Generalizing trace monoids

S. Karageorgieva, N. Kumar, R. Liu, A. Roze
Supervised by Prof. Azadeh Farzan

November 12, 2022

What Are Traces?

- Let Σ be a finite **alphabet** of **letters**.

What Are Traces?

- Let Σ be a finite **alphabet** of **letters**.
- Let $\mathbb{I}$ be an irreflexive symmetric relation (simple graph) with vertices in Σ^*.

What Are Traces?

- Let Σ be a finite **alphabet** of **letters**.
- Let $\mathbb{I}$ be an irreflexive symmetric relation (simple graph) with vertices in Σ^*.
- A **generalized trace monoid** is a monoid with the presentation

$$\langle \Sigma \mid uv = vu \text{ for} (u, v) \in \mathbb{I} \rangle.$$

Its elements are called **traces**.

What Are Traces?

- Let Σ be a finite **alphabet** of **letters**.
- Let $\mathbb{I}$ be an irreflexive symmetric relation (simple graph) with vertices in Σ^*.
- A **generalized trace monoid** is a monoid with the presentation

$$\langle \Sigma \mid uv = vu \text{ for} (u, v) \in \mathbb{I} \rangle.$$

  Its elements are called **traces**.
- *Reminder:* in the context of classical trace theory, $\mathbb{I}$ is a graph over Σ.

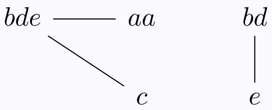
What Are Traces?

- Let Σ be a finite **alphabet** of **letters**.
- Let $\mathbb{I}$ be an irreflexive symmetric relation (simple graph) with vertices in Σ^*.
- A **generalized trace monoid** is a monoid with the presentation

$$\langle \Sigma \mid uv = vu \text{ for} (u, v) \in \mathbb{I} \rangle.$$

  Its elements are called **traces**.
- *Reminder:* in the context of classical trace theory, $\mathbb{I}$ is a graph over Σ.
- Example:

$$ebdaac = bdeaac$$
$$bdeaac = aabdec$$
$$aabdec = aacbde$$



Applications of Traces in Computer Science

- Words $\longleftrightarrow$ program executions

Applications of Traces in Computer Science

- Words $\longleftrightarrow$ program executions
- Languages $\longleftrightarrow$ sets of executions

Applications of Traces in Computer Science

- Words $\longleftrightarrow$ program executions
- Languages $\longleftrightarrow$ sets of executions
  - Models for programs

Applications of Traces in Computer Science

- Words $\longleftrightarrow$ program executions
- Languages $\longleftrightarrow$ sets of executions
  - Models for programs
  - Models for desirable properties of programs

Applications of Traces in Computer Science

- Words $\longleftrightarrow$ program executions
- Languages $\longleftrightarrow$ sets of executions
  - Models for programs
  - Models for desirable properties of programs
- Regular languages $\longleftrightarrow$ well-behaved models

Applications of Traces in Computer Science

- Words $\longleftrightarrow$ program executions
- Languages $\longleftrightarrow$ sets of executions
  - Models for programs
  - Models for desirable properties of programs
- Regular languages $\longleftrightarrow$ well-behaved models
  - Closure properties (complement, union, intersection, Kleene star)

Applications of Traces in Computer Science

- Words $\longleftrightarrow$ program executions
- Languages $\longleftrightarrow$ sets of executions
  - Models for programs
  - Models for desirable properties of programs
- Regular languages $\longleftrightarrow$ well-behaved models
  - Closure properties (complement, union, intersection, Kleene star)
  - Constant-space algorithm for checking membership

Applications of Traces in Computer Science

- Words $\longleftrightarrow$ program executions
- Languages $\longleftrightarrow$ sets of executions
  - Models for programs
  - Models for desirable properties of programs
- Regular languages $\longleftrightarrow$ well-behaved models
  - Closure properties (complement, union, intersection, Kleene star)
  - Constant-space algorithm for checking membership
  - Decidability of emptiness and language inclusion

Applications of Traces in Computer Science

- Words $\longleftrightarrow$ program executions
- Languages $\longleftrightarrow$ sets of executions
  - Models for programs
  - Models for desirable properties of programs
- Regular languages $\longleftrightarrow$ well-behaved models
  - Closure properties (complement, union, intersection, Kleene star)
  - Constant-space algorithm for checking membership
  - Decidability of emptiness and language inclusion
- Traces $\longleftrightarrow$ executions up to concurrency

The Goal of the Project

- Main goal:

The Goal of the Project

- Main goal:
  - Study one generalization of trace monoids.

