# Converting Self-Verifying Automata into Deterministic Automata

Galina Jirásková[1] Giovanni Pighizzini[2]

[1]Mathematical Institute
Slovak Academy of Sciences
Košice, Slovakia

[2]Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
Milano, Italy

LATA 2009 – Tarragona – April 7th, 2009

# Self-verifying machines

Standard machines (e.g. finite automata, pushdown automata, Turing machines) with nondeterministic transitions

The state set is partitioned in three groups:

- *accepting* states ("yes")
- *rejecting* states ("no")
- *neutral* states ("I do not know")

For each input word $x$ the following conditions must be satisfyied:

- At least one computation on input $x$ ends either in an accepting or in a rejecting state
- If a computation on $x$ ends in an accepting state then there are no computations on $x$ ending in rejecting states

# Self-verifying machines

Standard machines (e.g. finite automata, pushdown automata, Turing machines) with nondeterministic transitions

The state set is partitioned in three groups:

- *accepting* states ("yes")
- *rejecting* states ("no")
- *neutral* states ("I do not know")

For each input word $x$ the following conditions must be satisfyied:

- At least one computation on input $x$ ends either in an accepting or in a rejecting state
- If a computation on $x$ ends in an accepting state then there are no computations on $x$ ending in rejecting states

# Self-verifying machines

Standard machines (e.g. finite automata, pushdown automata, Turing machines) with nondeterministic transitions

The state set is partitioned in three groups:

- *accepting* states ("yes")
- *rejecting* states ("no")
- *neutral* states ("I do not know")

For each input word $x$ the following conditions must be satisfyied:

- At least one computation on input $x$ ends either in an accepting or in a rejecting state
- If a computation on $x$ ends in an accepting state then there are no computations on $x$ ending in rejecting states

# Self-verifying machines

Standard machines (e.g. finite automata, pushdown automata, Turing machines) with nondeterministic transitions

The state set is partitioned in three groups:

- *accepting* states ("yes")
- *rejecting* states ("no")
- *neutral* states ("I do not know")

For each input word $x$ the following conditions must be satisfyied:

- At least one computation on input $x$ ends either in an accepting or in a rejecting state
- If a computation on $x$ ends in an accepting state then there are no computations on $x$ ending in rejecting states

# Self-verifying machines

Some references:

- Ďuriš, Hromkovič, Rolim, and Schnitger (STACS 1997)
  Definition of the model in connection with the study of Las Vegas automata.

- Hromkovič and Schnitger (Information and Comp. 2001)
  Hromkovič and Schnitger (SIAM J. Comp. 2003)
  Further investigations in connection with Las Vegas computations and also *per se*.

- Assent and Seibert (RAIRO-ITA 2007)
  Simulation of self-verifying automata by deterministic automata.

# Basic properties

- Trivial complementation
- Given nondeterministic machines $M'$ and $M''$ for $L$ and $L^c$, we can build a self-verifying machine $M$ for $L$ as the "union" of $M'$ and $M''$, with a new initial state:




- Given a self-verifying machine for a language $L$ we can easily obtain nondeterministic machines for $L$ and for $L^c$

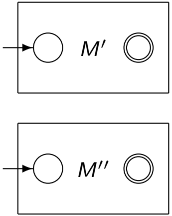
- Trivial complementation
- Given nondeterministic machines $M'$ and $M''$ for $L$ and $L^c$, we can build a self-verifying machine $M$ for $L$ as the "union" of $M'$ and $M''$, with a new initial state:




- Given a self-verifying machine for a language $L$ we can easily obtain nondeterministic machines for $L$ and for $L^c$

# Basic properties

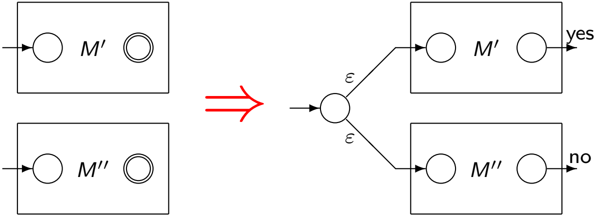
- Trivial complementation
- Given nondeterministic machines $M'$ and $M''$ for $L$ and $L^c$, we can build a self-verifying machine $M$ for $L$
  as the "union" of $M'$ and $M''$, with a new initial state:




- Given a self-verifying machine for a language $L$ we can easily obtain nondeterministic machines for $L$ and for $L^c$

# Basic properties

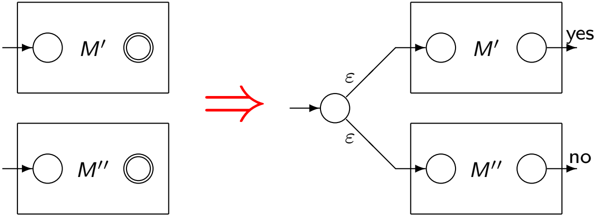
- Trivial complementation
- Given nondeterministic machines $M'$ and $M''$ for $L$ and $L^c$, we can build a self-verifying machine $M$ for $L$
  as the "union" of $M'$ and $M''$, with a new initial state:




- Given a self-verifying machine for a language $L$ we can easily obtain nondeterministic machines for $L$ and for $L^c$

# Basic properties

- Trivial complementation
- Given nondeterministic machines $M'$ and $M''$ for $L$ and $L^c$, we can build a self-verifying machine $M$ for $L$
as the "union" of $M'$ and $M''$, with a new initial state:




- Given a self-verifying machine for a language $L$ we can easily obtain nondeterministic machines for $L$ and for $L^c$

$A = (Q, \Sigma, \delta, q_0, F^a, F^r)$ where:

- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
- $q_0 \in Q$ is the initial state
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function
- $F^a \subseteq Q$ is the *set of accepting states*
- $F^r \subseteq Q$ is the *set of rejecting states*, $F^a \cap F^r = \emptyset$

# Self-verifying automata (svfa): definition

$A = (Q, \Sigma, \delta, q_0, F^a, F^r)$ where:

- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
- $q_0 \in Q$ is the initial state
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function
- $F^a \subseteq Q$ is the *set of accepting states*
- $F^r \subseteq Q$ it the *set of rejecting states*, s.t. $F^a \cap F^r = \emptyset$
- $Q - (F^a \cup F^r)$ is the set of *neutral states*

# Self-verifying automata (svfa): definition

$A = (Q, \Sigma, \delta, q_0, F^a, F^r)$ where:

- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
- $q_0 \in Q$ is the initial state
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function
- $F^a \subseteq Q$ is the *set of accepting states*
- $F^r \subseteq Q$ it the *set of rejecting states*, s.t. $F^a \cap F^r = \emptyset$
- $Q - (F^a \cup F^r)$ is the set of *neutral states*

# Self-verifying automata (svfa): definition

$A = (Q, \Sigma, \delta, q_0, F^a, F^r)$ where:

- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
- $q_0 \in Q$ is the initial state
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function
- $F^a \subseteq Q$ is the *set of accepting states*
- $F^r \subseteq Q$ it the *set of rejecting states*, s.t. $F^a \cap F^r = \emptyset$
- $Q - (F^a \cup F^r)$ is the set of *neutral states*

# Self-verifying automata (svfa): definition

$A = (Q, \Sigma, \delta, q_0, F^a, F^r)$ where:

- $Q$ is the finite set of states
- $\Sigma$ is the input alphabet
- $q_0 \in Q$ is the initial state
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function
- $F^a \subseteq Q$ is the *set of accepting states*
- $F^r \subseteq Q$ it the *set of rejecting states*, s.t. $F^a \cap F^r = \emptyset$
- $Q - (F^a \cup F^r)$ is the set of *neutral states*

# Self-verifying automata (svfa): definition

The following conditions must be satisfied:

- For each $w \in \Sigma^*$: $\delta(q_0, w) \cap (F^a \cup F^r) \neq \emptyset$
  namely, for each string there exists at least one accepting computation or one rejecting computation
- There are no strings $w \in \Sigma^*$ s.t. $\delta(q_0, w) \cap F^a \neq \emptyset$ and $\delta(q_0, w) \cap F^r \neq \emptyset$
  namely, the automaton cannot give contradictory answers

