Equivalence between FA and Regular Expression, Nonregular languages, and the Pumping Lemma

204213 Theory of Computation

Jittat Fakcharoenphol

Kasetsart University

November 22, 2008

Outline

Short review: NFA and DFA

- For a **deterministic** finite automaton, given its current state and an input symbol from the alphabet, the next state is determined.

Short review: NFA and DFA

- For a **deterministic** finite automaton, given its current state and an input symbol from the alphabet, the next state is determined.
- For a **nondeterministic** finite automaton, given its current state and an input symbol from the alphabet, there can be many possible states (or none).

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.
- Let $Q' = \mathcal{P}(Q)$.

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.
- Let $Q' = \mathcal{P}(Q)$.
- Define δ' so that M correctly simulates many copies of N.

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.
- Let $Q' = \mathcal{P}(Q)$.
- Define δ' so that M correctly simulates many copies of N.
- Carefully handle ε.

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q'_0, F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.
- Let $Q' = \mathcal{P}(Q)$.
- Define δ' so that M correctly simulates many copies of N.
- Carefully handle ε.
- M accepts any state $R \in Q'$ such that $R \cap F \neq \emptyset$.

Definition [regular expression]

- An inductive definition of regular expressions.

Definition [regular expression]

- An inductive definition of regular expressions.
- R is a **regular expression** if R is
  1. a for some $a \in \Sigma$,
  2. ε,
  3. $\emptyset$,
  4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions,
  5. $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions, and
  6. (R_1^*) where R_1 is a regular expression.

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular.

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular. **Proved last time**

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular. **Proved last time by considering how regular expressions can be constructed.**

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular. **Proved last time by considering how regular expressions can be constructed.**
- **Today:** If a language is regular, then it can be described by a regular expression.

The second part

Theorem 2

Any regular language can be described with a regular expression.

The second part

Theorem 2

Any regular language can be described with a regular expression.

How are we going to prove this?

The second part

Theorem 2

Any regular language can be described with a regular expression.

How are we going to prove this?
Think:

The second part

Theorem 2

Any regular language can be described with a regular expression.

How are we going to prove this?
Think:

- What do we know?

The second part

Theorem 2

Any regular language can be described with a regular expression.

How are we going to prove this?
Think:

- What do we know?
    - A is a regular language.

The second part

Theorem 2

Any regular language can be described with a regular expression.

How are we going to prove this?
Think:

- What do we know?
  - A is a regular language.
- What does that mean?

The second part

Theorem 2

Any regular language can be described with a regular expression.

How are we going to prove this?
Think:

- What do we know?
  - A is a regular language.
- What does that mean? ummm...

The second part

Theorem 2

Any regular language can be described with a regular expression.

How are we going to prove this?
Think:

- What do we know?
  - A is a regular language.
- What does that mean? ummm... (hint: use definition)

The second part

Theorem 2

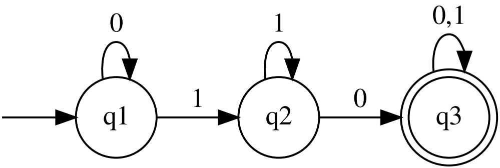
Any regular language can be described with a regular expression.

How are we going to prove this?
Think:

- What do we know?
  - A is a regular language.
- What does that mean? ummm... (hint: use definition)
  - There is a DFA M that recognizes A.

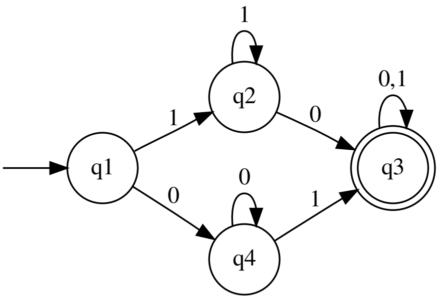
Practice: M_1

Okay, let's do some practice.



What is a regular expression describing the language recognized by M_1.

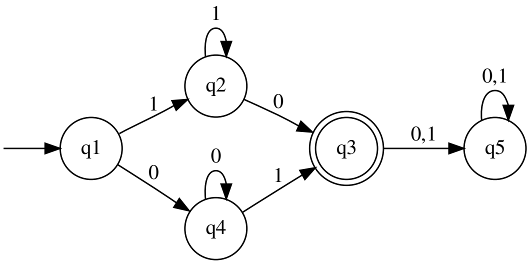
Practice: M_2



What is a regular expression describing the language recognized by M_2.