The Goal of the Project

- Main goal:
  - Study one generalization of trace monoids.
  - Recover useful properties.

The Goal of the Project

- Main goal:
  - Study one generalization of trace monoids.
  - Recover useful properties.
- Two important languages:

The Goal of the Project

- Main goal:
  - Study one generalization of trace monoids.
  - Recover useful properties.
- Two important languages:
  1. The ordered language $\mathcal{L}_{x \prec y}$, for $x, y \in \Sigma$, contains all words $xuy \in \Sigma^*$ such that $\dot{x}u\dot{y}$ is not equivalent to any word of the form $v\dot{y}v'\dot{x}v''$.

The Goal of the Project

- Main goal:
  - Study one generalization of trace monoids.
  - Recover useful properties.
- Two important languages:
  1. The ordered language $\mathcal{L}_{x \prec y}$, for $x, y \in \Sigma$, contains all words $xuy \in \Sigma^*$ such that $\dot{x}u\dot{y}$ is not equivalent to any word of the form $v\dot{y}v'\dot{x}v''$.
     - ⋆ Can event $\dot{y}$ possibly occur before event $\dot{x}$?

The Goal of the Project

- Main goal:
  - Study one generalization of trace monoids.
  - Recover useful properties.
- Two important languages:
  1. The ordered language $\mathcal{L}_{x \prec y}$, for $x, y \in \Sigma$, contains all words $xuy \in \Sigma^*$ such that $\dot{x}u\dot{y}$ is not equivalent to any word of the form $v\dot{y}v'\dot{x}v''$.
     - Can event $\dot{y}$ possibly occur before event $\dot{x}$?
  2. The lexicographical language $Lex(\mathcal{L})$ contains the lexicographically least representative of each trace in the trace language $\mathcal{L}$.

The Goal of the Project

- Main goal:
  - Study one generalization of trace monoids.
  - Recover useful properties.
- Two important languages:
  1. The ordered language $\mathcal{L}_{x \prec y}$, for $x, y \in \Sigma$, contains all words $xuy \in \Sigma^*$ such that $\dot{x}u\dot{y}$ is not equivalent to any word of the form $v\dot{y}v'\dot{x}v''$.
     - ★ Can event $\dot{y}$ possibly occur before event $\dot{x}$?
  2. The lexicographical language $Lex(\mathcal{L})$ contains the lexicographically least representative of each trace in the trace language $\mathcal{L}$.
     - ★ Normal form which faithfully represents the original trace language.

The Goal of the Project

- Answer: No!

$$\langle \{a, b, c\} \mid bc = cb, bca = abc \rangle.$$

The Goal of the Project

- Answer: No!

$$\langle \{a, b, c\} \mid bc = cb, bca = abc \rangle.$$

Problem

How can we recover regularity of the languages $\mathcal{L}_{x \prec y}$ and $Lex(\mathcal{L})$ in the generalized setting?

The Goal of the Project

- Answer: No!

$$\langle \{a, b, c\} \mid bc = cb, bca = abc \rangle.$$

Problem

How can we recover regularity of the languages $\mathcal{L}_{x \prec y}$ and $Lex(\mathcal{L})$ in the generalized setting?

- What additional and sufficient conditions can we impose on our choice of $\mathbb{I}$ to retain regularity?

The Goal of the Project

- Answer: No!

$$\langle \{a, b, c\} \mid bc = cb, bca = abc \rangle.$$

Problem

How can we recover regularity of the languages $\mathcal{L}_{x \prec y}$ and $Lex(\mathcal{L})$ in the generalized setting?

- What additional and sufficient conditions can we impose on our choice of $\mathbb{I}$ to retain regularity?
- Is regularity preserved in simpler but related languages?