# Self-verifying automata (svfa): definition

The following conditions must be satisfied:

- For each $w \in \Sigma^*$: $\delta(q_0, w) \cap (F^a \cup F^r) \neq \emptyset$
  namely, for each string there exists at least one accepting computation or one rejecting computation

- There are no strings $w \in \Sigma^*$ s.t. $\delta(q_0, w) \cap F^a \neq \emptyset$ and $\delta(q_0, w) \cap F^r \neq \emptyset$
  namely, the automaton cannot give contradictory answers

# Self-verifying automata (svfa): definition

The following conditions must be satisfied:

- For each $w \in \Sigma^*$: $\delta(q_0, w) \cap (F^a \cup F^r) \neq \emptyset$
  namely, for each string there exists at least one accepting computation or one rejecting computation

- There are no strings $w \in \Sigma^*$ s.t. $\delta(q_0, w) \cap F^a \neq \emptyset$ and $\delta(q_0, w) \cap F^r \neq \emptyset$
  namely, the automaton cannot give contradictory answers

# Languages of svfa's

We associate with an svfa $A$ the following languages:

- The set of strings *accepted* by $A$:

$$L^a(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^a \neq \emptyset\}$$

- The set of strings *rejected* by $A$:

$$L^r(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^r \neq \emptyset\}$$

By the previous conditions $L^r(A) = \Sigma^* - L^a(A)$.

The *language accepted* by $A$ is defined as $L^a(A)$.

# Languages of svfa's

We associate with an svfa $A$ the following languages:

- The set of strings *accepted* by $A$:

$$L^a(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^a \neq \emptyset\}$$

- The set of strings *rejected* by $A$:

$$L^r(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^r \neq \emptyset\}$$

By the previous conditions $L^r(A) = \Sigma^* - L^a(A)$.

The *language accepted* by $A$ is defined as $L^a(A)$.

We associate with an svfa $A$ the following languages:

- The set of strings *accepted* by $A$:

$$L^a(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^a \neq \emptyset\}$$

- The set of strings *rejected* by $A$:

$$L^r(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^r \neq \emptyset\}$$

By the previous conditions $L^r(A) = \Sigma^* - L^a(A)$.

The *language accepted* by $A$ is defined as $L^a(A)$.

# Languages of svfa's

We associate with an svfa $A$ the following languages:

- The set of strings *accepted* by $A$:

$$L^a(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^a \neq \emptyset\}$$

- The set of strings *rejected* by $A$:

$$L^r(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^r \neq \emptyset\}$$

By the previous conditions $L^r(A) = \Sigma^* - L^a(A)$.

The *language accepted* by $A$ is defined as $L^a(A)$.

# Languages of svfa's

We associate with an svfa $A$ the following languages:

- The set of strings *accepted* by $A$:

$$L^a(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^a \neq \emptyset\}$$

- The set of strings *rejected* by $A$:

$$L^r(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F^r \neq \emptyset\}$$

By the previous conditions $L^r(A) = \Sigma^* - L^a(A)$.

The *language accepted* by $A$ is defined as $L^a(A)$.

*First question*

What is the class of languages accepted by svfa's?

The answer to this question is easy:

- Each svfa is a nondeterministic automaton
- Each deterministic automaton is also an svfa

Hence:

Svfa's characterize the class of *regular languages*

Thus, each svfa can be converted into an equivalent dfa.

*First question*

What is the class of languages accepted by svfa's?

The answer to this question is easy:

- Each svfa is a nondeterministic automaton
- Each deterministic automaton is also an svfa

Hence:

Svfa's characterize the class of *regular languages*

Thus, each svfa can be converted into an equivalent dfa.

# Questions about svfa's

*First question*

What is the class of languages accepted by svfa's?

The answer to this question is easy:

- Each svfa is a nondeterministic automaton
- Each deterministic automaton is also an svfa

Hence:

Svfa's characterize the class of *regular languages*

Thus, each svfa can be converted into an equivalent dfa.

# Questions about svfa's

*Second question*

How much it costs, in terms of states, the conversion of an $n$-state svfa into an equivalent dfa?

- Classical subset construction: upper bound $2^n$

*Second question*

How much it costs, in terms of states, the conversion of an $n$-state svfa into an equivalent dfa?

- Classical subset construction: upper bound $2^n$
- It is possible to do better: Assent and Seibert (2007) reduced the upper bound to $O\left(\frac{2^n}{\sqrt{n}}\right)$, leaving open the optimality

In this work we further investigate this problem:

- We reduce the upper bound to a function $g(n)$ which grows like $3^{\frac{n}{3}}$
- We prove that our upper bound $g(n)$ is tight

*Second question*

How much it costs, in terms of states, the conversion of an $n$-state svfa into an equivalent dfa?

- Classical subset construction: upper bound $2^n$
- It is possible to do better: Assent and Seibert (2007) reduced the upper bound to $O\left(\frac{2^n}{\sqrt{n}}\right)$, leaving open the optimality

In this work we further investigate this problem:

- We reduce the upper bound to a function $g(n)$ which grows like $3^{\frac{n}{3}}$
- We prove that our upper bound $g(n)$ is tight

# Questions about svfa's

*Second question*

How much it costs, in terms of states, the conversion of an $n$-state svfa into an equivalent dfa?

- Classical subset construction: upper bound $2^n$
- It is possible to do better: Assent and Seibert (2007) reduced the upper bound to $O\left(\frac{2^n}{\sqrt{n}}\right)$, leaving open the optimality

In this work we further investigate this problem:

- We reduce the upper bound to a function $g(n)$ which grows like $3^{\frac{n}{3}}$
- We prove that our upper bound $g(n)$ is tight

*Second question*

How much it costs, in terms of states, the conversion of an $n$-state svfa into an equivalent dfa?

- Classical subset construction: upper bound $2^n$
- It is possible to do better: Assent and Seibert (2007) reduced the upper bound to $O\left(\frac{2^n}{\sqrt{n}}\right)$, leaving open the optimality

In this work we further investigate this problem:

- We reduce the upper bound to a function $g(n)$ which grows like $3^{\frac{n}{3}}$
- We prove that our upper bound $g(n)$ is tight

# Conversion of svfa's into dfa's

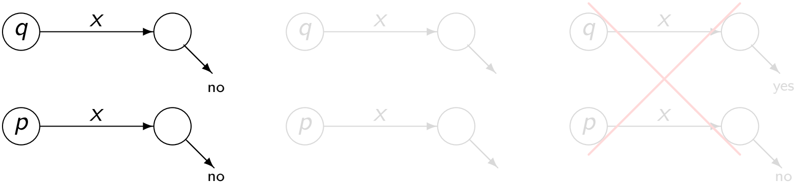
Let $A$ be an svfa

- Two states $q, p$ of $A$ are said to be *compatible* iff starting from them and reading a same string $x$ it is not possible to obtain contradictory answers

# Conversion of svfa's into dfa's

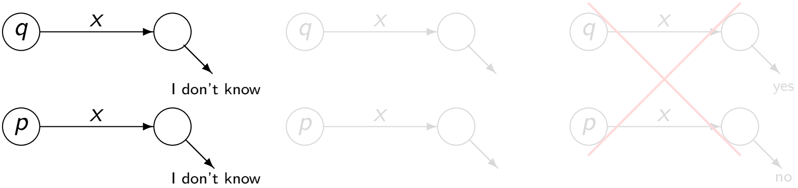
Let $A$ be an svfa

- Two states $q, p$ of $A$ are said to be *compatible* iff starting from them and reading a same string $x$ it is not possible to obtain contradictory answers



Let $A$ be an svfa

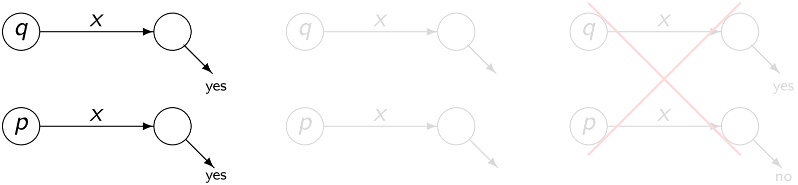
- Two states $q, p$ of $A$ are said to be *compatible* iff starting from them and reading a same string $x$ it is not possible to obtain contradictory answers



