NFAs continued, Closure Properties of Regular Languages

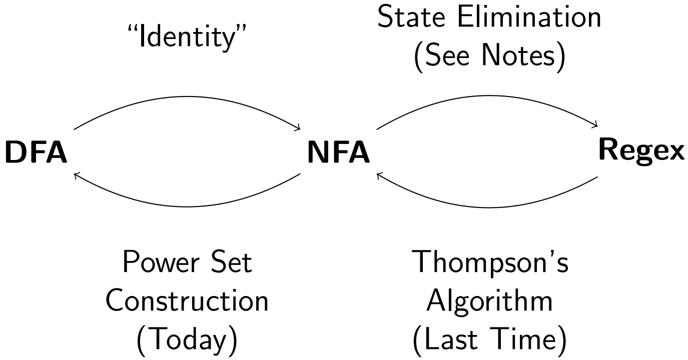
Lecture 5

February 04, 2025

Part I

Equivalence of DFAs, NFAs, and Regular Expressions

Roadmap

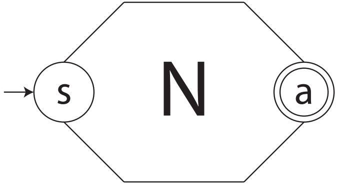


"Identity"
State Elimination
(See Notes)
DFA
NFA
Regex
Power Set
Construction
(Today)
Thompson's
Algorithm
(Last Time)

Last Time on CS374

Theorem

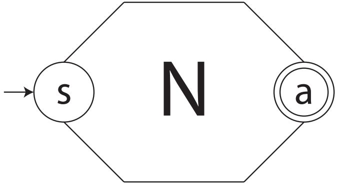
For every regular expression r, *there is a normal form NFA* N *such that* $L(N) = L(r)$.



Last Time on CS374

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.



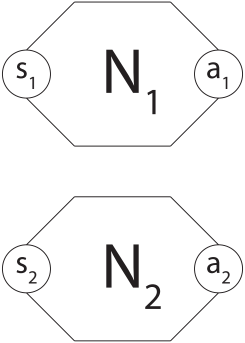
Last time, dealt with base cases ($\emptyset$, ϵ, single character).

Thompson's Algorithm: Union

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

$r = r_1 + r_2$

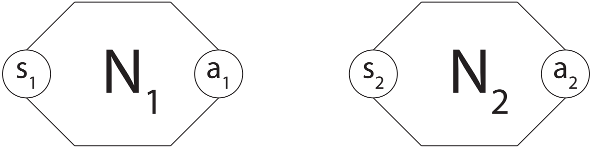


Thompson's Algorithm: Concatenation

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

$r = r_1 r_2$

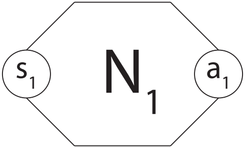


Thompson's Algorithm: Kleene Star

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

$r = r_1^*$

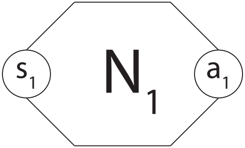


Thompson's Algorithm: Kleene Star (Attempt 2)

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

$r = r_1^*$

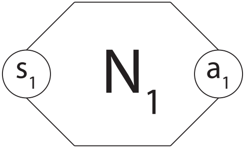


Thompson's Algorithm: Kleene Star (Attempt 3)

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

$r = r_1^*$

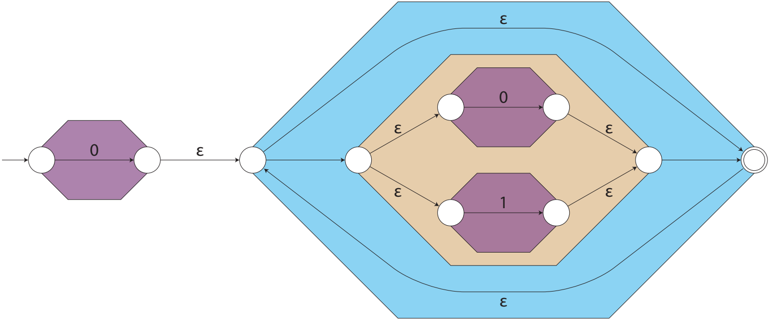


Thompson's Algorithm: Example

$r = 0(0+1)^*$

Thompson's Algorithm: Example

$r = 0(0+1)^*$



Powerset Construction: Intuition

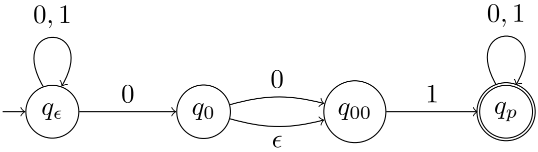
An NFA accepts a string w if it is *possible* to reach an accepting state when reading the string w.