The Goal of the Project

- Answer: No!

$$\langle \{a, b, c\} \mid bc = cb, bca = abc \rangle.$$

Problem

How can we recover regularity of the languages $\mathcal{L}_{x \prec y}$ and $Lex(\mathcal{L})$ in the generalized setting?

- What additional and sufficient conditions can we impose on our choice of $\mathbb{I}$ to retain regularity?
- Is regularity preserved in simpler but related languages?
- What other results can we obtain that give us insight into the languages using generalized traces?

Conditions on $\mathbb{I}$

- Consider the relation $\mathbb{I} = \{(a, bc), (b, c)\}$. Then a commutes with all words of the form $(bc)^*$.

Conditions on $\mathbb{I}$

- Consider the relation $\mathbb{I} = \{(a, bc), (b, c)\}$. Then a commutes with all words of the form $(bc)^*$.
- However, take

$$abccb \leftrightarrow abcbc \leftrightarrow bcbca$$

Then a commutes with every word w in which $|w|_b = |w|_c$, which is not regular.

Conditions on $\mathbb{I}$

- Consider the relation $\mathbb{I} = \{(a, bc), (b, c)\}$. Then a commutes with all words of the form $(bc)^*$.
- However, take

$$abccb \leftrightarrow abcbc \leftrightarrow bcbca$$

  Then a commutes with every word w in which $|w|_b = |w|_c$, which is not regular.
- The irregularity arises from the fact that these words can be internally transformed into different words.

Conditions on $\mathbb{I}$

- Consider the relation $\mathbb{I} = \{(a, bc), (b, c)\}$. Then a commutes with all words of the form $(bc)^*$.
- However, take

$$abccb \leftrightarrow abcbc \leftrightarrow bcbca$$

  Then a commutes with every word w in which $|w|_b = |w|_c$, which is not regular.
- The irregularity arises from the fact that these words can be internally transformed into different words.
- Not a problem in the non-generalized case, since letters are discrete!

Conditions on $\mathbb{I}$

- Consider the relation $\mathbb{I} = \{(a, bc), (b, c)\}$. Then a commutes with all words of the form $(bc)^*$.
- However, take

$$abccb \leftrightarrow abcbc \leftrightarrow bcbca$$

  Then a commutes with every word w in which $|w|_b = |w|_c$, which is not regular.
- The irregularity arises from the fact that these words can be internally transformed into different words.
- Not a problem in the non-generalized case, since letters are discrete!
- Can we make strings in $\tilde{\mathbb{I}} = \{w \mid \exists v \text{ such that } (w, v) \in \mathbb{I}\}$ behave like letters?

Unique Decompositions

Yes! We identified conditions that allow words in $\tilde{\mathbb{I}}$ to behave like letters:

Unique Decompositions

Yes! We identified conditions that allow words in $\tilde{\mathbb{I}}$ to behave like letters:

Prefix-Suffix Condition

If $u, v \in \tilde{\mathbb{I}}$, then no **prefix** of u can be a **suffix** of v.

Unique Decompositions

Yes! We identified conditions that allow words in $\tilde{\mathbb{I}}$ to behave like letters:

Prefix-Suffix Condition

If $u, v \in \tilde{\mathbb{I}}$, then no **prefix** of u can be a **suffix** of v.

Partitioning Condition

We can partition Σ into $\Sigma_1 \cup \Sigma_2$, such that words in Σ_1^* can only commute with words in Σ_2^* (vice versa). Furthermore, no $v' \in \tilde{\mathbb{I}}$ is a prefix or suffix of $v \in \tilde{\mathbb{I}}$.

Unique Decompositions

Yes! We identified conditions that allow words in $\tilde{\mathbb{I}}$ to behave like letters:

Prefix-Suffix Condition

If $u, v \in \tilde{\mathbb{I}}$, then no **prefix** of u can be a **suffix** of v.

Partitioning Condition

We can partition Σ into $\Sigma_1 \cup \Sigma_2$, such that words in Σ_1^* can only commute with words in Σ_2^* (vice versa). Furthermore, no $v' \in \tilde{\mathbb{I}}$ is a prefix or suffix of $v \in \tilde{\mathbb{I}}$.