# Conversion of svfa's into dfa's

Let $A$ be an svfa

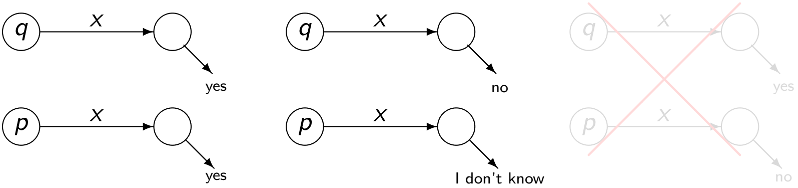
- Two states $q, p$ of $A$ are said to be *compatible* iff starting from them and reading a same string $x$ it is not possible to obtain contradictory answers



# Conversion of svfa's into dfa's

Let $A$ be an svfa

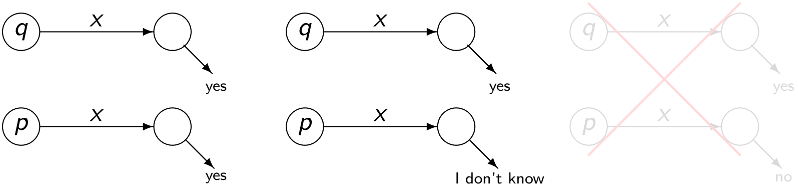
- Two states $q, p$ of $A$ are said to be *compatible* iff starting from them and reading a same string $x$ it is not possible to obtain contradictory answers



# Conversion of svfa's into dfa's

Let $A$ be an svfa

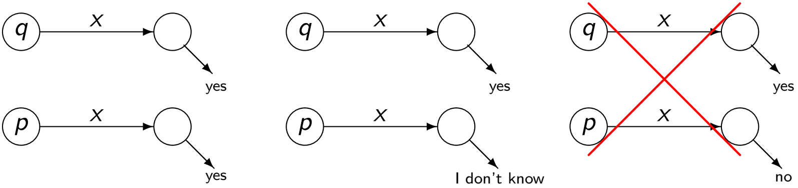
- Two states $q, p$ of $A$ are said to be *compatible* iff starting from them and reading a same string $x$ it is not possible to obtain contradictory answers



# Conversion of svfa's into dfa's

Let $A$ be an svfa

- Two states $q, p$ of $A$ are said to be *compatible* iff starting from them and reading a same string $x$ it is not possible to obtain contradictory answers



# The subset automaton $A_{sub}$

- Using the standard subset construction, from the given svfa $A$ we build a dfa
- Let $A_{sub}$ such a dfa, *restricted to its reachable states*
- We study the properties of $A_{sub}$

# The subset automaton $A_{sub}$

- Using the standard subset construction, from the given svfa $A$ we build a dfa
- Let $A_{sub}$ such a dfa, *restricted to its reachable states*
- We study the properties of $A_{sub}$
- Our goal is to discover which states of $A_{sub}$ are redundant

# The subset automaton $A_{sub}$

- Using the standard subset construction, from the given svfa $A$ we build a dfa
- Let $A_{sub}$ such a dfa, *restricted to its reachable states*
- We study the properties of $A_{sub}$
- Our goal is to discover which states of $A_{sub}$ are equivalent

# The subset automaton $A_{sub}$

- Using the standard subset construction, from the given svfa $A$ we build a dfa
- Let $A_{sub}$ such a dfa, *restricted to its reachable states*
- We study the properties of $A_{sub}$
- Our goal is to discover which states of $A_{sub}$ are equivalent

# The subset automaton $A_{sub}$

- Using the standard subset construction, from the given svfa $A$ we build a dfa
- Let $A_{sub}$ such a dfa, *restricted to its reachable states*
- We study the properties of $A_{sub}$
- Our goal is to discover which states of $A_{sub}$ are equivalent

# Properties of the subset automaton

Let $\alpha$ be a state of the subset automaton $A_{sub}$. Then:

Each two states $q, p \in \alpha$ are compatible

*Proof*
If $q, p \in \alpha$ are not compatible then:




Given a string $y$ s.t. $\alpha$ is reached on $y$, the original svfa on $yx$ should give contradictory answers!

Let $\alpha$ be a state of the subset automaton $A_{sub}$. Then:

Each two states $q, p \in \alpha$ are compatible

*Proof*

If $q, p \in \alpha$ are not compatible then:



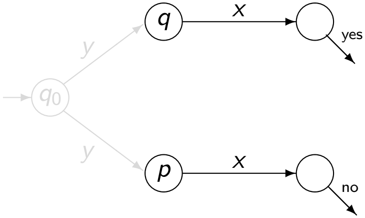

Given a string $y$ s.t. $\alpha$ is reached on $y$, the original svfa on $yx$ should give contradictory answers!

Let $\alpha$ be a state of the subset automaton $A_{sub}$. Then:

Each two states $q, p \in \alpha$ are compatible

*Proof*

If $q, p \in \alpha$ are not compatible then:



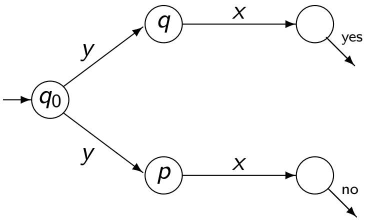

Given a string $y$ s.t. $\alpha$ is reached on $y$, the original svfa on $yx$ should give contradictory answers!

# Properties of the subset automaton

Let $\alpha$ be a state of the subset automaton $A_{sub}$. Then:

Each two states $q, p \in \alpha$ are compatible

*Proof*
If $q, p \in \alpha$ are not compatible then:




Given a string $y$ s.t. $\alpha$ is reached on $y$, the original svfa on $yx$ should give contradictory answers!

# Properties of the subset automaton

Let $\alpha$ be a state of the subset automaton $A_{sub}$. Then:

For each $x \in \Sigma^*$ there exists a state $q \in \alpha$ s.t.
$\delta(q, x) \cap (F^a \cup F^r) \neq \emptyset$

*Proof*

If starting from each $q \in \alpha$, the answer on $x$ is "I don't know":




Given a string $y$ s.t. $\alpha$ is reached on $y$, the original svfa on $yx$ cannot give any answer!

# Properties of the subset automaton

Let $\alpha$ be a state of the subset automaton $A_{sub}$. Then:

For each $x \in \Sigma^*$ there exists a state $q \in \alpha$ s.t.
$$\delta(q, x) \cap (F^a \cup F^r) \neq \emptyset$$

*Proof*
If starting from each $q \in \alpha$, the answer on $x$ is "I don't know":



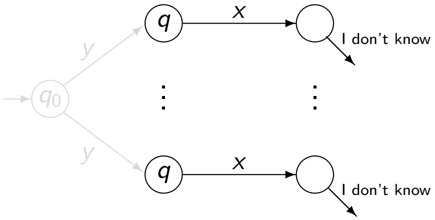

Given a string $y$ s.t. $\alpha$ is reached on $y$, the original svfa on $yx$ cannot give any answer!

# Properties of the subset automaton

Let $\alpha$ be a state of the subset automaton $A_{sub}$. Then:

For each $x \in \Sigma^*$ there exists a state $q \in \alpha$ s.t.
$$\delta(q, x) \cap (F^a \cup F^r) \neq \emptyset$$

*Proof*
If starting from each $q \in \alpha$, the answer on $x$ is "I don't know":



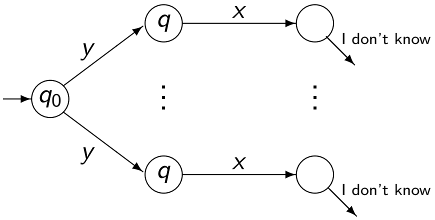

Given a string $y$ s.t. $\alpha$ is reached on $y$, the original svfa on $yx$ cannot give any answer!