How would we work out by hand if an accepting path is possible?

Powerset Construction: Intuition

An NFA accepts a string w if it is *possible* to reach an accepting state when reading the string w.

How would we work out by hand if an accepting path is possible?

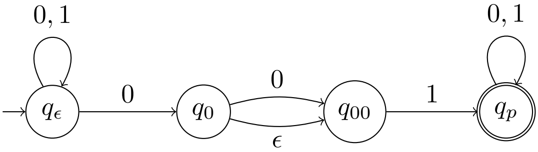


Is there an accepting path for **1001**?

Powerset Construction: Intuition

An NFA accepts a string w if it is *possible* to reach an accepting state when reading the string w.

How would we work out by hand if an accepting path is possible?



Is there an accepting path for **1001**?
What about for **1100**?

Powerset Construction: Intuition

Nothing we did by hand required "guessing" or "multiple possibilities"—this suggests that we can do it with a DFA!

Powerset Construction: Intuition

Nothing we did by hand required "guessing" or "multiple possibilities"—this suggests that we can do it with a DFA!

What would our DFA need to "keep track of"?

Powerset Construction: Intuition

Nothing we did by hand required "guessing" or "multiple possibilities"—this suggests that we can do it with a DFA!

What would our DFA need to "keep track of"?
Possible states we could be at given the string so far.

Powerset Construction: Intuition

Nothing we did by hand required "guessing" or "multiple possibilities"—this suggests that we can do it with a DFA!

What would our DFA need to "keep track of"?
Possible states we could be at given the string so far.

How should our DFA update when it sees a new character?

Powerset Construction: Intuition

Nothing we did by hand required "guessing" or "multiple possibilities"—this suggests that we can do it with a DFA!

What would our DFA need to "keep track of"?
Possible states we could be at given the string so far.

How should our DFA update when it sees a new character?
Consider new set of possible states after reading the character (including possible ϵ transitions).

Powerset Construction: Intuition

Nothing we did by hand required "guessing" or "multiple possibilities"—this suggests that we can do it with a DFA!

What would our DFA need to "keep track of"?
Possible states we could be at given the string so far.

How should our DFA update when it sees a new character?
Consider new set of possible states after reading the character (including possible ϵ transitions).

When should our DFA accept?

Powerset Construction: Intuition

Nothing we did by hand required “guessing” or “multiple possibilities”—this suggests that we can do it with a DFA!

What would our DFA need to “keep track of”?
Possible states we could be at given the string so far.

How should our DFA update when it sees a new character?
Consider new set of possible states after reading the character (including possible ϵ transitions).

When should our DFA accept?
When we could be at an accepting state.

Powerset Construction: Construction

Theorem

For every NFA N there is a DFA M such that $L(M) = L(N)$.

Powerset Construction: Construction

Theorem

For every NFA N there is a DFA M such that $L(M) = L(N)$.

For $N = (Q_N, \Sigma, s_N, \delta_N, A_N)$, create $M = (Q_M, \Sigma, \delta_M, s_M, A_M)$:

- $Q_M =$
- $s_M =$
- $\delta_M(q_M, a) =$
- $A_M =$

Intuition: track "set of possible states".

Powerset Construction: Construction

Theorem

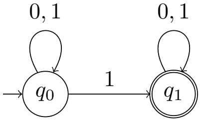
For every NFA N there is a DFA M such that $L(M) = L(N)$.

For $N = (Q_N, \Sigma, s_N, \delta_N, A_N)$, create $M = (Q_M, \Sigma, \delta_M, s_M, A_M)$:

- $Q_M = \mathcal{P}(Q_N)$
- $s_M = \epsilon\text{reach}(s_N)$
- $\delta_M(q_M, a) = \bigcup_{q_N \in q_M} \bigcup_{r_N \in \delta_N(q_N, a)} \epsilon\text{reach}(r_N)$
- $A_M = \{q_M | q_M \cap A_N \neq \emptyset\}$

Intuition: track "set of possible states".