Weak Prefix-Suffix Condition

If $u, v \in \tilde{\mathbb{I}}$, then u cannot be a **prefix** nor **suffix** of v.

Unique Decompositions

Yes! We identified conditions that allow words in $\tilde{\mathbb{I}}$ to behave like letters:

Prefix-Suffix Condition

If $u, v \in \tilde{\mathbb{I}}$, then no **prefix** of u can be a **suffix** of v.

Partitioning Condition

We can partition Σ into $\Sigma_1 \cup \Sigma_2$, such that words in Σ_1^* can only commute with words in Σ_2^* (vice versa). Furthermore, no $v' \in \tilde{\mathbb{I}}$ is a prefix or suffix of $v \in \tilde{\mathbb{I}}$.

Weak Prefix-Suffix Condition

If $u, v \in \tilde{\mathbb{I}}$, then u cannot be a **prefix** nor **suffix** of v.

Result

We regain regularity of $\mathcal{L}_{x \prec y}$!

Prefix-Suffix Condition: Proof Sketch

- Consider words such that $w = w_1 w_2 w_3 \dots w_n$ where $w_i \in \tilde{\mathbb{I}}$.

Prefix-Suffix Condition: Proof Sketch

- Consider words such that $w = w_1 w_2 w_3 \ldots w_n$ where $w_i \in \tilde{\mathbb{I}}$.
- If such a decomposition exists, it is **unique** (easy observation).

Prefix-Suffix Condition: Proof Sketch

- Consider words such that $w = w_1 w_2 w_3 \ldots w_n$ where $w_i \in \tilde{\mathbb{I}}$.
- If such a decomposition exists, it is **unique** (easy observation).
- What if a swap happens?

Prefix-Suffix Condition: Proof Sketch

- Consider words such that $w = w_1 w_2 w_3 \ldots w_n$ where $w_i \in \tilde{\mathbb{I}}$.
- If such a decomposition exists, it is **unique** (easy observation).
- What if a swap happens?
  - Either we swap $w_i \leftrightarrow w_{i+1}$.

Prefix-Suffix Condition: Proof Sketch

- Consider words such that $w = w_1 w_2 w_3 \dots w_n$ where $w_i \in \tilde{\mathbb{I}}$.
- If such a decomposition exists, it is **unique** (easy observation).
- What if a swap happens?
  - Either we swap $w_i \leftrightarrow w_{i+1}$.
  - Or we swap something 'internal' to w_i.

Prefix-Suffix Condition: Proof Sketch

- Consider words such that $w = w_1 w_2 w_3 \ldots w_n$ where $w_i \in \tilde{\mathbb{I}}$.
- If such a decomposition exists, it is **unique** (easy observation).
- What if a swap happens?
  - Either we swap $w_i \leftrightarrow w_{i+1}$.
  - Or we swap something 'internal' to w_i.
  - We **cannot** break w_i, since that would violate the prefix-suffix condition!

Prefix-Suffix Condition: Proof Sketch

- Consider words such that $w = w_1 w_2 w_3 \dots w_n$ where $w_i \in \tilde{\mathbb{I}}$.
- If such a decomposition exists, it is **unique** (easy observation).
- What if a swap happens?
  - Either we swap $w_i \leftrightarrow w_{i+1}$.
  - Or we swap something 'internal' to w_i.
  - We **cannot** break w_i, since that would violate the prefix-suffix condition!
- It follows that the w_i behave like letters, and the question reduces to being able to move w_1 beyond w_n.

Prefix-Suffix Condition: Proof Sketch

- Consider words such that $w = w_1 w_2 w_3 \dots w_n$ where $w_i \in \tilde{\mathbb{I}}$.
- If such a decomposition exists, it is **unique** (easy observation).
- What if a swap happens?
  - Either we swap $w_i \leftrightarrow w_{i+1}$.
  - Or we swap something 'internal' to w_i.
  - We **cannot** break w_i, since that would violate the prefix-suffix condition!
- It follows that the w_i behave like letters, and the question reduces to being able to move w_1 beyond w_n.
- We can apply the same proof as the non-generalized case, essentially a homomorphism to the letter case.