While you're trying to figure out the regular expression, try to think about a "mechanical" method for constructing it from a DFA.

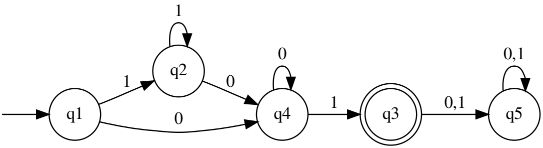
Practice: M_3



What is a regular expression describing the language recognized by M_3.

While you're trying to figure out the regular expression, try to think about a "mechanical" method for constructing it from a DFA.

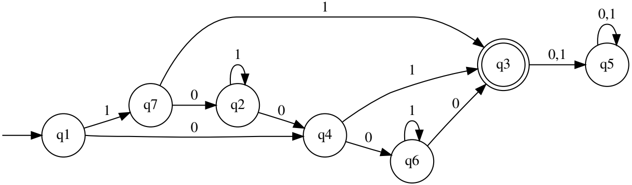
Practice: M_4



What is a regular expression describing the language recognized by M_4.

While you're trying to figure out the regular expression, try to think about a "mechanical" method for constructing it from a DFA.

Practice: M_5



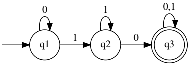
What is a regular expression describing the language recognized by M_5.

While you're trying to figure out the regular expression, try to think about a "mechanical" method for constructing it from a DFA.

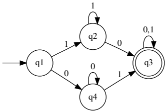
Easy rules?

Just like a way to calculating resistances:

- Series



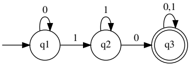
- Parallel



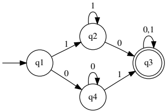
Easy rules?

Just like a way to calculating resistances:

- Series



- Parallel



But things could mess up really quickly. (Think about M_5.)

"Baby step"

- Instead of trying to convert the whole DFA to a regular expression in one step, we will try to make some progress.

"Baby step"

- Instead of trying to convert the whole DFA to a regular expression in one step, we will try to make some progress.
- If we can always make some progress, we surely get to the finish line for sure.

“Baby step”

- Instead of trying to convert the whole DFA to a regular expression in one step, we will try to make some progress.
- If we can always make some progress, we surely get to the finish line for sure. How?

"Baby step"

- Instead of trying to convert the whole DFA to a regular expression in one step, we will try to make some progress.
- If we can always make some progress, we surely get to the finish line for sure. How? **Think about induction.**
- But what kind of progress?

"Baby step"

- Instead of trying to convert the whole DFA to a regular expression in one step, we will try to make some progress.
- If we can always make some progress, we surely get to the finish line for sure. How? **Think about induction.**
- But what kind of progress?
  - It maybe better to start by asking what kind of finishing line that we want.

Goal

Simplest FA for Regular Expression construction:

Goal

Simplest FA for Regular Expression construction:

- DFA (or even NFA) with two states: one start state and one accept state.

Goal

Simplest FA for Regular Expression construction:

- DFA (or even NFA) with two states: one start state and one accept state.

But how could we get there?

Goal

Simplest FA for Regular Expression construction:

- DFA (or even NFA) with two states: one start state and one accept state.

But how could we get there?
After thinking a bit

Goal

Simplest FA for Regular Expression construction:

- DFA (or even NFA) with two states: one start state and one accept state.

But how could we get there?
After thinking a bit it is quite straight-forward.

Goal

Simplest FA for Regular Expression construction:

- DFA (or even NFA) with two states: one start state and one accept state.

But how could we get there?
After thinking a bit it is quite straight-forward.

- Try to reduce the number of states.

Goal

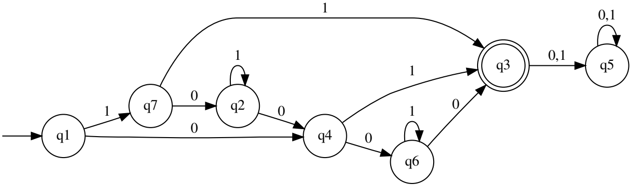
Simplest FA for Regular Expression construction:

- DFA (or even NFA) with two states: one start state and one accept state.