# Properties of the subset automaton

Let $\alpha$ be a state of the subset automaton $A_{sub}$. Then:

For each $x \in \Sigma^*$ there exists a state $q \in \alpha$ s.t.
$\delta(q, x) \cap (F^a \cup F^r) \neq \emptyset$

*Proof*
If starting from each $q \in \alpha$, the answer on $x$ is "I don't know":




Given a string $y$ s.t. $\alpha$ is reached on $y$, the original svfa on $yx$ cannot give any answer!

# Compatibility graph

We define the following *compatibility graph* $G$, associated with the given svfa $A$:

- The nodes of $G$ are the states of $A$
- Two states $q, p$ are connected by an edge iff $q$ and $p$ are compatible

Hence:

each state of $\hat{A}_{sub}$ represents a clique of $G$

We define the following *compatibility graph* $G$, associated with the given svfa $A$:

- The nodes of $G$ are the states of $A$
- Two states $q, p$ are connected by an edge iff $q$ and $p$ are compatible

Hence:

each state of $A_{sub}$ represents a *clique* of $G$

# Compatibility graph

We define the following *compatibility graph* $G$, associated with the given svfa $A$:

- The nodes of $G$ are the states of $A$
- Two states $q, p$ are connected by an edge iff $q$ and $p$ are compatible

Hence:

each state of $A_{sub}$ represents a *clique* of $G$

# Compatibility graph

We define the following *compatibility graph* $G$, associated with the given svfa $A$:

- The nodes of $G$ are the states of $A$
- Two states $q, p$ are connected by an edge iff $q$ and $p$ are compatible

Hence:

each state of $A_{sub}$ represents a *clique* of $G$

# Properties of the subset automaton

Let $\alpha, \beta \subseteq Q$ two states of $A_{sub}$

If $\alpha \cup \beta$ is a clique of $G$ then $\alpha$ and $\beta$ are equivalent

# Properties of the subset automaton

Let $\alpha, \beta \subseteq Q$ two states of $A_{sub}$

If $\alpha \cup \beta$ is a clique of $G$ then $\alpha$ and $\beta$ are equivalent

*Proof*

By contradiction, let $x$ be a string distinguishing $\alpha$ and $\beta$:




subset automaton $A_{sub}$

Then $\exists q \in \alpha$, $p \in \beta$ s.t.:




given svfa $A$

This should imply that $q$ and $p$ are not compatible.

Hence, $\alpha \cup \beta$ cannot be a clique of $G$!

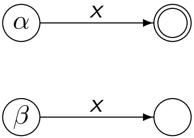
# Properties of the subset automaton

Let $\alpha, \beta \subseteq Q$ two states of $A_{sub}$

If $\alpha \cup \beta$ is a clique of $G$ then $\alpha$ and $\beta$ are equivalent

*Proof*

By contradiction, let $x$ be a string distinguishing $\alpha$ and $\beta$:




subset automaton $A_{sub}$

Then $\exists q \in \alpha$, $p \in \beta$ s.t.:




given svfa $A$

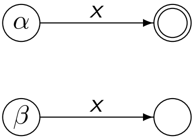
This should imply that $q$ and $p$ are not compatible.
Hence, $\alpha \cup \beta$ cannot be a clique of $G$!

# Properties of the subset automaton

Let $\alpha, \beta \subseteq Q$ two states of $A_{sub}$

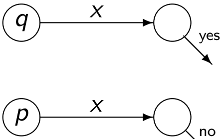
If $\alpha \cup \beta$ is a clique of $G$ then $\alpha$ and $\beta$ are equivalent

*Proof*
By contradiction, let $x$ be a string distinguishing $\alpha$ and $\beta$:




subset automaton $A_{sub}$

Then $\exists q \in \alpha$, $p \in \beta$ s.t.:




given svfa $A$

This should imply that $q$ and $p$ are not compatible.
Hence, $\alpha \cup \beta$ cannot be a clique of $G$!

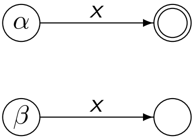
# Properties of the subset automaton

Let $\alpha, \beta \subseteq Q$ two states of $A_{sub}$

If $\alpha \cup \beta$ is a clique of $G$ then $\alpha$ and $\beta$ are equivalent

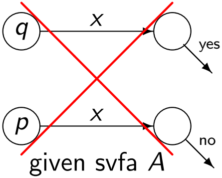
*Proof*

By contradiction, let $x$ be a string distinguishing $\alpha$ and $\beta$:




subset automaton $A_{sub}$

Then $\exists q \in \alpha$, $p \in \beta$ s.t.:




given svfa $A$

This should imply that $q$ and $p$ are not compatible.
Hence, $\alpha \cup \beta$ cannot be a clique of $G$!

# Subset automatom

By the previous properties:

- Each state of $A_{sub}$ corresponds to a clique of the compatibility graph $G$
- If the union of two states $\alpha, \beta$ of $A_{sub}$ is still a clique then $\alpha$ and $\beta$ are equivalent

Hence,

We can reduce the size of $A_{sub}$ by considering exactly one state for each *maximal clique of G*

In other words, the number of the states of the minimal dfa equivalent to $A$ is bounded by the number of maximal cliques of $G$

# Subset automatom

By the previous properties:

- Each state of $A_{sub}$ corresponds to a clique of the compatibility graph $G$
- If the union of two states $\alpha, \beta$ of $A_{sub}$ is still a clique then $\alpha$ and $\beta$ are equivalent

Hence,

We can reduce the size of $A_{sub}$ by considering exactly one state for each *maximal clique of G*

In other words, the number of the states of the minimal dfa equivalent to $A$ is bounded by the number of maximal cliques of $G$

# Subset automatom

By the previous properties:

- Each state of $A_{sub}$ corresponds to a clique of the compatibility graph $G$
- If the union of two states $\alpha, \beta$ of $A_{sub}$ is still a clique then $\alpha$ and $\beta$ are equivalent

Hence,

We can reduce the size of $A_{sub}$ by considering exactly one state for each *maximal clique of G*

In other words, the number of the states of the minimal dfa equivalent to $A$ is bounded by the number of maximal cliques of $G$

How many maximal cliques can a graph with $n$ nodes have?

This question was answered by Moon and Moser (1965).

They proved the following *exact bound* $f(n)$ for the maximum number of maximal cliques in a graph with $n$ nodes:

$$f(n) = \begin{cases} 3^{\lfloor \frac{n}{3} \rfloor} & \text{if } n \equiv 0 \pmod 3 \\ 4 \cdot 3^{\lfloor \frac{n}{3} \rfloor - 1} & \text{if } n \equiv 1 \pmod 3 \\ 2 \cdot 3^{\lfloor \frac{n}{3} \rfloor} & \text{if } n \equiv 2 \pmod 3 \end{cases}$$

# Cliques in graphs

How many maximal cliques can a graph with $n$ nodes have?

This question was answered by Moon and Moser (1965).

They proved the following *exact bound* $f(n)$ for the maximum number of maximal cliques in a graph with $n$ nodes:

$$f(n) = \begin{cases} 3^{\lfloor \frac{n}{3} \rfloor} & \text{if } n \equiv 0 \pmod 3 \\ 4 \cdot 3^{\lfloor \frac{n}{3} \rfloor - 1} & \text{if } n \equiv 1 \pmod 3 \\ 2 \cdot 3^{\lfloor \frac{n}{3} \rfloor} & \text{if } n \equiv 2 \pmod 3 \end{cases}$$

# Conversion of svfa's into dfa's: upper bound

Using the result of Moon and Moser, we can prove that

Each $n$-state svfa's can be simulated by a dfa with at most $g(n) = 1 + f(n-1)$ states

*Proof*

- We proved that $A_{sub}$ can be reduced to a dfa with at most one state for each maximal clique of $G$
- From the defintion, it follows that each two states which are compatible with $q_0$ are compatible with each other
- Hence $q_0$ belongs only to one maximal clique

Notice that $g(n) = O(3^{\frac{n}{3}})$

Using the result of Moon and Moser, we can prove that

Each $n$-state svfa's can be simulated by a dfa with at most $g(n) = 1 + f(n-1)$ states