Partitioning + Weak Prefix-Suffix Condition: Proof Sketch

- We can partition any word into $w = u_1v_1u_2v_2 \dots u_nv_n$ where $u_i \in \Sigma_1^*, v_i \in \Sigma_2^*$.

Partitioning + Weak Prefix-Suffix Condition: Proof Sketch

- We can partition any word into $w = u_1 v_1 u_2 v_2 \dots u_n v_n$ where $u_i \in \Sigma_1^*, v_i \in \Sigma_2^*$.
- Decomposition is again unique.

Partitioning + Weak Prefix-Suffix Condition: Proof Sketch

- We can partition any word into $w = u_1 v_1 u_2 v_2 \dots u_n v_n$ where $u_i \in \Sigma_1^*, v_i \in \Sigma_2^*$.
- Decomposition is again unique.
- We then check if each u_i and v_i can be broken into $\{u_{ij}\}_{j \in [k]}, \{v_{ij}\}_{j \in [k']}$, where the elements of each decomposition are in $\tilde{\mathbb{I}}$

$$u_1 v_1 \cdot u_2 \cdot v_2 \dots u_n v_n \equiv u_1 v_1 \cdot u_{2,1} \dots u_{2,k} \cdot v_2 \dots u_n v_n$$

Partitioning + Weak Prefix-Suffix Condition: Proof Sketch

- We can partition any word into $w = u_1 v_1 u_2 v_2 \ldots u_n v_n$ where $u_i \in \Sigma_1^*, v_i \in \Sigma_2^*$.
- Decomposition is again unique.
- We then check if each u_i and v_i can be broken into $\{u_{ij}\}_{j \in [k]}, \{v_{ij}\}_{j \in [k']}$, where the elements of each decomposition are in $\tilde{\mathbb{I}}$

$$u_1 v_1 \cdot u_2 \cdot v_2 \ldots u_n v_n \equiv u_1 v_1 \cdot u_{2,1} \ldots u_{2,k} \cdot v_2 \ldots u_n v_n$$

- Further consider each u_{ij} and $v_{i'j'}$ to be a block.

Partitioning + Weak Prefix-Suffix Condition: Proof Sketch

- We can partition any word into $w = u_1 v_1 u_2 v_2 \dots u_n v_n$ where $u_i \in \Sigma_1^*, v_i \in \Sigma_2^*$.
- Decomposition is again unique.
- We then check if each u_i and v_i can be broken into $\{u_{ij}\}_{j \in [k]}, \{v_{ij}\}_{j \in [k']}$, where the elements of each decomposition are in $\tilde{\mathbb{I}}$

$$u_1 v_1 \cdot u_2 \cdot v_2 \dots u_n v_n \equiv u_1 v_1 \cdot u_{2,1} \dots u_{2,k} \cdot v_2 \dots u_n v_n$$

- Further consider each u_{ij} and $v_{i'j'}$ to be a block.

Partitioning + Weak Prefix-Suffix Condition: Proof Sketch

- We can partition any word into $w = u_1 v_1 u_2 v_2 \dots u_n v_n$ where $u_i \in \Sigma_1^*, v_i \in \Sigma_2^*$.
- Decomposition is again unique.
- We then check if each u_i and v_i can be broken into $\{u_{ij}\}_{j \in [k]}, \{v_{ij}\}_{j \in [k']}$, where the elements of each decomposition are in $\tilde{\mathbb{I}}$

$$u_1 v_1 \cdot u_2 \cdot v_2 \dots u_n v_n \equiv u_1 v_1 \cdot u_{2,1} \dots u_{2,k} \cdot v_2 \dots u_n v_n$$

- Further consider each u_{ij} and $v_{i'j'}$ to be a block.
- Since elements of $\tilde{\mathbb{I}}$ cannot be prefixes or suffixes of each other, swaps are limited.