But how could we get there?
After thinking a bit it is quite straight-forward.

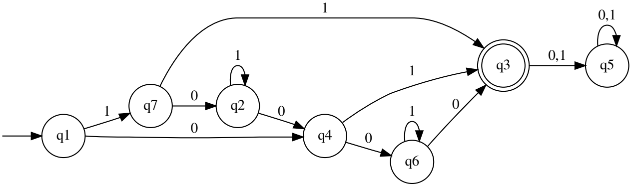
- Try to reduce the number of states.
- Each step decreases the number of states by one.

Let's try with M_5



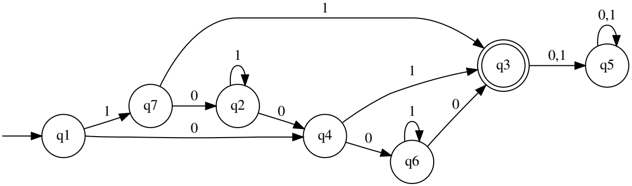
- Try to remove q_7.

Let's try with M_5



- Try to remove q_7.
- Start over. Try to remove q_4.

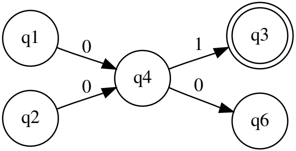
Let's try with M_5



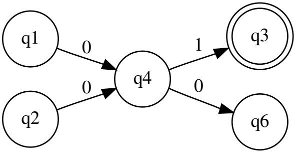
- Try to remove q_7.
- Start over. Try to remove q_4.

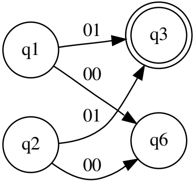
Now you get an idea.

Removing q_4

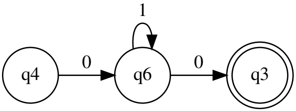


Removing q_4





Removing state q_6



Removing state q_6

Note: Power-up required

- To accommodate the state reduction procedure, we have to allow transition edges with regular expressions.

Note: Power-up required

- To accommodate the state reduction procedure, we have to allow transition edges with regular expressions.
- This is fine: we shall define the generalized nondeterministic finite automata.

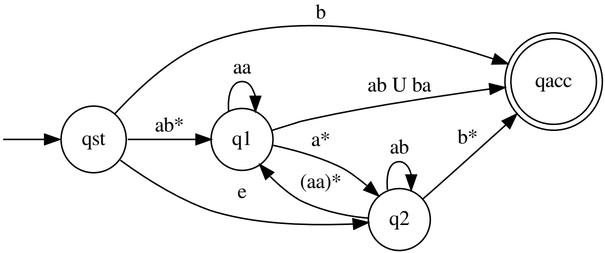
Generalized Nondeterministic Finite Automata

- A **generalized nondeterministic finite automata** are nondeterministic finite automata where we allow regular expressions as labels on transition arrows.

Generalized Nondeterministic Finite Automata

- A **generalized nondeterministic finite automata** are nondeterministic finite automata where we allow regular expressions as labels on transition arrows.
- A GNFA can move to a new state only if it can read a block of input symbols that is described by the regular expression on the arrow.

An example of a GNFA



Special form

We also require that the GNFA that we'll use satisfies the following conditions.

- The start state has transition arrows to every other states, but no arrows coming in from any other state.

Special form

We also require that the GNFA that we'll use satisfies the following conditions.

- The start state has transition arrows to every other states, but no arrows coming in from any other state.
- There is a single accept state, and it has arrows coming from every other states but no arrows going to any other state. Further more the accept state is not the same as the start state.

Special form

We also require that the GNFA that we'll use satisfies the following conditions.

- The start state has transition arrows to every other states, but no arrows coming in from any other state.
- There is a single accept state, and it has arrows coming from every other states but no arrows going to any other state. Further more the accept state is not the same as the start state.
- Except fot the start state and accept state, one arrow goes from every state to every other state and also each state to itself.

Special form

We also require that the GNFA that we'll use satisfies the following conditions.