*Proof*

- We proved that $A_{sub}$ can be reduced to a dfa with at most one state for each maximal clique of $G$
- From the defintion, it follows that each two states which are compatible with $q_0$ are compatible with each other
- Hence $q_0$ belongs only to one maximal clique

Notice that $g(n) = O(3^{\frac{n}{3}})$

# Conversion of svfa's into dfa's: upper bound

Using the result of Moon and Moser, we can prove that

Each $n$-state svfa's can be simulated by a dfa with at most $g(n) = 1 + f(n-1)$ states

*Proof*

- We proved that $A_{sub}$ can be reduced to a dfa with at most one state for each maximal clique of $G$
- From the defintion, it follows that each two states which are compatible with $q_0$ are compatible with each other
- Hence $q_0$ belongs only to one maximal clique
- The other maximal cliques can involve at most the remaining $n-1$ states, hence they are at most $f(n-1)$
- This gives the upper bound $g(n) = 1 + f(n-1)$

Notice that $g(n) = O(3^{\frac{n}{3}})$

# Conversion of svfa's into dfa's: upper bound

Using the result of Moon and Moser, we can prove that

Each $n$-state svfa's can be simulated by a dfa with at most $g(n) = 1 + f(n-1)$ states

*Proof*

- We proved that $A_{sub}$ can be reduced to a dfa with at most one state for each maximal clique of $G$
- From the defintion, it follows that each two states which are compatible with $q_0$ are compatible with each other
- Hence $q_0$ belongs only to one maximal clique
- The other maximal cliques can involve at most the remaining $n-1$ states, hence they are at most $f(n-1)$
- This gives the upper bound $g(n) = 1 + f(n-1)$

Notice that $g(n) = O(3^{\frac{n}{3}})$

Using the result of Moon and Moser, we can prove that

Each $n$-state svfa's can be simulated by a dfa with at most $g(n) = 1 + f(n-1)$ states

*Proof*

- We proved that $A_{sub}$ can be reduced to a dfa with at most one state for each maximal clique of $G$
- From the defintion, it follows that each two states which are compatible with $q_0$ are compatible with each other
- Hence $q_0$ belongs only to one maximal clique
- The other maximal cliques can involve at most the remaining $n-1$ states, hence they are at most $f(n-1)$
- This gives the upper bound $g(n) = 1 + f(n-1)$

Notice that $g(n) = O(3^{\frac{n}{3}})$

# Conversion of svfa's into dfa's: upper bound

Using the result of Moon and Moser, we can prove that

Each $n$-state svfa's can be simulated by a dfa with at most $g(n) = 1 + f(n-1)$ states

*Proof*

- We proved that $A_{sub}$ can be reduced to a dfa with at most one state for each maximal clique of $G$
- From the defintion, it follows that each two states which are compatible with $q_0$ are compatible with each other
- Hence $q_0$ belongs only to one maximal clique
- The other maximal cliques can involve at most the remaining $n-1$ states, hence they are at most $f(n-1)$
- This gives the upper bound $g(n) = 1 + f(n-1)$

Notice that $g(n) = O(3^{\frac{n}{3}})$

# Optimality

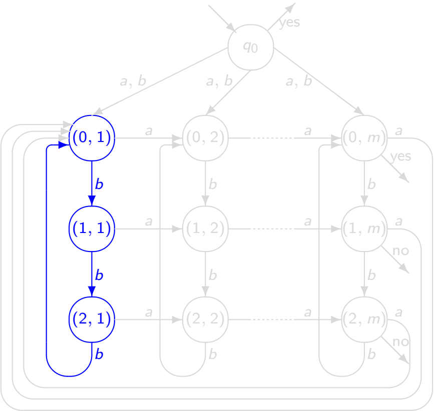
*The upper bound $g(n)$ is tight:*
for each integer $n \geq 1$ we can show an example of $n$-state svfa $A_n$ whose minimal equivalent dfa has exactly $g(n)$ states.

For $n = 3m + 1$, $A_n$ is the following:

# Optimality

*The upper bound $g(n)$ is tight:*
for each integer $n \geq 1$ we can show an example of $n$-state svfa $A_n$ whose minimal equivalent dfa has exactly $g(n)$ states.

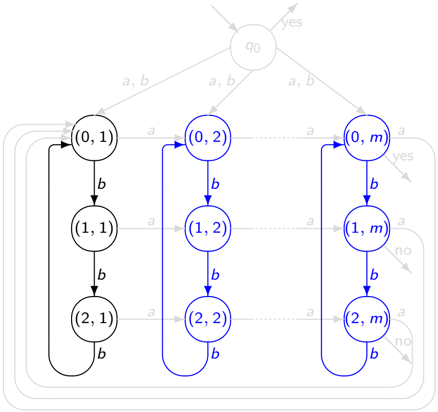
For $n = 3m + 1$, $A_n$ is the following:



# Optimality

*The upper bound $g(n)$ is tight:*
for each integer $n \geq 1$ we can show an example of $n$-state svfa $A_n$ whose minimal equivalent dfa has exactly $g(n)$ states.

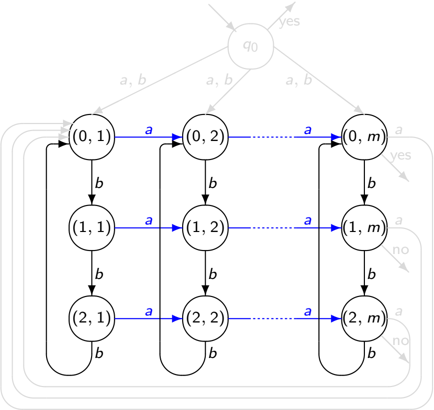
For $n = 3m + 1$, $A_n$ is the following:



# Optimality

*The upper bound $g(n)$ is tight:*
for each integer $n \geq 1$ we can show an example of $n$-state svfa $A_n$ whose minimal equivalent dfa has exactly $g(n)$ states.

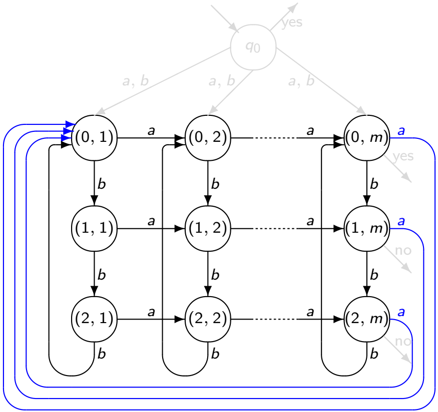
For $n = 3m + 1$, $A_n$ is the following:



# Optimality

*The upper bound $g(n)$ is tight:*
for each integer $n \geq 1$ we can show an example of $n$-state svfa $A_n$ whose minimal equivalent dfa has exactly $g(n)$ states.

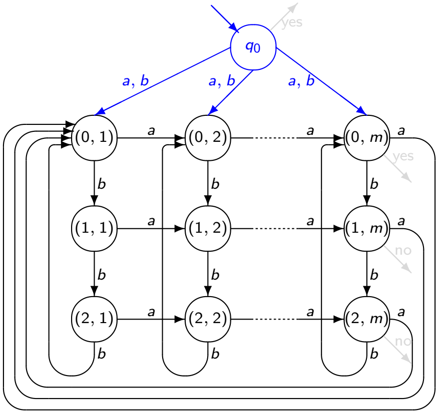
For $n = 3m + 1$, $A_n$ is the following:



# Optimality

*The upper bound $g(n)$ is tight:*
for each integer $n \geq 1$ we can show an example of $n$-state svfa $A_n$ whose minimal equivalent dfa has exactly $g(n)$ states.

