Non-Self-Embedding Grammars and Descriptional Complexity

NCMA 2017

Giovanni Pighizzini **Luca Prigioniero**

Università degli Studi di Milano

August 17

UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA

Non-Self-Embedding Grammars: Recalling the Definition

- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{*}{\Longrightarrow} \alpha A \beta$
- α, β:
- NSE grammars only generate regular languages [Chomsky, 1959a, Chomsky, 1959b, Anselmo et al., 2002]

Non-Self-Embedding Grammars: Recalling the Definition

- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β:
- NSE grammars only generate regular languages
  [Chomsky, 1959a, Chomsky, 1959b, Anselmo et al., 2002]

Non-Self-Embedding Grammars: Recalling the Definition

- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β: both are non-empty vs. at least one is empty for each derivation
- NSE grammars only generate regular languages

[Chomsky, 1959a, Chomsky, 1959b, Anselmo et al., 2002]

Non-Self-Embedding Grammars: Recalling the Definition

- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β: both are non-empty vs. at least one is empty for each derivation
- NSE grammars only generate regular languages

[Chomsky, 1959a, Chomsky, 1959b, Anselmo et al., 2002]

Non-Self-Embedding Grammars: Recalling the Definition

- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β: both are non-empty vs. at least one is empty for each derivation
- NSE grammars only generate regular languages

[Chomsky, 1959a, Chomsky, 1959b, Anselmo et al., 2002]

Non-Self-Embedding Grammars: Recalling the Definition

- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β: both are non-empty vs. at least one is empty for each derivation
- NSE grammars only generate regular languages

[Chomsky, 1959a, Chomsky, 1959b, Anselmo et al., 2002]

Non-Self-Embedding Grammars: Recalling the Definition

- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β: both are non-empty vs. at least one is empty for each derivation
- NSE grammars only generate regular languages

[Chomsky, 1959a, Chomsky, 1959b, Anselmo et al., 2002]

Non-Self-Embedding Grammars: Recalling the Definition

- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β: both are non-empty vs. at least one is empty for each derivation
- NSE grammars only generate regular languages
  [Chomsky, 1959a, Chomsky, 1959b, Anselmo et al., 2002]

