BILL START RECORDING

HW05 Solutions

Problem 1a, 1b

Problem 1a, 1b

a) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cap L(M_2)$ of size **FILL IN**

Problem 1a, 1b

a) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cap L(M_2)$ of size **FILL IN**

FILL IN is $n_1 n_2$.

Problem 1a, 1b

a) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cap L(M_2)$ of size **FILL IN**

FILL IN is $n_1 n_2$.
Use Cross Product Construction.

Problem 1a, 1b

a) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cap L(M_2)$ of size **FILL IN**

FILL IN is $n_1 n_2$.

Use Cross Product Construction.

YOU SHOULD DO: This also works if you begin with two NFA's.

Problem 1a, 1b

a) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cap L(M_2)$ of size **FILL IN**

FILL IN is $n_1 n_2$.
Use Cross Product Construction.

YOU SHOULD DO: This also works if you begin with two NFA's.

b) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns an NFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**

Problem 1a, 1b

a) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cap L(M_2)$ of size **FILL IN**

FILL IN is $n_1 n_2$.
Use Cross Product Construction.

YOU SHOULD DO: This also works if you begin with two NFA's.

b) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns an NFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**

FILL IN is $n_1 + n_2$.

Problem 1a, 1b

a) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cap L(M_2)$ of size **FILL IN**

FILL IN is $n_1 n_2$.
Use Cross Product Construction.

YOU SHOULD DO: This also works if you begin with two NFA's.

b) There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns an NFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**

FILL IN is $n_1 + n_2$.
Just have a transition from the final states of M_1 to the start state of M_2, and the final states are just the final states of M_2.

Problem 1c, 1d

Problem 1c, 1d

c) There is an algorithm that will, given a regex α of length n, returns an NFA for $L(\alpha)$ of size **FILL IN**

Problem 1c, 1d

c) There is an algorithm that will, given a regex α of length n, returns an NFA for $L(\alpha)$ of size **FILL IN**

FILL IN is $\leq 2n$ ($O(n)$ is fine).

Problem 1c, 1d

c) There is an algorithm that will, given a regex α of length n, returns an NFA for $L(\alpha)$ of size **FILL IN**

FILL IN is $\leq 2n$ ($O(n)$ is fine).
The construction is on the slides.
Its by induction on the length of the regex.

Problem 1c, 1d

c) There is an algorithm that will, given a regex α of length n, returns an NFA for $L(\alpha)$ of size **FILL IN**

FILL IN is $\leq 2n$ ($O(n)$ is fine).
The construction is on the slides.
Its by induction on the length of the regex.

YOU SHOULD DO: Redo the proof keeping track of the length of the rules. You will PROVE that you end up with an NFA of size $2n$.

Problem 1c, 1d

c) There is an algorithm that will, given a regex α of length n, returns an NFA for $L(\alpha)$ of size **FILL IN**

FILL IN is $\leq 2n$ ($O(n)$ is fine).
The construction is on the slides.
Its by induction on the length of the regex.

YOU SHOULD DO: Redo the proof keeping track of the length of the rules. You will PROVE that you end up with an NFA of size $2n$.

d) There is an algorithm that will, given a DFA M of size n, returns a regex for $L(M)$ of size **FILL IN**

Problem 1c, 1d

c) There is an algorithm that will, given a regex α of length n, returns an NFA for $L(\alpha)$ of size **FILL IN**

FILL IN is $\leq 2n$ ($O(n)$ is fine).
The construction is on the slides.
Its by induction on the length of the regex.

YOU SHOULD DO: Redo the proof keeping track of the length of the rules. You will PROVE that you end up with an NFA of size $2n$.

d) There is an algorithm that will, given a DFA M of size n, returns a regex for $L(M)$ of size **FILL IN**

FILL IN is $2^{O(n)}$.

Problem 1c, 1d

c) There is an algorithm that will, given a regex α of length n, returns an NFA for $L(\alpha)$ of size **FILL IN**

FILL IN is $\leq 2n$ ($O(n)$ is fine).
The construction is on the slides.
Its by induction on the length of the regex.

YOU SHOULD DO: Redo the proof keeping track of the length of the rules. You will PROVE that you end up with an NFA of size $2n$.

d) There is an algorithm that will, given a DFA M of size n, returns a regex for $L(M)$ of size **FILL IN**

FILL IN is $2^{O(n)}$.