Powerset Construction: Example



Proof of Correctness: Lemma Statement

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta_N^*(s_N, w) = \delta_M^*(s_M, w)$.

Proof of Correctness: Lemma Statement

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta^*_N(s_N, w) = \delta^*_M(s_M, w)$.

Once we prove this lemma, we're done!

Proof of Correctness: Lemma Statement

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta^*_N(s_N, w) = \delta^*_M(s_M, w)$.

Once we prove this lemma, we're done!

- N accepts w if and only if $\delta^*_N(s_N, w) \cap A_N \neq \emptyset$.

Proof of Correctness: Lemma Statement

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta^*_N(s_N, w) = \delta^*_M(s_M, w)$.

Once we prove this lemma, we're done!

- N accepts w if and only if $\delta^*_N(s_N, w) \cap A_N \neq \emptyset$.
- M accepts w if and only if $\delta^*_M(s_M, w) \in A_M$, where $A_M = \{q_M | q_M \cap A_N \neq \emptyset\}$.

Proof of Correctness: Lemma Statement

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta^*_N(s_N, w) = \delta^*_M(s_M, w)$.

Once we prove this lemma, we're done!

- N accepts w if and only if $\delta^*_N(s_N, w) \cap A_N \neq \emptyset$.
- M accepts w if and only if $\delta^*_M(s_M, w) \in A_M$, where $A_M = \{q_M | q_M \cap A_N \neq \emptyset\}$.
- So this lemma says N accepts w if and only if M does!

Proof of Correctness: Base Case

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta_N^*(s_N, w) = \delta_M^*(s_M, w)$.

We will prove this by induction on the length of w.

Proof of Correctness: Base Case

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta_N^*(s_N, w) = \delta_M^*(s_M, w)$.

We will prove this by induction on the length of w.

Base Case: $w = \epsilon$.

Proof of Correctness: Base Case

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta_N^*(s_N, w) = \delta_M^*(s_M, w)$.

We will prove this by induction on the length of w.

Base Case: $w = \epsilon$.

- By definition of δ^* for NFAs, $\delta_N^*(s_N, \epsilon) =$

Proof of Correctness: Base Case

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta^*_N(s_N, w) = \delta^*_M(s_M, w)$.

We will prove this by induction on the length of w.

Base Case: $w = \epsilon$.

- By definition of δ^* for NFAs, $\delta^*_N(s_N, \epsilon) = \epsilon\text{reach}(s_N)$.

Proof of Correctness: Base Case

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta^*_N(s_N, w) = \delta^*_M(s_M, w)$.

We will prove this by induction on the length of w.

Base Case: $w = \epsilon$.

- By definition of δ^* for NFAs, $\delta^*_N(s_N, \epsilon) = \epsilon\text{reach}(s_N)$.
- By definition of δ^* for DFAs, $\delta^*_M(s_M, \epsilon) =$

Proof of Correctness: Base Case

Lemma

Let N be an NFA and M be the DFA obtained by applying the powerset construction to N. For all w, $\delta^*_N(s_N, w) = \delta^*_M(s_M, w)$.

We will prove this by induction on the length of w.

Base Case: $w = \epsilon$.

- By definition of δ^* for NFAs, $\delta^*_N(s_N, \epsilon) = \epsilon\text{reach}(s_N)$.
- By definition of δ^* for DFAs, $\delta^*_M(s_M, \epsilon) = s_M = \epsilon\text{reach}(s_N)$.

Aside: Suffix Definitions

Instead of viewing non-empty strings as ax for $a \in \Sigma$, this proof will be simpler if we view them as xa.

Aside: Suffix Definitions

Instead of viewing non-empty strings as ax for $a \in \Sigma$, this proof will be simpler if we view them as xa.

For DFAs, we define $\delta^*(q, xa) =$

Aside: Suffix Definitions

Instead of viewing non-empty strings as ax for $a \in \Sigma$, this proof will be simpler if we view them as xa.