Non-Self-Embedding Grammars to Automata

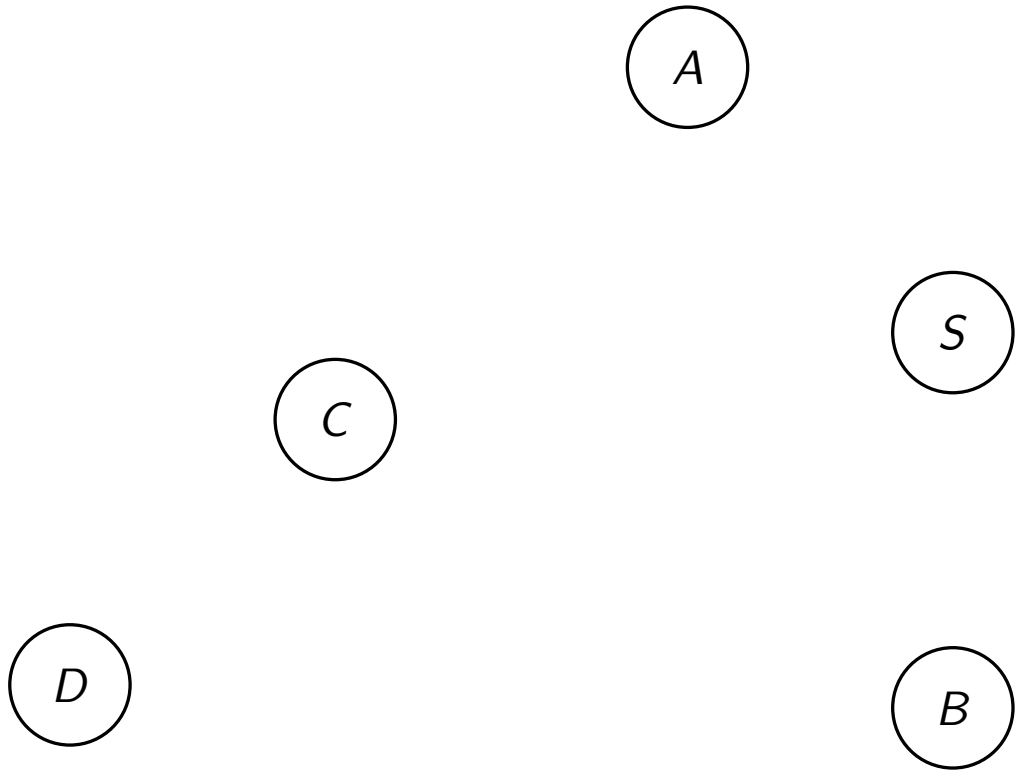
Step I: The Production Transition Graph

$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$

Non-Self-Embedding Grammars to Automata

Step I: The Production Transition Graph

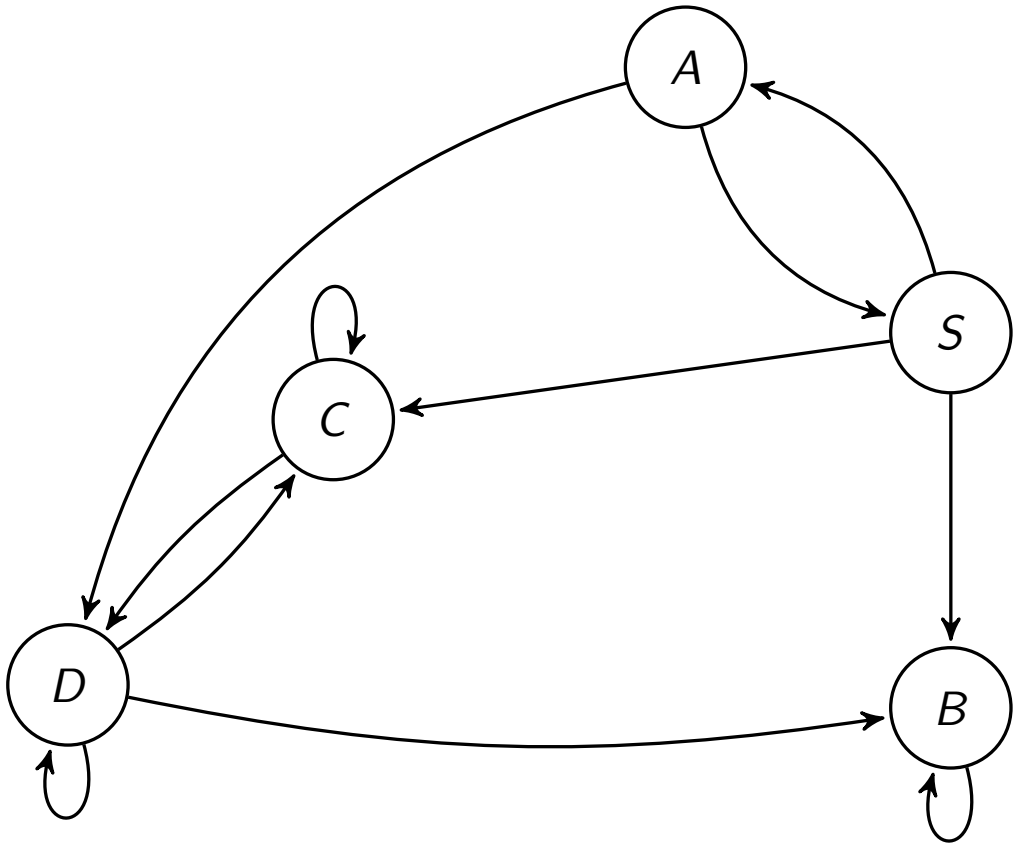
$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step I: The Production Transition Graph