Partitioning + Weak Prefix-Suffix Condition: Proof Sketch

- We can partition any word into $w = u_1 v_1 u_2 v_2 \dots u_n v_n$ where $u_i \in \Sigma_1^*, v_i \in \Sigma_2^*$.
- Decomposition is again unique.
- We then check if each u_i and v_i can be broken into $\{u_{ij}\}_{j \in [k]}, \{v_{ij}\}_{j \in [k']}$, where the elements of each decomposition are in $\tilde{\mathbb{I}}$

$$u_1 v_1 \cdot u_2 \cdot v_2 \dots u_n v_n \equiv u_1 v_1 \cdot u_{2,1} \dots u_{2,k} \cdot v_2 \dots u_n v_n$$

- Further consider each u_{ij} and $v_{i'j'}$ to be a block.
- Since elements of $\tilde{\mathbb{I}}$ cannot be prefixes or suffixes of each other, swaps are limited.
- We can again replicate the same proof as the non-generalized case.

Unique Decompositions

- Idea of both these proofs: **Unique Decomposition**

Unique Decompositions

- Idea of both these proofs: **Unique Decomposition**
- We place conditions on $\mathbb{I}$ such that the elements of $\tilde{\mathbb{I}}$ behave like letters

Unique Decompositions

- Idea of both these proofs: **Unique Decomposition**
- We place conditions on $\mathbb{I}$ such that the elements of $\tilde{\mathbb{I}}$ behave like letters
- Essentially a reduction from the generalized case to the non-generalized case

Unique Decompositions

- Idea of both these proofs: **Unique Decomposition**
- We place conditions on $\mathbb{I}$ such that the elements of $\tilde{\mathbb{I}}$ behave like letters
- Essentially a reduction from the generalized case to the non-generalized case
- Can we apply other methods to understand regularity in this context without imposing additional structure to $\mathbb{I}$?

Simpler languages

Another approach is to consider simpler but related languages instead of imposing conditions on $\mathbb{I}$. Increasing the complexity of the languages we can get “closer” to the languages of interest.

Simpler languages

Another approach is to consider simpler but related languages instead of imposing conditions on $\mathbb{I}$. Increasing the complexity of the languages we can get "closer" to the languages of interest.

Theorem

Let $\mathbb{I}$ be **any** subset of $\Sigma^* \times \Sigma^*$

- $L_v := \{u \mid uv \equiv vu\}$ is a submonoid.

Simpler languages

Another approach is to consider simpler but related languages instead of imposing conditions on $\mathbb{I}$. Increasing the complexity of the languages we can get "closer" to the languages of interest.

Theorem

Let $\mathbb{I}$ be **any** subset of $\Sigma^* \times \Sigma^*$

- $L_v := \{u \mid uv \equiv vu\}$ is a submonoid.
- $L_v \cap w^*$ is regular for any $w \in \Sigma^*$.

Simpler languages

Another approach is to consider simpler but related languages instead of imposing conditions on $\mathbb{I}$. Increasing the complexity of the languages we can get "closer" to the languages of interest.

Theorem

Let $\mathbb{I}$ be **any** subset of $\Sigma^* \times \Sigma^*$

- $L_v := \{u \mid uv \equiv vu\}$ is a submonoid.
- $L_v \cap w^*$ is regular for any $w \in \Sigma^*$.
- $\{w_1^n w_2^m \mid w_2^n w_1^m \equiv w_2^m w_1^n\}$ is regular for any $w_1, w_2 \in \Sigma^*$.

Simpler Languages Cont.

Idea of the proof

- $L_v \cap w^*$ is a submonoid of $w^* \cong \mathbb{N}$, hence, it is of the form $(w^d)^* \setminus E$ for some d and some finite set E.

Simpler Languages Cont.

Idea of the proof

- $L_v \cap w^*$ is a submonoid of $w^* \cong \mathbb{N}$, hence, it is of the form $(w^d)^* \setminus E$ for some d and some finite set E.
- If we fix m, then $\{w_1^n \mid w_1^n w_2^m \equiv w_2^m w_1^n\}$ is regular and equal to $(w_1^{d_m})^* \setminus E_m$ for some d_m and finite E_m. We want to prove that the sequence $\{d_m\}$ is "nice", so that DFA's could handle it. This can be done in two steps:

Simpler Languages Cont.