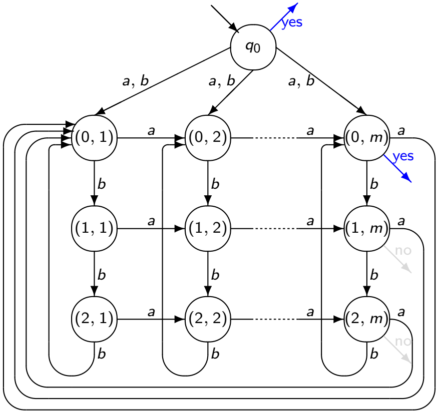
For $n = 3m + 1$, $A_n$ is the following:



# Optimality

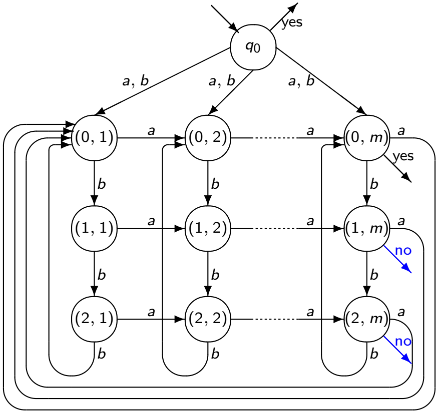
*The upper bound $g(n)$ is tight:*
for each integer $n \geq 1$ we can show an example of $n$-state svfa $A_n$ whose minimal equivalent dfa has exactly $g(n)$ states.

For $n = 3m + 1$, $A_n$ is the following:

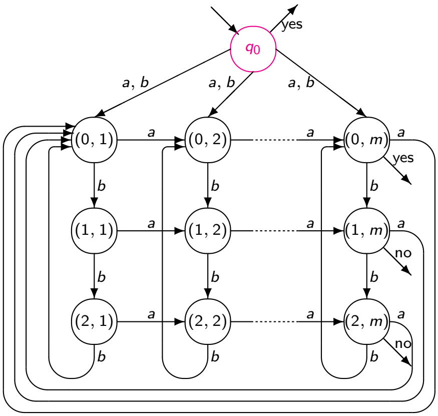


*The upper bound $g(n)$ is tight:*
for each integer $n \geq 1$ we can show an example of $n$-state svfa $A_n$ whose minimal equivalent dfa has exactly $g(n)$ states.

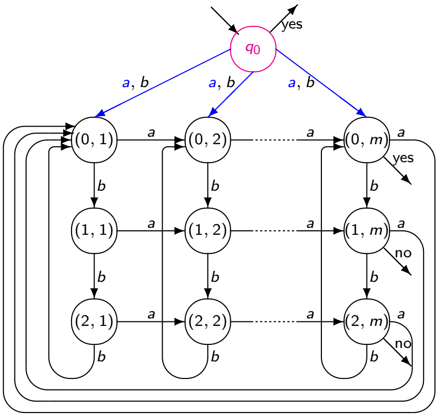
For $n = 3m + 1$, $A_n$ is the following:



# Properties of $A_n$

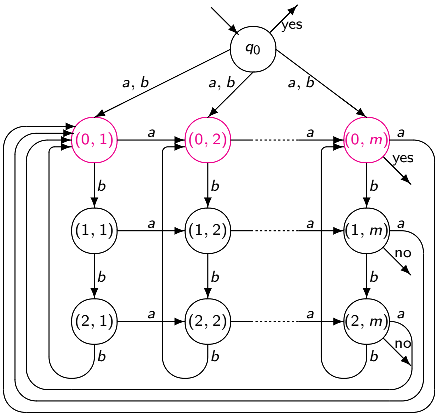


# Properties of $A_n$



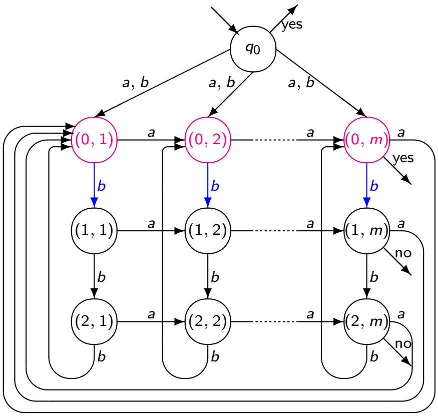

*a*

# Properties of $A_n$



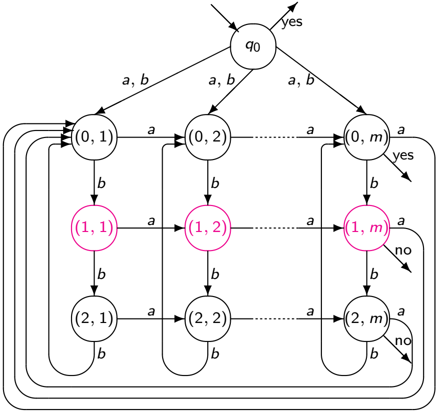

$a$

# Properties of $A_n$



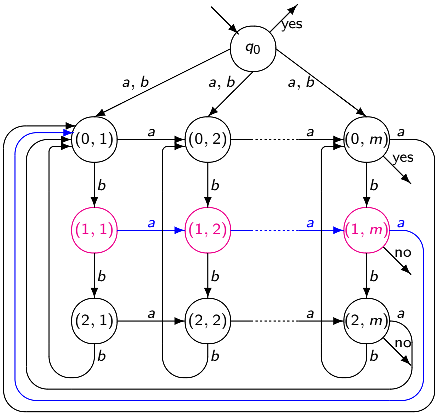

$ab$

# Properties of $A_n$



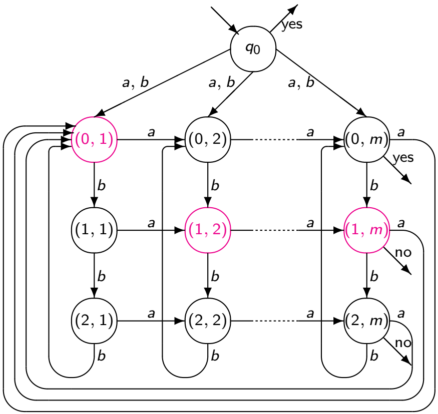

$ab$

# Properties of $A_n$

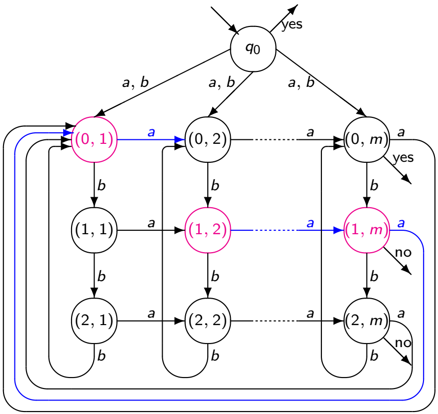



$aba$

# Properties of $A_n$



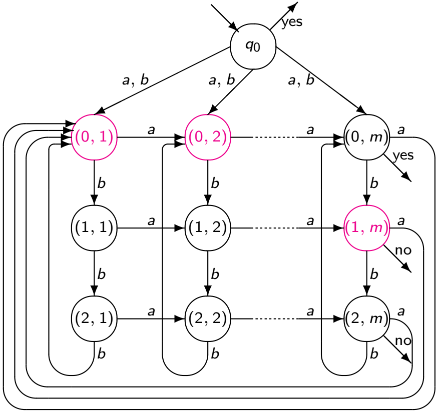

*aba*



yes
$q_0$
a, b
a, b
a, b
(0, 1)
a
(0, 2)
a
(0, m)
a
yes
b
b
b
(1, 1)
a
(1, 2)
a
(1, m)
a
no
b
b
b
(2, 1)
a
(2, 2)
a
(2, m)
a
no
b
b
b