- The start state has transition arrows to every other states, but no arrows coming in from any other state.
- There is a single accept state, and it has arrows coming from every other states but no arrows going to any other state. Further more the accept state is not the same as the start state.
- Except fot the start state and accept state, one arrow goes from every state to every other state and also each state to itself.

This form of GNFA will be easy to be converted into a regular expression.

Our approach

- Since language A is regular, we have a DFA M recognizing A.

Our approach

- Since language A is regular, we have a DFA M recognizing A.
- From M, we convert it to an equivalent GNFA G.

Our approach

- Since language A is regular, we have a DFA M recognizing A.
- From M, we convert it to an equivalent GNFA G. **(To be discussed in part 1)**

Our approach

- Since language A is regular, we have a DFA M recognizing A.
- From M, we convert it to an equivalent GNFA G. **(To be discussed in part 1)**
- If G has more than 2 states, find an equivalent GNFA G' with fewer states.

Our approach

- Since language A is regular, we have a DFA M recognizing A.
- From M, we convert it to an equivalent GNFA G. **(To be discussed in part 1)**
- If G has more than 2 states, find an equivalent GNFA G' with fewer states. **(To be proved in part 2)**

Part 1: DFA $\Rightarrow$ GNFA

- Given a DFA M, we'll construct an equivalent GNFA G.

Part 1: DFA $\Rightarrow$ GNFA

- Given a DFA M, we'll construct an equivalent GNFA G. (Recall the conditions of GNFAs.)
- Basically, we have to fix the the start state and accept state.

Part 1: DFA $\Rightarrow$ GNFA

- Given a DFA M, we'll construct an equivalent GNFA G. (Recall the conditions of GNFAs.)
- Basically, we have to fix the the start state and accept state.
- Magic helpers:
  - Arrows with ϵ
  - Arrows with $\emptyset$

DFA $\Rightarrow$ GNFA: Construction

Given $M = (Q, \Sigma, \delta, q_0, F)$:

- Add new start state q_{start}, add an arrow from q_{start} to q_0.

DFA $\Rightarrow$ GNFA: Construction

Given $M = (Q, \Sigma, \delta, q_0, F)$:

- Add new start state q_{start}, add an arrow from q_{start} to q_0.
- Add new accept state q_{accept}, add an arrow from every state $q \in F$ to q_{accept}.

DFA $\Rightarrow$ GNFA: Construction

Given $M = (Q, \Sigma, \delta, q_0, F)$:

- Add new start state q_{start}, add an arrow from q_{start} to q_0.
- Add new accept state q_{accept}, add an arrow from every state $q \in F$ to q_{accept}.
- Add all other arrows labelled with $\emptyset$.

Part 2: GNFA $\Rightarrow$ Regular expression

- If a GNFA G has 2 states, the conversion is straight-forward.

Part 2: GNFA $\Rightarrow$ Regular expression

- If a GNFA G has 2 states, the conversion is straight-forward.
- If a GNFA G has more than 2 states:
  - Pick one state $q_{rip} \notin \{q_{start}, q_{accept}\}$.

Part 2: GNFA $\Rightarrow$ Regular expression

- If a GNFA G has 2 states, the conversion is straight-forward.
- If a GNFA G has more than 2 states:
  - Pick one state $q_{rip} \notin \{q_{start}, q_{accept}\}$. (There should be one, why?)

Part 2: GNFA $\Rightarrow$ Regular expression

- If a GNFA G has 2 states, the conversion is straight-forward.
- If a GNFA G has more than 2 states:
    - Pick one state $q_{rip} \notin \{q_{start}, q_{accept}\}$. (There should be one, why?)
    - Build an equivalent G' by removing q_{rip}
    - Repeat.

Definition [GNFA]

A **generalized nondeterministic finite automaton** is a 5-tuple, $(Q, \Sigma, \delta, q_{start}, q_{accept})$, where

1. Q is the finite set of states,
2. Σ is the input alphabet,
3. $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow \mathcal{R}$ is the transition function,
4. q_{start} is the start state, and
5. q_{accept} is the accept state.

A note on δ for GNFAs

- Previously, the transition function of DFAs or NFAs are functions that take the current states and the input symbol and output the next state or the set of possible next states.