Idea of the proof

- $L_v \cap w^*$ is a submonoid of $w^* \cong \mathbb{N}$, hence, it is of the form $(w^d)^* \setminus E$ for some d and some finite set E.
- If we fix m, then $\{w_1^n \mid w_1^n w_2^m \equiv w_2^m w_1^n\}$ is regular and equal to $(w_1^{d_m})^* \setminus E_m$ for some d_m and finite E_m. We want to prove that the sequence $\{d_m\}$ is "nice", so that DFA's could handle it. This can be done in two steps:
- Step 1: It is easy to show that
  - $\{d_m\}$ is bounded.
  - d_{i+j} divides $lcm(d_i, d_j)$

Simpler Languages Cont.

Idea of the proof

- $L_v \cap w^*$ is a submonoid of $w^* \cong \mathbb{N}$, hence, it is of the form $(w^d)^* \setminus E$ for some d and some finite set E.
- If we fix m, then $\{w_1^n \mid w_1^n w_2^m \equiv w_2^m w_1^n\}$ is regular and equal to $(w_1^{d_m})^* \setminus E_m$ for some d_m and finite E_m. We want to prove that the sequence $\{d_m\}$ is "nice", so that DFA's could handle it. This can be done in two steps:
- Step 1: It is easy to show that
  - $\{d_m\}$ is bounded.
  - d_{i+j} divides $lcm(d_i, d_j)$
- Step 2: These properties imply that $\{d_m \mid m \geq M\}$ is periodic for sufficiently large M.

Simpler Languages Cont.

Idea of the proof

- $L_v \cap w^*$ is a submonoid of $w^* \cong \mathbb{N}$, hence, it is of the form $(w^d)^* \setminus E$ for some d and some finite set E.
- If we fix m, then $\{w_1^n \mid w_1^n w_2^m \equiv w_2^m w_1^n\}$ is regular and equal to $(w_1^{d_m})^* \setminus E_m$ for some d_m and finite E_m. We want to prove that the sequence $\{d_m\}$ is "nice", so that DFA's could handle it. This can be done in two steps:
- Step 1: It is easy to show that
  - $\{d_m\}$ is bounded.
  - d_{i+j} divides $lcm(d_i, d_j)$
- Step 2: These properties imply that $\{d_m \mid m \geq M\}$ is periodic for sufficiently large M.
- This yields the regularity of $\{w_1^n w_2^m \mid w_1^n w_2^m \equiv w_2^m w_1^n\}$.

Combining the Two Assumptions

We can combine these approaches (restricting $\mathbb{I}$ and simplifying the languages):

Combining the Two Assumptions

We can combine these approaches (restricting $\mathbb{I}$ and simplifying the languages):

Corollary

If $\mathbb{I} \subseteq \Sigma_1^* \times \Sigma_2^* \cup \Sigma_2^* \times \Sigma_1^*$, then $L_{a \prec b} \cap a^* b^*$ is regular.

Combining the Two Assumptions

We can combine these approaches (restricting $\mathbb{I}$ and simplifying the languages):

Corollary

If $\mathbb{I} \subseteq \Sigma_1^* \times \Sigma_2^* \cup \Sigma_2^* \times \Sigma_1^*$, then $L_{a \prec b} \cap a^* b^*$ is regular.

Theorem

Suppose $\mathbb{I} \subseteq \Sigma_1^* \times \Sigma_2^* \cup \Sigma_2^* \times \Sigma_1^*$, $|\mathbb{I}| < \infty$ and $a \in \Sigma_1$.

- $L_a := \{u \in \Sigma_2^* \mid au \equiv ua\}$ is regular

Combining the Two Assumptions

We can combine these approaches (restricting $\mathbb{I}$ and simplifying the languages):

Corollary

If $\mathbb{I} \subseteq \Sigma_1^* \times \Sigma_2^* \cup \Sigma_2^* \times \Sigma_1^*$, then $L_{a \prec b} \cap a^* b^*$ is regular.

Theorem

Suppose $\mathbb{I} \subseteq \Sigma_1^* \times \Sigma_2^* \cup \Sigma_2^* \times \Sigma_1^*$, $|\mathbb{I}| < \infty$ and $a \in \Sigma_1$.