This is the $R(i,j,k)$ construction.

Problem 1c, 1d

c) There is an algorithm that will, given a regex α of length n, returns an NFA for $L(\alpha)$ of size **FILL IN**

FILL IN is $\leq 2n$ ($O(n)$ is fine).
The construction is on the slides.
Its by induction on the length of the regex.

YOU SHOULD DO: Redo the proof keeping track of the length of the rules. You will PROVE that you end up with an NFA of size $2n$.

d) There is an algorithm that will, given a DFA M of size n, returns a regex for $L(M)$ of size **FILL IN**

FILL IN is $2^{O(n)}$.

This is the $R(i,j,k)$ construction.

YOU SHOULD DO: This construction also works if you start with an NFA.

Problem 2

PROVE the following statements by giving an algorithm (your algorithm may use the algorithms in problem 1 as subroutines) and fill in where it says **FILLIN**. No proof of the **FILL IN** is needed.

Problem 2

PROVE the following statements by giving an algorithm (your algorithm may use the algorithms in problem 1 as subroutines) and fill in where it says **FILLIN**. No proof of the **FILL IN** is needed.

Go to the next slide.

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

1) Input two DFA's M_1, M_2 of sizes n_1, n_2.

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

1) Input two DFA's M_1, M_2 of sizes n_1, n_2.

2) Add-an-e-transition construction: NFA M such that

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

1) Input two DFA's M_1, M_2 of sizes n_1, n_2.

2) Add-an-e-transition construction: NFA M such that $L(M) = L(M_1)L(M_2)$.

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

1) Input two DFA's M_1, M_2 of sizes n_1, n_2.

2) Add-an-e-transition construction: NFA M such that
$L(M) = L(M_1)L(M_2)$.
M is of size $n_1 + n_2$.

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

1) Input two DFA's M_1, M_2 of sizes n_1, n_2.

2) Add-an-e-transition construction: NFA M such that $L(M) = L(M_1)L(M_2)$.
M is of size $n_1 + n_2$.

3) Powerset construction on M: DFA D

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

1) Input two DFA's M_1, M_2 of sizes n_1, n_2.

2) Add-an-e-transition construction: NFA M such that
$L(M) = L(M_1)L(M_2)$.
M is of size $n_1 + n_2$.

3) Powerset construction on M: DFA D
$L(D) = L(M_1)L(M_2)$.

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

1) Input two DFA's M_1, M_2 of sizes n_1, n_2.

2) Add-an-e-transition construction: NFA M such that
$L(M) = L(M_1)L(M_2)$.
M is of size $n_1 + n_2$.

3) Powerset construction on M: DFA D
$L(D) = L(M_1)L(M_2)$.
D has size $2^{n_1+n_2}$.

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

1) Input two DFA's M_1, M_2 of sizes n_1, n_2.

2) Add-an-e-transition construction: NFA M such that
$L(M) = L(M_1)L(M_2)$.
M is of size $n_1 + n_2$.

3) Powerset construction on M: DFA D
$L(D) = L(M_1)L(M_2)$.
D has size $2^{n_1+n_2}$.

FILL IN is $2^{n_1+n_2}$.

Problem 2a

There is an algorithm that will, given two DFA's M_1, M_2 of sizes n_1, n_2, returns a DFA for $L(M_1) \cdot L(M_2)$ of size **FILL IN**.

1) Input two DFA's M_1, M_2 of sizes n_1, n_2.

2) Add-an-e-transition construction: NFA M such that
$L(M) = L(M_1)L(M_2)$.
M is of size $n_1 + n_2$.

3) Powerset construction on M: DFA D
$L(D) = L(M_1)L(M_2)$.
D has size $2^{n_1+n_2}$.

FILL IN is $2^{n_1+n_2}$.

Reality The blowup in real life is no where near exponential.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.
2) Induction Construction: NFAs M_1, M_2 such that

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap \ L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that $L(M_1) = L(\alpha_1)$ and M_1 is of size $\leq 2n_1$.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\leq 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\leq 2n_2$.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\leq 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\leq 2n_2$.