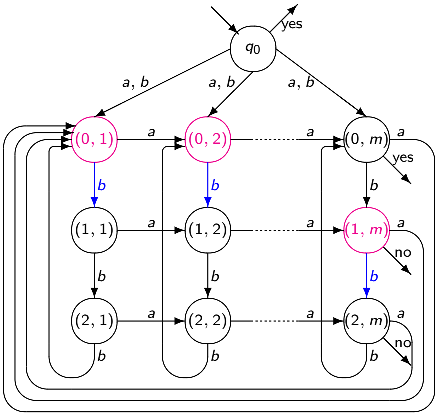

$abaa$

# Properties of $A_n$



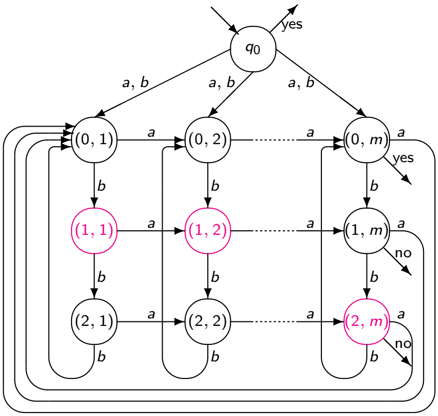

*abaa*

# Properties of $A_n$



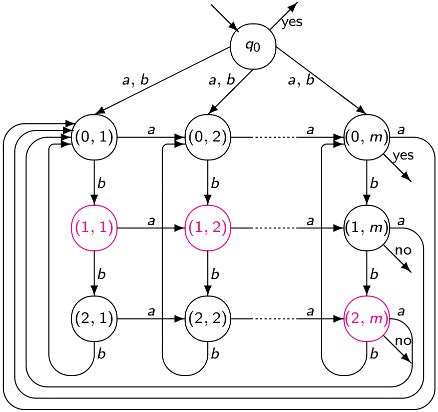

$abaab$

## Properties of $A_n$



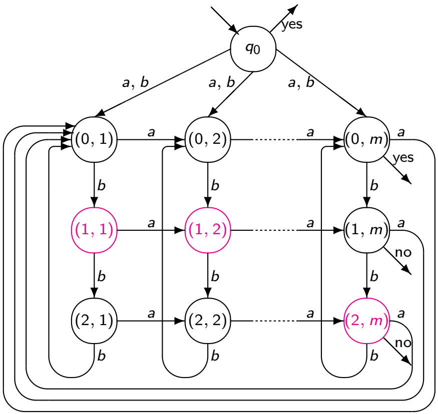

*abaab*

# Properties of $A_n$




The reachable states of the subset automaton $A_{sub}$ are:

- $\{q_0\}$
- the $3^m$ subsets obtained by taking one state from each column in the "grid part" (hence $A_n$ is an svfa!)




- We can verify that each two states of $A_{sub}$ are distinguishable

# Properties of $A_n$

Summing up:

- The subset automaton $A_{sub}$ has exactly $g(n) = 1 + 3^{\frac{n-1}{3}}$ states
- All these states are pairwise distinguishable
- Hence, it is the minimal dfa equivalent to $A_n$
-

Hence:

the *exact cost* for the conversion of $n$-state svfa's into equivalent dfa's is:

$$g(n) = \begin{cases} 1 + 3^{\frac{n-1}{3}} & \text{if } n \equiv 1 \pmod 3 \text{ and } n \geq 4 \\ 1 + 4 \cdot 3^{\frac{n-2}{3}-1} & \text{if } n \equiv 2 \pmod 3 \text{ and } n \geq 5 \\ 1 + 2 \cdot 3^{\frac{n}{3}-1} & \text{if } n \equiv 0 \pmod 3 \text{ and } n \geq 3 \\ n & \text{if } n \leq 2 \end{cases}$$

# Properties of $A_n$

Summing up:

- The subset automaton $A_{sub}$ has exactly $g(n) = 1 + 3^{\frac{n-1}{3}}$ states
- All these states are pairwise distinguishable
- Hence, it is the minimal dfa equivalent to $A_n$
- The argument can be easily adapted, for the values of $n$ which are not of the form $3m+1$

Hence:

the *exact cost* for the conversion of $n$-state svfa's into equivalent dfa's is:

$$g(n) = \begin{cases} 1 + 3^{\frac{n-1}{3}} & \text{if } n \equiv 1 \pmod 3 \text{ and } n \geqslant 4 \\ 1 + 4 \cdot 3^{\frac{n-2}{3}-1} & \text{if } n \equiv 2 \pmod 3 \text{ and } n \geqslant 5 \\ 1 + 2 \cdot 3^{\frac{n}{3}-1} & \text{if } n \equiv 0 \pmod 3 \text{ and } n \geqslant 3 \\ n & \text{if } n \leqslant 2 \end{cases}$$

# Properties of $A_n$

Summing up:

- The subset automaton $A_{sub}$ has exactly $g(n) = 1 + 3^{\frac{n-1}{3}}$ states
- All these states are pairwise distinguishable
- Hence, it is the minimal dfa equivalent to $A_n$
- The argument can be easily adapted, for the values of $n$ which are not of the form $3m + 1$

Hence:

the *exact cost* for the conversion of $n$-state svfa's into equivalent dfa's is:

$$g(n) = \begin{cases} 1 + 3^{\frac{n-1}{3}} & \text{if } n \equiv 1 \pmod 3 \text{ and } n \geqslant 4 \\ 1 + 4 \cdot 3^{\frac{n-2}{3}-1} & \text{if } n \equiv 2 \pmod 3 \text{ and } n \geqslant 5 \\ 1 + 2 \cdot 3^{\frac{n}{3}-1} & \text{if } n \equiv 0 \pmod 3 \text{ and } n \geqslant 3 \\ n & \text{if } n \leqslant 2 \end{cases}$$

# Properties of $A_n$

Summing up:

- The subset automaton $A_{sub}$ has exactly $g(n) = 1 + 3^{\frac{n-1}{3}}$ states
- All these states are pairwise distinguishable
- Hence, it is the minimal dfa equivalent to $A_n$
- The argument can be easily adapted, for the values of $n$ which are not of the form $3m + 1$

Hence:

the *exact cost* for the conversion of $n$-state svfa's into equivalent dfa's is:

$$g(n) = \begin{cases} 1 + 3^{\frac{n-1}{3}} & \text{if } n \equiv 1 \pmod 3 \text{ and } n \geqslant 4 \\ 1 + 4 \cdot 3^{\frac{n-2}{3}-1} & \text{if } n \equiv 2 \pmod 3 \text{ and } n \geqslant 5 \\ 1 + 2 \cdot 3^{\frac{n}{3}-1} & \text{if } n \equiv 0 \pmod 3 \text{ and } n \geqslant 3 \\ n & \text{if } n \leqslant 2 \end{cases}$$

# Properties of $A_n$

Summing up:

- The subset automaton $A_{sub}$ has exactly $g(n) = 1 + 3^{\frac{n-1}{3}}$ states
- All these states are pairwise distinguishable
- Hence, it is the minimal dfa equivalent to $A_n$
- The argument can be easily adapted, for the values of $n$ which are not of the form $3m + 1$

Hence:

the *exact cost* for the conversion of $n$-state svfa's into equivalent dfa's is:

$$g(n) = \begin{cases} 1 + 3^{\frac{n-1}{3}} & \text{if } n \equiv 1 \pmod 3 \text{ and } n \geqslant 4 \\ 1 + 4 \cdot 3^{\frac{n-2}{3}-1} & \text{if } n \equiv 2 \pmod 3 \text{ and } n \geqslant 5 \\ 1 + 2 \cdot 3^{\frac{n}{3}-1} & \text{if } n \equiv 0 \pmod 3 \text{ and } n \geqslant 3 \\ n & \text{if } n \leqslant 2 \end{cases}$$

# Properties of $A_n$

Summing up:

- The subset automaton $A_{sub}$ has exactly $g(n) = 1 + 3^{\frac{n-1}{3}}$ states
- All these states are pairwise distinguishable
- Hence, it is the minimal dfa equivalent to $A_n$
- The argument can be easily adapted, for the values of $n$ which are not of the form $3m + 1$

Hence:

the *exact cost* for the conversion of $n$-state svfa's into equivalent dfa's is:

$$g(n) = \begin{cases} 1 + 3^{\frac{n-1}{3}} & \text{if } n \equiv 1 \pmod 3 \text{ and } n \geqslant 4 \\ 1 + 4 \cdot 3^{\frac{n-2}{3}-1} & \text{if } n \equiv 2 \pmod 3 \text{ and } n \geqslant 5 \\ 1 + 2 \cdot 3^{\frac{n}{3}-1} & \text{if } n \equiv 0 \pmod 3 \text{ and } n \geqslant 3 \\ n & \text{if } n \leqslant 2 \end{cases}$$

# Svfa's with multiple initial states

**What happens if we allow multiple initial states?**

- All the initial states of $A$ must be compatible each others
- The initial state of the minimal dfa is the maximal clique containing all of them
- This gives an upper bound $f(n) = g(n+1) - 1$
-

# Svfa's with multiple initial states

What happens if we allow multiple initial states?