A note on δ for GNFAs

- Previously, the transition function of DFAs or NFAs are functions that take the current states and the input symbol and output the next state or the set of possible next states.
- The focus there is to answer the question "what's next?"

A note on δ for GNFAs

- Previously, the transition function of DFAs or NFAs are functions that take the current states and the input symbol and output the next state or the set of possible next states.
- The focus there is to answer the question "what's next?"
- For GNFA, we do not care that much how it actually works, but we want to use it to do the conversion.

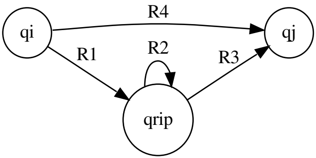
A note on δ for GNFAs

- Previously, the transition function of DFAs or NFAs are functions that take the current states and the input symbol and output the next state or the set of possible next states.
- The focus there is to answer the question "what's next?"
- For GNFA, we do not care that much how it actually works, but we want to use it to do the conversion.
- We focus more on "what's the regular expression on this arrow?".

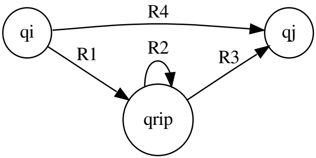
A note on δ for GNFAs

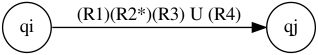
- Previously, the transition function of DFAs or NFAs are functions that take the current states and the input symbol and output the next state or the set of possible next states.
- The focus there is to answer the question "what's next?"
- For GNFA, we do not care that much how it actually works, but we want to use it to do the conversion.
- We focus more on "what's the regular expression on this arrow?". That's how the definition of the transition function δ in this case is defined quite differently.

Removing q_{rip}



Removing q_{rip}





Q' and δ'

From $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$, we'll construct an equivalent $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$.

Q' and δ'

From $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$, we'll construct an equivalent $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$.

- Fix q_{rip}.

Q' and δ'

From $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$, we'll construct an equivalent $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$.

- Fix q_{rip}. Let $Q' = Q - \{q_{rip}\}$.

Q' and δ'

From $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$, we'll construct an equivalent $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$.

- Fix q_{rip}. Let $Q' = Q - \{q_{rip}\}$.
- for any $q_i \in Q' - \{q_{accept}\}$ and $q_j \in Q' - \{q_{start}\}$,

Q' and δ'

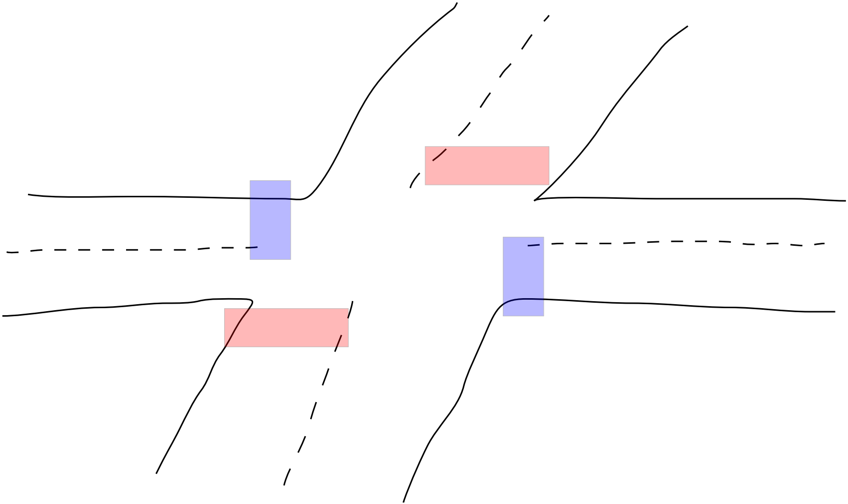
From $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$, we'll construct an equivalent $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$.

- Fix q_{rip}. Let $Q' = Q - \{q_{rip}\}$.
- for any $q_i \in Q' - \{q_{accept}\}$ and $q_j \in Q' - \{q_{start}\}$, let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

where $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

Traffic light control



Extracting string constants

```
#include <stdio.h>

main()
{
  int a, b;
  scanf("%d %d",&a,&b);
  printf("Hello!  \"welcome\" %d\n",a+b);
}
```

What is the limit of DFA/NFA/RegEx?