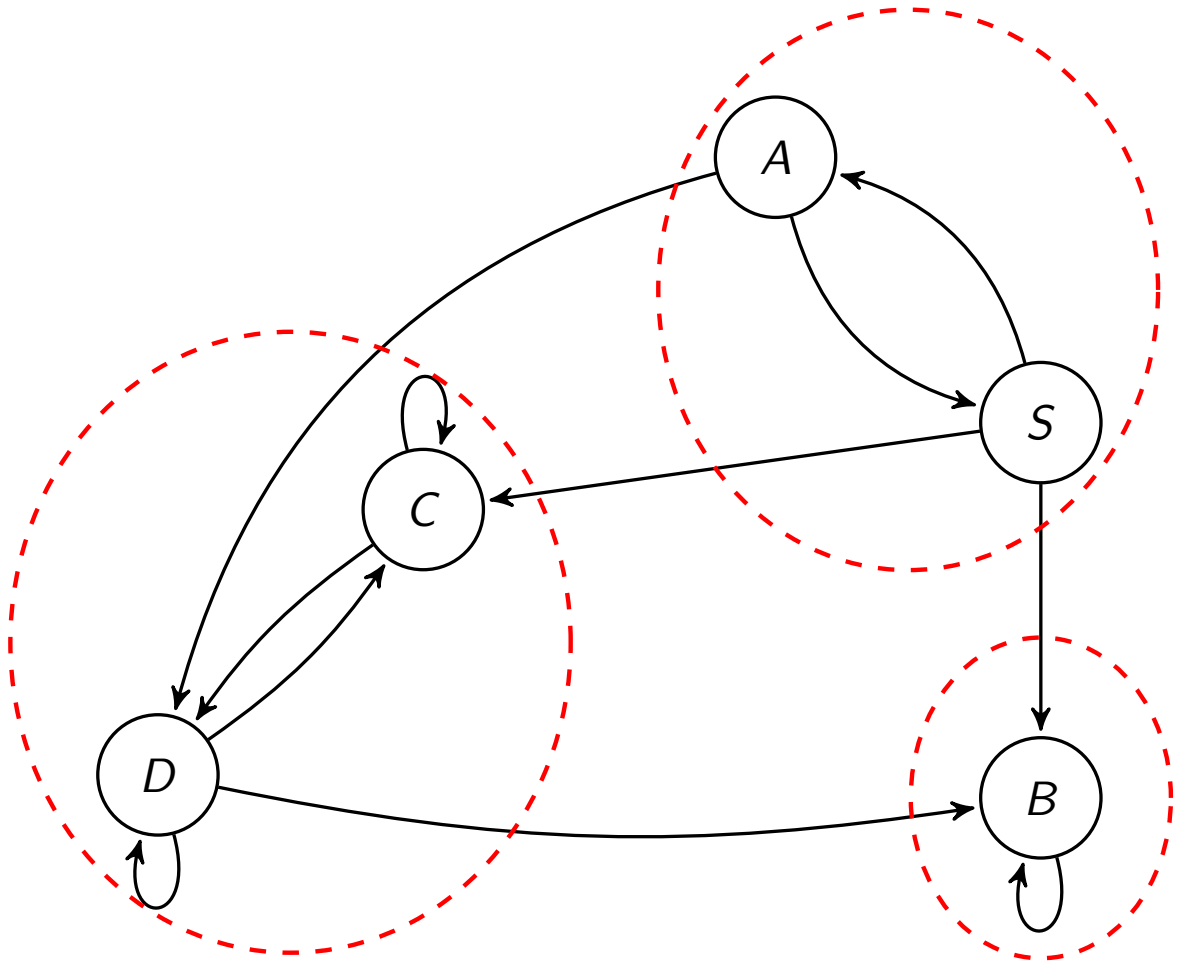
$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step I: The Production Transition Graph