For DFAs, we define $\delta^*(q, xa) = \delta(\delta^*(q, x), a)$

Aside: Suffix Definitions

Instead of viewing non-empty strings as ax for $a \in \Sigma$, this proof will be simpler if we view them as xa.

For DFAs, we define $\delta^*(q, xa) = \delta(\delta^*(q, x), a)$

For NFAs, we define $\delta^*(q, xa) =$

Aside: Suffix Definitions

Instead of viewing non-empty strings as ax for $a \in \Sigma$, this proof will be simpler if we view them as xa.

For DFAs, we define $\delta^*(q, xa) = \delta(\delta^*(q, x), a)$

For NFAs, we define $\delta^*(q, xa) = \bigcup_{r \in \delta^*(q,x)} \bigcup_{p \in \delta(r,a)} \epsilon\text{reach}(p)$

Aside: Suffix Definitions

Instead of viewing non-empty strings as ax for $a \in \Sigma$, this proof will be simpler if we view them as xa.

For DFAs, we define $\delta^*(q, xa) = \delta(\delta^*(q, x), a)$

For NFAs, we define $\delta^*(q, xa) = \bigcup_{r \in \delta^*(q,x)} \bigcup_{p \in \delta(r,a)} \epsilon\text{reach}(p)$

Note: Base case ($w = \epsilon$) remains the same.

Proof of Correctness: Inductive Case

Inductive Case: $w = xa$.

Proof of Correctness: Inductive Case

Inductive Case: $w = xa$.

- By (suffix) definition of δ^* for NFAs,

$$\delta^*_N(s_N, xa) =$$

Proof of Correctness: Inductive Case

Inductive Case: $w = xa$.

- By (suffix) definition of δ^* for NFAs,

$$\delta_N^*(s_N, xa) = \bigcup_{r_N \in \delta_N^*(s_n, x)} \; \bigcup_{p_N \in \delta_N(r_N, a)} \epsilon\text{reach}(p_N)$$

Proof of Correctness: Inductive Case

Inductive Case: $w = xa$.

- By (suffix) definition of δ^* for NFAs,

$$\delta_N^*(s_N, xa) = \bigcup_{r_N \in \delta_N^*(s_n, x)} \bigcup_{p_N \in \delta_N(r_N, a)} \epsilon\text{reach}(p_N)$$

- By (suffix) definition of δ^* for DFAs,

$$\delta_M^*(s_M, xa) =$$

Proof of Correctness: Inductive Case

Inductive Case: $w = xa$.

- By (suffix) definition of δ^* for NFAs,

$$\delta_N^*(s_N, xa) = \bigcup_{r_N \in \delta_N^*(s_n, x)} \bigcup_{p_N \in \delta_N(r_N, a)} \epsilon\text{reach}(p_N)$$

- By (suffix) definition of δ^* for DFAs,

$$\begin{aligned} \delta_M^*(s_M, xa) &= \delta_M(\delta_M^*(s_m, x), a) \\ &= \bigcup_{r_N \in \delta_M^*(s_M, x)} \bigcup_{p_N \in \delta_N(r_N, a)} \epsilon\text{reach}(p_N) \end{aligned}$$

Proof of Correctness: Inductive Case

Inductive Case: $w = xa$.

- By (suffix) definition of δ^* for NFAs,

$$\delta_N^*(s_N, xa) = \bigcup_{r_N \in \delta_N^*(s_n, x)} \bigcup_{p_N \in \delta_N(r_N, a)} \epsilon\text{reach}(p_N)$$

- By (suffix) definition of δ^* for DFAs,

$$\begin{aligned} \delta_M^*(s_M, xa) &= \delta_M(\delta_M^*(s_m, x), a) \\ &= \bigcup_{r_N \in \delta_M^*(s_M, x)} \bigcup_{p_N \in \delta_N(r_N, a)} \epsilon\text{reach}(p_N) \end{aligned}$$

$|x| < |w|$, so by the IH $\delta_N^*(s_N, x) = \delta_M^*(s_M, x)$!

Incremental Construction

For an NFA with n states, the powerset DFA will have 2^n states—this quickly gets tedious to draw out by hand.

Incremental Construction

For an NFA with n states, the powerset DFA will have 2^n states—this quickly gets tedious to draw out by hand.