3) Powerset Construction: DFAs D_1 and D_2 such that

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap\ L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Powerset Construction: DFAs D_1 and D_2 such that
$L(D_1) = L(M_1) = L(\alpha_1)$ and D_1 has 2^{2n_1} states.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Powerset Construction: DFAs D_1 and D_2 such that
$L(D_1) = L(M_1) = L(\alpha_1)$ and D_1 has 2^{2n_1} states.
$L(D_2) = L(M_2) = L(\alpha_2)$ and D_2 has 2^{2n_2} states.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap\ L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Powerset Construction: DFAs D_1 and D_2 such that
$L(D_1) = L(M_1) = L(\alpha_1)$ and D_1 has 2^{2n_1} states.
$L(D_2) = L(M_2) = L(\alpha_2)$ and D_2 has 2^{2n_2} states.

4) Cross product Const: DFA D such that

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Powerset Construction: DFAs D_1 and D_2 such that
$L(D_1) = L(M_1) = L(\alpha_1)$ and D_1 has 2^{2n_1} states.
$L(D_2) = L(M_2) = L(\alpha_2)$ and D_2 has 2^{2n_2} states.

4) Cross product Const: DFA D such that
$L(D) = L(M_1)L(M_2)$ and D has of size $2^{2n_1}2^{2n_2} = 2^{2n_1+2n_2}$.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap\ L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Powerset Construction: DFAs D_1 and D_2 such that
$L(D_1) = L(M_1) = L(\alpha_1)$ and D_1 has 2^{2n_1} states.
$L(D_2) = L(M_2) = L(\alpha_2)$ and D_2 has 2^{2n_2} states.

4) Cross product Const: DFA D such that
$L(D) = L(M_1)L(M_2)$ and D has of size $2^{2n_1}2^{2n_2} = 2^{2n_1+2n_2}$.

5) $R(i,j,k)$: Regex of size $2^{O(2^{2n_1+2n_2})}$.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Powerset Construction: DFAs D_1 and D_2 such that
$L(D_1) = L(M_1) = L(\alpha_1)$ and D_1 has 2^{2n_1} states.
$L(D_2) = L(M_2) = L(\alpha_2)$ and D_2 has 2^{2n_2} states.

4) Cross product Const: DFA D such that
$L(D) = L(M_1)L(M_2)$ and D has of size $2^{2n_1}2^{2n_2} = 2^{2n_1+2n_2}$.

5) $R(i, j, k)$: Regex of size $2^{O(2^{2n_1+2n_2})}$.
FILL IN is $2^{O(2^{2n_1+2n_2})}$.

Problem 2b

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Powerset Construction: DFAs D_1 and D_2 such that
$L(D_1) = L(M_1) = L(\alpha_1)$ and D_1 has 2^{2n_1} states.
$L(D_2) = L(M_2) = L(\alpha_2)$ and D_2 has 2^{2n_2} states.

4) Cross product Const: DFA D such that
$L(D) = L(M_1)L(M_2)$ and D has of size $2^{2n_1}2^{2n_2} = 2^{2n_1+2n_2}$.

5) $R(i,j,k)$: Regex of size $2^{O(2^{2n_1+2n_2})}$.
FILL IN is $2^{O(2^{2n_1+2n_2})}$.
This answer gets FULL CREDIT but there is a BETTER way on next page.

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap\ L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that $L(M_1) = L(\alpha_1)$ and M_1 is of size $\leq 2n_1$.

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Cross Product Cont for NFAS: NFAs N such that

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Cross Product Cont for NFAS: NFAs N such that
$L(N) = L(M_1) \cap L(M_2)$

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Cross Product Cont for NFAS: NFAs N such that
$L(N) = L(M_1) \cap L(M_2)$
M has $2n_1 \times 2n_2 = 4n_1n_2$ states.

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap \ L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Cross Product Cont for NFAS: NFAs N such that
$L(N) = L(M_1) \cap L(M_2)$
M has $2n_1 \times 2n_2 = 4n_1 n_2$ states.

5) $R(i, j, k)$ on NFAs: Regex of size $2^{O(n_1 n_2)}$.

Problem 2b- Alt Solution

There is an algorithm that will, given two regex's α_1, α_2 of sizes n_1, n_2, returns a regex for $L(\alpha_1) \cap\ L(\alpha_2)$ of size **FILL IN**.

1) Input two regex's α_1, α_2 of sizes n_1, n_2.

2) Induction Construction: NFAs M_1, M_2 such that
$L(M_1) = L(\alpha_1)$ and M_1 is of size $\le 2n_1$.
$L(M_2) = L(\alpha_2)$ and M_2 is of size $\le 2n_2$.