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?
  - Yes. We'll see one now:

$$B = \{0^n 1^n | n \geq 0\}.$$

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?
    - Yes. We'll see one now:

$$B = \{0^n 1^n | n \geq 0\}.$$

    - Really? I don't believe it until I (or you) have proved it.

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?
  - Yes. We'll see one now:

$$B = \{0^n 1^n | n \geq 0\}.$$

  - Really? I don't believe it until I (or you) have proved it.
  - Some intuition: any DFA M recognizing B seems to have to remember the number of 0, but since M has finite state it will remember incorrectly when n is very large.

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?
  - Yes. We'll see one now:

$$B = \{0^n 1^n | n \geq 0\}.$$

  - Really? I don't believe it until I (or you) have proved it.
  - Some intuition: any DFA M recognizing B seems to have to remember the number of 0, but since M has finite state it will remember incorrectly when n is very large.
  - Again, that's **not** a proof.

Two other languages

How about these languages?

Two other languages

How about these languages?

- $C = \{w|\ w \text{ has an equal number of 0's and 1's }\}$.

Two other languages

How about these languages?

- $C = \{w \mid w$ has an equal number of 0's and 1's $\}$.
- $D = \{w \mid w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Two other languages

How about these languages?

- $C = \{w|\ w$ has an equal number of 0's and 1's $\}$.
- $D = \{w|\ w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Interestingly, both seems to require remembering the lengths which can be really long.

Two other languages

How about these languages?

- $C = \{w|\ w$ has an equal number of 0's and 1's $\}$.
- $D = \{w|\ w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Interestingly, both seems to require remembering the lengths which can be really long.

Solution:

Two other languages

How about these languages?

- $C = \{w|\ w$ has an equal number of 0's and 1's $\}$.
- $D = \{w|\ w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Interestingly, both seems to require remembering the lengths which can be really long.
Solution: C is not regular,

Two other languages

How about these languages?

- $C = \{w|\ w$ has an equal number of 0's and 1's $\}$.
- $D = \{w|\ w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Interestingly, both seems to require remembering the lengths which can be really long.
Solution: C is not regular, but D is!

Main tool: the pumping lemma

- The pumping lemma:

  For any regular language, there is a string length, called the pumping length*, such that*

Main tool: the pumping lemma

- The pumping lemma:

  For any regular language, there is a string length, called the pumping length*, such that for any string as long as the pumping length can be "pumped".*

Main tool: the pumping lemma

- The pumping lemma:

  For any regular language, there is a string length, called the pumping length, such that for any string as long as the pumping length can be "pumped".

- "pumped" — the string contains a section that can be repeated any number of times while the resulting string remains in the language.

Theorem [Pumping Lemma]

Theorem 3 (Pumping lemma)

If A is a regular language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s maybe divided into three pieces $s = xyz$, satisfying the following conditions:

1. *for each $i \geq 0$, $xy^iz \in A$,*
2. *$|y| > 0$, and*
3. *$|xy| \leq p$.*

Proving that B is not regular (1)

- Let B be the language $\{0^n1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.

Proving that B is not regular (1)

- Let B be the language $\{0^n1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular.

Proving that B is not regular (1)

- Let B be the language $\{0^n1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular. From the pumping lemma, we know that there exists a pumping length p.

Proving that B is not regular (1)

- Let B be the language $\{0^n 1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular. From the pumping lemma, we know that there exists a pumping length p.
- Let $s = 0^p 1^p$. We know that $s \in B$,

Proving that B is not regular (1)

- Let B be the language $\{0^n1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular. From the pumping lemma, we know that there exists a pumping length p.
- Let $s = 0^p1^p$. We know that $s \in B$, and $|s| \geq p$.

Proving that B is not regular (1)

- Let B be the language $\{0^n 1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular. From the pumping lemma, we know that there exists a pumping length p.
- Let $s = 0^p 1^p$. We know that $s \in B$, and $|s| \geq p$.
- Now applying the pumping lemma, we have that s can be split into $s = xyz$, and for any i, $xy^i z$ is also in B.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y?