- $L_a := \{u \in \Sigma_2^* \mid au \equiv ua\}$ is regular
- $L_{a,v} := \{u \in \Sigma_2^* \mid vau \equiv vua\}$ is regular
- $L_{v,a} := \{u \in \Sigma_2^* \mid uav \equiv auv\}$ is regular

Combining the Two Assumptions

We can combine these approaches (restricting $\mathbb{I}$ and simplifying the languages):

Corollary

If $\mathbb{I} \subseteq \Sigma_1^* \times \Sigma_2^* \cup \Sigma_2^* \times \Sigma_1^*$, then $L_{a \prec b} \cap a^* b^*$ is regular.

Theorem

Suppose $\mathbb{I} \subseteq \Sigma_1^* \times \Sigma_2^* \cup \Sigma_2^* \times \Sigma_1^*$, $|\mathbb{I}| < \infty$ and $a \in \Sigma_1$.

- $L_a := \{u \in \Sigma_2^* \mid au \equiv ua\}$ is regular
- $L_{a,v} := \{u \in \Sigma_2^* \mid vau \equiv vua\}$ is regular
- $L_{v,a} := \{u \in \Sigma_2^* \mid uav \equiv auv\}$ is regular

Proof

The idea is to use the fact that 2NFA's (nondeterministic Turing machines that cannot write) recognize regular languages. We can construct a 2NFA that can simulate the swaps using its head to keep track of the position of the a.

Other directions

Reminder

We want to find a sufficient condition for the regularity of $Lex(\Sigma^*)$

Other directions

Reminder

We want to find a sufficient condition for the regularity of $Lex(\Sigma^*)$

- We noticed that $Lex(\Sigma^*)^c = \Sigma^* \setminus Lex(\Sigma^*)$ has a nice algebraic structure: it is an ideal ($x \in Lex(\Sigma^*)^c \Rightarrow yxz \in Lex(\Sigma^*)^c$ for all $y, z \in \Sigma^*$)

Other directions

Reminder

We want to find a sufficient condition for the regularity of $Lex(\Sigma^*)$

- We noticed that $Lex(\Sigma^*)^c = \Sigma^* \setminus Lex(\Sigma^*)$ has a nice algebraic structure: it is an ideal ($x \in Lex(\Sigma^*)^c \Rightarrow yxz \in Lex(\Sigma^*)^c$ for all $y, z \in \Sigma^*$)
- We can consider the set S of minimal words of $Lex(\Sigma^*)^c$, that is, $S = \{w \mid w \in Lex(\Sigma^*)^c$, no proper subword of w is an element of $Lex(\Sigma^*)^c\}$

Other directions

Reminder

We want to find a sufficient condition for the regularity of $Lex(\Sigma^*)$

- We noticed that $Lex(\Sigma^*)^c = \Sigma^* \setminus Lex(\Sigma^*)$ has a nice algebraic structure: it is an ideal ($x \in Lex(\Sigma^*)^c \Rightarrow yxz \in Lex(\Sigma^*)^c$ for all $y, z \in \Sigma^*$)
- We can consider the set S of minimal words of $Lex(\Sigma^*)^c$, that is, $S = \{w \mid w \in Lex(\Sigma^*)^c$, no proper subword of w is an element of $Lex(\Sigma^*)^c\}$
- It turns out that $Lex(\Sigma^*)$ is regular iff S is regular.

Other directions

Reminder

We want to find a sufficient condition for the regularity of $Lex(\Sigma^*)$

- We noticed that $Lex(\Sigma^*)^c = \Sigma^* \setminus Lex(\Sigma^*)$ has a nice algebraic structure: it is an ideal ($x \in Lex(\Sigma^*)^c \Rightarrow yxz \in Lex(\Sigma^*)^c$ for all $y, z \in \Sigma^*$)
- We can consider the set S of minimal words of $Lex(\Sigma^*)^c$, that is, $S = \{w \mid w \in Lex(\Sigma^*)^c$, no proper subword of w is an element of $Lex(\Sigma^*)^c\}$
- It turns out that $Lex(\Sigma^*)$ is regular iff S is regular.
- It would be great to find a condition on $\mathbb{I}$ that would guarantee that S is regular/finite.

Questions?