- This is sometimes unavoidable: some languages *require* DFAs that are exponentially bigger than their smallest NFA.

Incremental Construction

For an NFA with n states, the powerset DFA will have 2^n states—this quickly gets tedious to draw out by hand.

- This is sometimes unavoidable: some languages *require* DFAs that are exponentially bigger than their smallest NFA.
- However, for many NFAs, applying the powerset construction will result in states that can't be reached from the start state, and so can be ignored.

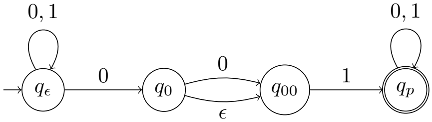
Incremental Construction

For an NFA with n states, the powerset DFA will have 2^n states—this quickly gets tedious to draw out by hand.

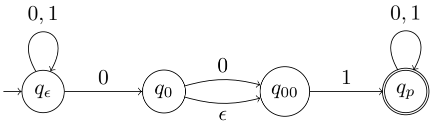
- This is sometimes unavoidable: some languages *require* DFAs that are exponentially bigger than their smallest NFA.
- However, for many NFAs, applying the powerset construction will result in states that can't be reached from the start state, and so can be ignored.

When applying the powerset construction by hand, we can simply add states "as needed" to avoid having too many.

Incremental Construction: Example



Incremental Construction: Tabular



q'	in A'?	$\bigcup_{q \in q'} \delta(q, 0)$	$\delta'(q', 0)$	$\bigcup_{q \in q'} \delta(q, 1)$	$\delta'(q', 1)$

Part II

Closure Properties of Regular Languages

Properties of Regular Languages

Regular languages have three equivalent characterizations:

- Languages defined by regular expressions
- Languages accepted by DFAs
- Languages accepted by NFAs

We can use any of these three to prove properties about regular languages.

- The complement of a regular language is regular: take a DFA and invert the accepting states.
- Boolean combinations of regular languages (union, intersection, difference, ...) are regular: take a DFA for each and apply the product construction
- Concatenation, Kleene star via NFAs or regular expressions
- ...

Generic Closure Property

In general, closure properties will look something like the following:

Theorem

Let L be a regular language and L' be [some modification of L]. Then L' is also a regular language.

Generic Closure Property

In general, closure properties will look something like the following:

Theorem

Let L be a regular language and L' be [some modification of L]. Then L' is also a regular language.

How would we go about proving a statement like this?

Generic Closure Property

In general, closure properties will look something like the following:

Theorem

Let L be a regular language and L' be [some modification of L]. Then L' is also a regular language.

How would we go about proving a statement like this?

- To show that L' is regular, we just have to argue that it has a regular expression/DFA/ NFA.
- We know L is regular, so it has a regular expression/DFA/NFA.
- Try to modify or use the regular expression/DFA/NFA for L in order to build what we want for L'

Example 1: PREFIX

Let L be a language over Σ.

Definition

$\text{PREFIX}(L) = \{w \mid \exists x \in \Sigma^* \text{ s.t. } wx \in L\}$

Example 1: PREFIX

Let L be a language over Σ.

Definition

$\text{PREFIX}(L) = \{w \mid \exists x \in \Sigma^* \text{ s.t. } wx \in L\}$

Theorem

If L is regular then $PREFIX(L)$ is regular.

Example 1: PREFIX

Let L be a language over Σ.

Definition

$\text{PREFIX}(L) = \{w \mid \exists x \in \Sigma^* \text{ s.t. } wx \in L\}$

Theorem

If L is regular then PREFIX(L) is regular.

Intuitive idea:

- Want to know if it's possible to extend our string to get something in L.

Example 1: PREFIX

Let L be a language over Σ.

Definition

$\text{PREFIX}(L) = \{w \mid \exists x \in \Sigma^* \text{ s.t. } wx \in L\}$

Theorem

If L is regular then PREFIX(L) is regular.

Intuitive idea:

- Want to know if it's possible to extend our string to get something in L.
- Phrased in terms of a DFA for L: is it still possible for us to reach an accepting state by reading additional characters?

Example 1: PREFIX

Theorem

If L is regular then PREFIX(L) is regular.