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.
- **Case 2:** $y = 1^k$ for some $k > 0$. Again for the same reason, this case is impossible.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.
- **Case 2:** $y = 1^k$ for some $k > 0$. Again for the same reason, this case is impossible.
- **Case 3:** $y = 0^j1^k$ for some $j > 0$ and $k > 0$. Note that in this case we'll have that $xy^2z = x0^j1^k0^j1^kz \in B$, which is, again, not possible.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.
- **Case 2:** $y = 1^k$ for some $k > 0$. Again for the same reason, this case is impossible.
- **Case 3:** $y = 0^j1^k$ for some $j > 0$ and $k > 0$. Note that in this case we'll have that $xy^2z = x0^j1^k0^j1^kz \in B$, which is, again, not possible.
- For any cases, we have reached the contradiction.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.
- **Case 2:** $y = 1^k$ for some $k > 0$. Again for the same reason, this case is impossible.
- **Case 3:** $y = 0^j1^k$ for some $j > 0$ and $k > 0$. Note that in this case we'll have that $xy^2z = x0^j1^k0^j1^kz \in B$, which is, again, not possible.
- For any cases, we have reached the contradiction.
- Thus, B is not regular.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.
- Take the pumping length p.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.
- Take the pumping length p.
- Find some string s, of length at least p, such that after pumped s will not be in the language.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.
- Take the pumping length p.
- Find some string s, of length at least p, such that after pumped s will not be in the language.
- Get the desired contradiction.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.
- Take the pumping length p.
- Find some string s, of length at least p, such that after pumped s will not be in the language.
- Get the desired contradiction.
- **Happy!**

Practice: Language C

- Let $C = \{w | \ w$ has an equal number of 0's and 1's $\}$

Practice: Language C

- Let $C = \{w|\ w$ has an equal number of 0's and 1's $\}$
- **Hint:** don't forget condition 3.

Practice: Language F

- Let $F = \{ww | w \in \{0, 1\}^*\}$.

Practice: Language F

- Let $F = \{ww | w \in \{0, 1\}^*\}$.
- **Hint:** choose the right $s \in F$.

Proving the pumping lemma: idea (1)

- Since A is regular, we know that there exists $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes A.

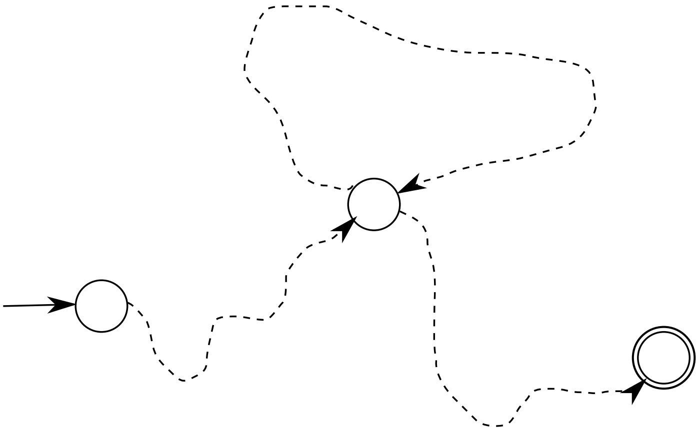
Proving the pumping lemma: idea (1)

- Since A is regular, we know that there exists $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes A.
- Think about what happens when M **accepts** a really long string.

Proving the pumping lemma: idea (1)

- Since A is regular, we know that there exists $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes A.
- Think about what happens when M **accepts** a really long string.
- Since Q is finite, when taking a really long string, you'll see some state on the sequence of states from q_0 to some accept state (remember?) repeats.

Proving the pumping lemma: idea (2)



Proving the pumping lemma: steps

- Let $p = |Q|$.

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p + 1$ states visited by this path (including the start state).

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p+1$ states visited by this path (including the start state). (Why do we have visited at least $p+1$ state?)

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p + 1$ states visited by this path (including the start state). (Why do we have visited at least $p + 1$ state?)
- Since M has p states, but we visited $p + 1$ states,

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p + 1$ states visited by this path (including the start state). (Why do we have visited at least $p+1$ state?)
- Since M has p states, but we visited $p + 1$ states, we should have visited some state twice.

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p + 1$ states visited by this path (including the start state). (Why do we have visited at least $p + 1$ state?)
- Since M has p states, but we visited $p + 1$ states, we should have visited some state twice.
- (Now you try to fill the rest.)