3) Cross Product Cont for NFAS: NFAs N such that
$L(N) = L(M_1) \cap L(M_2)$
M has $2n_1 \times 2n_2 = 4n_1n_2$ states.

5) $R(i,j,k)$ on NFAs: Regex of size $2^{O(n_1n_2)}$.

FILL IN is $2^{4n_1n_2}$.

Problem 3a and 3b

Problem 3a and 3b

$$\{a^{\lfloor \log_2(n) \rfloor} : n \geq 1\}$$

Problem 3a and 3b

$\{a^{\lfloor \log_2(n) \rfloor} : n \geq 1\}$
REGULAR: This is aa^*.

Problem 3a and 3b

$\{a^{\lfloor \log_2(n) \rfloor} : n \geq 1\}$
REGULAR: This is aa^*.

$\{a^{2n} : n \geq 1\}$

Problem 3a and 3b

$\{a^{\lfloor \log_2(n) \rfloor} : n \geq 1\}$
REGULAR: This is aa^*.

$\{a^{2n} : n \geq 1\}$
REGULAR: This is $(aa)(aa)^*$

Problem 3c: $L = \{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular

Problem 3c: $L = \{a^{2^n} : n \in \mathsf{N}\}$ is Not Regular

Intuition POW2 keep getting further apart.

Problem 3c: $L = \{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular

Intuition POW2 keep getting further apart.
Pumping Lemma says you can always add some constant k to produce a word in the language.

Problem 3c: $L = \{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular

Intuition POW2 keep getting further apart.
Pumping Lemma says you can always add some constant k to produce a word in the language.

Proof

Problem 3c: $L = \{a^{2^n} : n \in \mathsf{N}\}$ is Not Regular

Intuition POW2 keep getting further apart.
Pumping Lemma says you can always add some constant k to produce a word in the language.

Proof

By Pumping Lemma for long enough $a^{2^n} \in L$ there exist $x = a^j$, $y = a^k$, $z = a^\ell$ with $xyz = a^{2^n}$. Also $a^j(a^k)^i a^\ell \in L$.
(Note $k \geq 1$.)

Problem 3c: $L = \{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular

Intuition POW2 keep getting further apart.
Pumping Lemma says you can always add some constant k to produce a word in the language.

Proof
By Pumping Lemma for long enough $a^{2^n} \in L$ there exist $x = a^j$, $y = a^k$, $z = a^\ell$ with $xyz = a^{2^n}$. Also $a^j(a^k)^i a^\ell \in L$. (Note $k \geq 1$.)

$$(\forall i \geq 0)[j + ik + \ell \text{ is a POW2}].$$

Recall that $j + k + \ell = 2^n$.

Problem 3c: $L = \{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular

Intuition POW2 keep getting further apart.
Pumping Lemma says you can always add some constant k to produce a word in the language.

Proof
By Pumping Lemma for long enough $a^{2^n} \in L$ there exist $x = a^j$, $y = a^k$, $z = a^\ell$ with $xyz = a^{2^n}$. Also $a^j(a^k)^i a^\ell \in L$. (Note $k \geq 1$.)

$$(\forall i \geq 0)[j + ik + \ell \text{ is a POW2}].$$

Recall that $j + k + \ell = 2^n$.
Hence

$$(\forall i \geq 0)[j + ik + \ell = 2^n + (i-1)k \text{ is a POW2}].$$

Problem 3c: $L = \{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular

Intuition POW2 keep getting further apart.
Pumping Lemma says you can always add some constant k to produce a word in the language.

Proof
By Pumping Lemma for long enough $a^{2^n} \in L$ there exist $x = a^j$, $y = a^k$, $z = a^\ell$ with $xyz = a^{2^n}$. Also $a^j(a^k)^i a^\ell \in L$. (Note $k \geq 1$.)

$$(\forall i \geq 0)[j + ik + \ell \text{ is a POW2}].$$

Recall that $j + k + \ell = 2^n$.
Hence

$$(\forall i \geq 0)[j + ik + \ell = 2^n + (i-1)k \text{ is a POW2}].$$

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.

Problem 3c: $L = \{a^{2^n} : n \in \mathsf{N}\}$ is Not Regular

Intuition POW2 keep getting further apart.
Pumping Lemma says you can always add some constant k to produce a word in the language.