- All the initial states of $A$ must be compatible each others
- The initial state of the minimal dfa is the maximal clique containing all of them
- This gives an upper bound $f(n) = g(n+1) - 1$
- The upper bound is optimal

What happens if we allow multiple initial states?

- All the initial states of $A$ must be compatible each others
- The initial state of the minimal dfa is the maximal clique containing all of them
- This gives an upper bound $f(n) = g(n+1) - 1$
- The upper bound is optimal

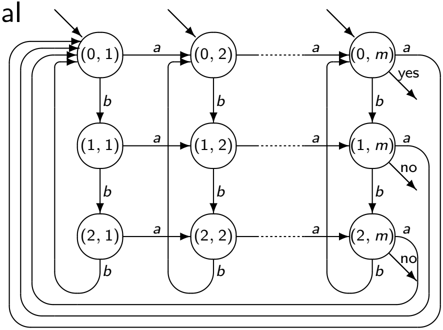
# Svfa's with multiple initial states

What happens if we allow multiple initial states?

- All the initial states of $A$ must be compatible each others
- The initial state of the minimal dfa is the maximal clique containing all of them
- This gives an upper bound $f(n) = g(n+1) - 1$
- The upper bound is optimal

What happens if we allow multiple initial states?

- All the initial states of $A$ must be compatible each others
- The initial state of the minimal dfa is the maximal clique containing all of them
- This gives an upper bound $f(n) = g(n+1) - 1$
- The upper bound is optimal



# The unary case

**What about the optimality in the unary case?**

- We proved the optimality using automata over a binary alphabet
- The cost of the conversion of unary nfa's into dfa's is $F(n) = e^{O(\sqrt{n \log n})}$ (Chrobak, 1986)
- $F(n)$ grows more slowly than $g(n)$.
  Hence $F(n)$ is a better upper bound for the conversion of svfa's into dfa's in the unary case

# The unary case

What about the optimality in the unary case?

- We proved the optimality using automata over a binary alphabet
- The cost of the conversion of unary nfa's into dfa's is $F(n) = e^{O(\sqrt{n \log n})}$ (Chrobak, 1986)
- $F(n)$ grows more slowly than $g(n)$.
  Hence $F(n)$ is a better upper bound for the conversion of svfa's into dfa's in the unary case
- This upper bound is not optimal!
-

**What about the optimality in the unary case?**

- We proved the optimality using automata over a binary alphabet
- The cost of the conversion of unary nfa's into dfa's is $F(n) = \mathrm{e}^{O(\sqrt{n \log n})}$ (Chrobak, 1986)
- $F(n)$ grows more slowly than $g(n)$.
  Hence $F(n)$ is a better upper bound for the conversion of svfa's into dfa's in the unary case
- This upper bound is not optimal!
- In fact, if $L$ is a unary language accepted by a $n$-state unary nfa such that the minimal dfa for $L$ requires $F(n)$ states, then each nfa for $L^c$ requires $F(n)$ states (Mera, Pighizzini, 2005)

# The unary case

What about the optimality in the unary case?

- We proved the optimality using automata over a binary alphabet
- The cost of the conversion of unary nfa's into dfa's is $F(n) = \mathrm{e}^{O(\sqrt{n \log n})}$ (Chrobak, 1986)
- $F(n)$ grows more slowly than $g(n)$.
  Hence $F(n)$ is a better upper bound for the conversion of svfa's into dfa's in the unary case
- This upper bound is not optimal!
- In fact, if $L$ is a unary language accepted by a $n$-state unary nfa such that the minimal dfa for $L$ requires $F(n)$ states, then *each nfa* for $L^c$ requires $F(n)$ states (Mera, Pighizzini, 2005)

What about the optimality in the unary case?

- We proved the optimality using automata over a binary alphabet
- The cost of the conversion of unary nfa's into dfa's is $F(n) = \mathrm{e}^{O(\sqrt{n \log n})}$ (Chrobak, 1986)
- $F(n)$ grows more slowly than $g(n)$.
  Hence $F(n)$ is a better upper bound for the conversion of svfa's into dfa's in the unary case
- This upper bound is not optimal!
- In fact, if $L$ is a unary language accepted by a $n$-state unary nfa such that the minimal dfa for $L$ requires $F(n)$ states, then *each nfa* for $L^c$ requires $F(n)$ states (Mera, Pighizzini, 2005)

What about the optimality in the unary case?

- We proved the optimality using automata over a binary alphabet
- The cost of the conversion of unary nfa's into dfa's is $F(n) = e^{O(\sqrt{n \log n})}$ (Chrobak, 1986)
- $F(n)$ grows more slowly than $g(n)$.
  Hence $F(n)$ is a better upper bound for the conversion of svfa's into dfa's in the unary case
- This upper bound is not optimal!
- In fact, if $L$ is a unary language accepted by a $n$-state unary nfa such that the minimal dfa for $L$ requires $F(n)$ states, then *each nfa* for $L^c$ requires $F(n)$ states (Mera, Pighizzini, 2005)

# Conclusion

- Each $n$-state svfa can be converted into an equivalent dfa with $g(n)$ states:
  - We found the value of $g(n)$, which grows like $3^{\frac{n}{3}}$
  - *The bound is exact:*
    for each integer $n$, there exists an svfa $A_n$ with $n$ states and an input alphabet of two letters such that the minimal equivalent dfa has $g(n)$ states.

- Each $n$-state svfa with multiple initial states can be converted into an equivalent dfa with $f(n) = g(n+1) - 1$ states.
  *Also this bound is exact.*

- In the unary case, a better upper bound is given by the function $F(n) = e^{O(\sqrt{n \ln n})}$.
  *However, this upper bound is not optimal.*
  It is an open problem to find a better upper bound for the unary case.

# Conclusion

- Each $n$-state svfa can be converted into an equivalent dfa with $g(n)$ states:
  - We found the value of $g(n)$, which grows like $3^{\frac{n}{3}}$
  - *The bound is exact:*
    for each integer $n$, there exists an svfa $A_n$ with $n$ states and an input alphabet of two letters such that the minimal equivalent dfa has $g(n)$ states.

- Each $n$-state svfa with multiple initial states can be converted into an equivalent dfa with $f(n) = g(n+1) - 1$ states.
  *Also this bound is exact.*

- In the unary case, a better upper bound is given by the function $F(n) = e^{O(\sqrt{n \log n})}$.
  *However, this upper bound is not optimal.*
  It is an open problem to find a better upper bound in the unary case.

# Conclusion

- Each $n$-state svfa can be converted into an equivalent dfa with $g(n)$ states:
  - We found the value of $g(n)$, which grows like $3^{\frac{n}{3}}$
  - *The bound is exact:*
    for each integer $n$, there exists an svfa $A_n$ with $n$ states and an input alphabet of two letters such that the minimal equivalent dfa has $g(n)$ states.
- Each $n$-state svfa with multiple initial states can be converted into an equivalent dfa with $f(n) = g(n+1) - 1$ states.
  *Also this bound is exact.*
- In the unary case, a better upper bound is given by the function $F(n) = e^{O(\sqrt{n \log n})}$.
  *However, this upper bound is not optimal.*
  It is an open problem to find a better upper bound in the unary case.

# Conclusion

- Each $n$-state svfa can be converted into an equivalent dfa with $g(n)$ states:
  - We found the value of $g(n)$, which grows like $3^{\frac{n}{3}}$
  - *The bound is exact:*
    for each integer $n$, there exists an svfa $A_n$ with $n$ states and an input alphabet of two letters such that the minimal equivalent dfa has $g(n)$ states.

- Each $n$-state svfa with multiple initial states can be converted into an equivalent dfa with $f(n) = g(n+1) - 1$ states.
  *Also this bound is exact.*

- In the unary case, a better upper bound is given by the function $F(n) = \mathrm{e}^{O(\sqrt{n \log n})}$.
  *However, this upper bound is not optimal.*
  It is an open problem to find a better upper bound in the unary case.