Let $M = (Q, \delta, s, A)$ be a DFA for L. Construct a DFA $M' = (Q', \delta', s', A')$ for PREFIX(L) as:

- $Q' =$
- $\delta'(q', a) =$
- $s' =$
- $A' =$

Example 1: PREFIX

Theorem

If L is regular then PREFIX(L) is regular.

Let $M = (Q, \delta, s, A)$ be a DFA for L. Construct a DFA $M' = (Q', \delta', s', A')$ for PREFIX(L) as:

- $Q' = Q$
- $\delta'(q', a) = \delta(q', a)$
- $s' = s$
- $A' = \{q \in Q \mid \text{in } M, q \text{ can reach a state in } A\}$

Example 2: SUFFIX

Let L be a language over Σ.

Definition

$\text{SUFFIX}(L) = \{w \mid \exists x \in \Sigma^* \text{ s.t. } xw \in L\}$

Example 2: SUFFIX

Let L be a language over Σ.

Definition

$\text{SUFFIX}(L) = \{w \mid \exists x \in \Sigma^* \text{ s.t. } xw \in L\}$

Theorem

If L is regular then $\text{SUFFIX}(L)$ is regular.

Example 2: SUFFIX

Let L be a language over Σ.

Definition

$\text{SUFFIX}(L) = \{w \mid \exists x \in \Sigma^* \text{ s.t. } xw \in L\}$

Theorem

If L is regular then $\text{SUFFIX}(L)$ is regular.

Intuitive idea:

- Want to know if it's possible to have read something *before* our string to get something in L.
- Phrased in terms of a DFA for L: could we run for some time before reading w and end up in an accepting state?
- How do we know what state to go to before starting to read w? Guess!

Example 2: SUFFIX

Theorem

If L is regular then SUFFIX(L) is regular.

Let $M = (Q, \delta, s, A)$ be a DFA for L. Construct an **NFA** $N = (Q', \delta', s', A')$ for SUFFIX(L) as:

- $Q' =$
- $\delta'(q', a) =$
- $s' =$
- $A' =$

Example 2: SUFFIX

Theorem

If L is regular then SUFFIX(L) is regular.

Let $M = (Q, \delta, s, A)$ be a DFA for L. Construct an NFA $N = (Q', \delta', s', A')$ for SUFFIX(L) as:

- $Q' = Q \cup \{s_{new}\}$
- $\delta'(q', a) =$
$$\begin{cases} \{\delta(q', a)\} & q' \in Q, a \in \Sigma \\ \{q \in Q \mid s \text{ can reach } q \text{ in } M\} & q' = s_{new}, a = \epsilon \\ \emptyset & \text{otherwise} \end{cases}$$
- $s' = s_{new}$
- $A' = A$

Example 3: SUFFIX, Again

Theorem

If L is regular then SUFFIX(L) is regular.

Alternative proof: inductively show that we can convert a regular expression r for L to a regular expression r' for SUFFIX(L).

Example 3: SUFFIX, Again

Theorem

If L is regular then SUFFIX(L) is regular.

Alternative proof: inductively show that we can convert a regular expression r for L to a regular expression r' for SUFFIX(L).

Base Cases	Inductive Cases
$r = \emptyset$ $r' =$	$r = r_1 + r_2$ $r' =$
$r = \epsilon$ $r' =$	$r = r_1 r_2$ $r' =$
$r = a$ $r' =$	$r = (r_1)^*$ $r' =$

Example 3: SUFFIX, Again

Theorem

If L is regular then SUFFIX(L) is regular.

Alternative proof: inductively show that we can convert a regular expression r for L to a regular expression r' for SUFFIX(L).

Base Cases	Inductive Cases
$r = \emptyset$ $r' = \emptyset$	$r = r_1 + r_2$ $r' = r'_1 + r'_2$
$r = \epsilon$ $r' = \epsilon$	$r = r_1 r_2$ $r' = r'_2 + r'_1 r_2$
$r = a$ $r' = \epsilon + a$	$r = (r_1)^*$ $r' = r'_1 (r_1)^*$

Exercises

Given language L, let $\text{MID}(L) = \{w \mid \exists x, y \in \Sigma^*, xwy \in L\}$.

Theorem

If L is regular then $MID(L)$ is regular.

Given L let $L^R = \{w^R \mid w \in L\}$ be the reverses of all strings in L

Theorem

If L is regular then L^R is regular.