Proof
By Pumping Lemma for long enough $a^{2^n} \in L$ there exist $x = a^j$, $y = a^k$, $z = a^\ell$ with $xyz = a^{2^n}$. Also $a^j(a^k)^i a^\ell \in L$. (Note $k \geq 1$.)

$$(\forall i \geq 0)[j + ik + \ell \text{ is a POW2}].$$

Recall that $j + k + \ell = 2^n$.
Hence

$$(\forall i \geq 0)[j + ik + \ell = 2^n + (i-1)k \text{ is a POW2}].$$

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.
See slide for exciting finish!

$\{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular (cont)

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.

$\{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular (cont)

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.

$2^n + k \geq 2^{n+1}$. So $k \geq 2^{n+1} - 2^n = 2^n$.

$\{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular (cont)

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.

$2^n + k \geq 2^{n+1}$. So $k \geq 2^{n+1} - 2^n = 2^n$.

$2^n + 2k \geq 2^{n+2}$. So $2k \geq 2^{n+2} - 2^n = 2^n \times 3$, so $k \geq \frac{3}{2} \times 2^n$.

$\{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular (cont)

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.

$2^n + k \geq 2^{n+1}$. So $k \geq 2^{n+1} - 2^n = 2^n$.

$2^n + 2k \geq 2^{n+2}$. So $2k \geq 2^{n+2} - 2^n = 2^n \times 3$, so $k \geq \frac{3}{2} \times 2^n$.

$\vdots$

$\{a^{2^n} : n \in \mathsf{N}\}$ is Not Regular (cont)

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.

$2^n + k \ge 2^{n+1}$. So $k \ge 2^{n+1} - 2^n = 2^n$.

$2^n + 2k \ge 2^{n+2}$. So $2k \ge 2^{n+2} - 2^n = 2^n \times 3$, so $k \ge \frac{3}{2} \times 2^n$.

$\vdots$

So
$(\forall i \ge 1)[2^n + ik \ge 2^{n+i}]$. So $ik \ge 2^{n+i} - 2^n = 2^n(2^i - 1)$ so
$k \ge \frac{2^i-1}{i}2^n$.

$\{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular (cont)

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.

$2^n + k \geq 2^{n+1}$. So $k \geq 2^{n+1} - 2^n = 2^n$.

$2^n + 2k \geq 2^{n+2}$. So $2k \geq 2^{n+2} - 2^n = 2^n \times 3$, so $k \geq \frac{3}{2} \times 2^n$.

$\vdots$

So
$(\forall i \geq 1)[2^n + ik \geq 2^{n+i}]$. So $ik \geq 2^{n+i} - 2^n = 2^n(2^i - 1)$ so
$k \geq \frac{2^i - 1}{i} 2^n$.
Key $\lim_{i \to \infty} \frac{2^i - 1}{i} = \infty$.

$\{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular (cont)

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.

$2^n + k \ge 2^{n+1}$. So $k \ge 2^{n+1} - 2^n = 2^n$.

$2^n + 2k \ge 2^{n+2}$. So $2k \ge 2^{n+2} - 2^n = 2^n \times 3$, so $k \ge \frac{3}{2} \times 2^n$.

$\vdots$

So
$(\forall i \ge 1)[2^n + ik \ge 2^{n+i}]$. So $ik \ge 2^{n+i} - 2^n = 2^n(2^i - 1)$ so
$k \ge \frac{2^i-1}{i}2^n$.
Key $\lim_{i \to \infty} \frac{2^i-1}{i} = \infty$.
So k is bigger than any natural number!

$\{a^{2^n} : n \in \mathbb{N}\}$ is Not Regular (cont)

So 2^n, $2^n + k$, $2^n + 2k$, $\ldots$ are all POW2.

$2^n + k \ge 2^{n+1}$. So $k \ge 2^{n+1} - 2^n = 2^n$.

$2^n + 2k \ge 2^{n+2}$. So $2k \ge 2^{n+2} - 2^n = 2^n \times 3$, so $k \ge \frac{3}{2} \times 2^n$.

$\vdots$

So
$(\forall i \ge 1)[2^n + ik \ge 2^{n+i}]$. So $ik \ge 2^{n+i} - 2^n = 2^n(2^i - 1)$ so
$k \ge \frac{2^i-1}{i} 2^n$.
Key $\lim_{i \to \infty} \frac{2^i-1}{i} = \infty$.
So k is bigger than any natural number!
Contradiction.