} \Leftrightarrow$ for all x such that $x^2 = x$, we have $x \geq 1$

→ Take $(M, \leq) \notin \mathbf{J}^+ * \mathbf{G}$ and x such that $x \not\geq 1$

→ Consider $\mu : \{a, b\}^* \to (M, \leq)$ such that $\mu(b) = 1$ and $\mu(a) = x$

The power of $\mathrm{UDyn}\Sigma_1^+$

Theorem

$\mathrm{UDyn}\Sigma_1^+ \cap \mathrm{Reg} = \mathbf{J}^+ * \mathbf{G}$

Upper bound: we can maintain with Prop the evaluation in a group of all prefixes
→ we can maintain the word labeled by a group
→ a Σ_1^+ formula can take care of the $\mathbf{J}^+$ part

Lower bound: lot of work on wreath product by $\mathbf{G}$ [Almeida, Escada 2002] [Pin, Weil 2002]
→ $(M, \leq)$ is in $\mathbf{J}^+ * \mathbf{G} \Leftrightarrow$ for all x such that $x^2 = x$, we have $x \geq 1$
→ Take $(M, \leq) \notin \mathbf{J}^+ * \mathbf{G}$ and x such that $x \not\geq 1$
→ Consider $\mu : \{a, b\}^* \to (M, \leq)$ such that $\mu(b) = 1$ and $\mu(a) = x$
→ It recognizes b^*: x is not in the upset of 1

The power of $\mathrm{UDyn}\Sigma_1^+$

Theorem

$\mathrm{UDyn}\Sigma_1^+ \cap \mathrm{Reg} = \mathbf{J}^+ * \mathbf{G}$

Upper bound: we can maintain with Prop the evaluation in a group of all prefixes
→ we can maintain the word labeled by a group
→ a Σ_1^+ formula can take care of the $\mathbf{J}^+$ part

Lower bound: lot of work on wreath product by $\mathbf{G}$ [Almeida, Escada 2002] [Pin, Weil 2002]
→ $(M, \leq)$ is in $\mathbf{J}^+ * \mathbf{G}$ $\Leftrightarrow$ for all x such that $x^2 = x$, we have $x \geq 1$
→ Take $(M, \leq) \notin \mathbf{J}^+ * \mathbf{G}$ and x such that $x \not\geq 1$
→ Consider $\mu : \{a, b\}^* \to (M, \leq)$ such that $\mu(b) = 1$ and $\mu(a) = x$
→ It recognizes b^*: x is not in the upset of 1
→ This language can be shown to not be in $\mathrm{UDyn}\Sigma_1^+$

Conclusion

The only problem left is to identify the regular languages of $\mathrm{UDyn}\Sigma_1$.

Conclusion

The only problem left is to identify the regular languages of $\mathrm{UDyn}\Sigma_1$.
→ Strictly more powerful than $\mathrm{UDyn}\Sigma_1^+$: witnessed by $\mathtt{b}^*$ and $(\mathtt{a}+\mathtt{b})^*\mathtt{aa}(\mathtt{a}+\mathtt{b})^*$.

Conclusion

The only problem left is to identify the regular languages of $\mathrm{UDyn}\Sigma_1$.
$\rightarrow$ Strictly more powerful than $\mathrm{UDyn}\Sigma_1^+$: witnessed by $\mathtt{b}^*$ and $(\mathtt{a}+\mathtt{b})^*\mathtt{aa}(\mathtt{a}+\mathtt{b})^*$.
$\rightarrow$ Conjecture: $\mathbf{J} * \mathbf{G} \subsetneq \mathrm{UDyn}\Sigma_1 \subseteq \mathbf{\Sigma_2} * \mathbf{G}$

Conclusion

The only problem left is to identify the regular languages of $\mathrm{UDyn}\Sigma_1$.

→ Strictly more powerful than $\mathrm{UDyn}\Sigma_1^+$: witnessed by $\mathtt{b}^*$ and $(\mathtt{a}+\mathtt{b})^*\mathtt{aa}(\mathtt{a}+\mathtt{b})^*$.

→ Conjecture: $\mathbf{J} * \mathbf{G} \subsetneq \mathrm{UDyn}\Sigma_1 \subseteq \mathbf{\Sigma_2} * \mathbf{G}$

→ We lack lower bounds!

Conclusion

The only problem left is to identify the regular languages of $\mathrm{UDyn}\Sigma_1$.
→ Strictly more powerful than $\mathrm{UDyn}\Sigma_1^+$: witnessed by $\mathtt{b}^*$ and $(\mathtt{a}+\mathtt{b})^*\mathtt{aa}(\mathtt{a}+\mathtt{b})^*$.
→ Conjecture: $\mathbf{J} * \mathbf{G} \subsetneq \mathrm{UDyn}\Sigma_1 \subseteq \mathbf{\Sigma_2} * \mathbf{G}$
→ We lack lower bounds!

Recap:

Dynamic class	UDynProp	$\mathrm{UDyn}\Sigma_1^+$	$\mathrm{UDyn}\Sigma_2$
Regular languages	$\mathbf{G}$	$\mathbf{J}^+ * \mathbf{G}$	Reg