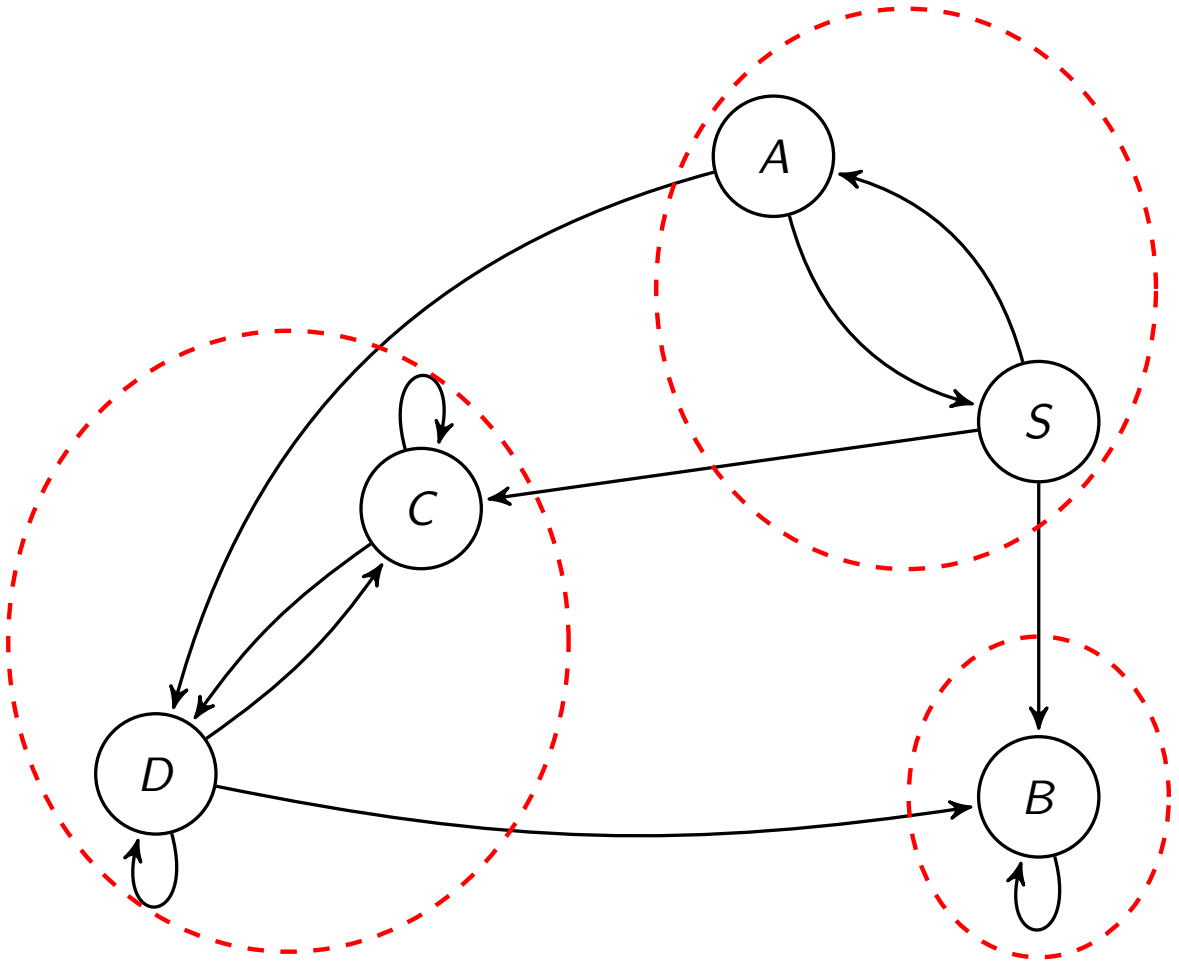
$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

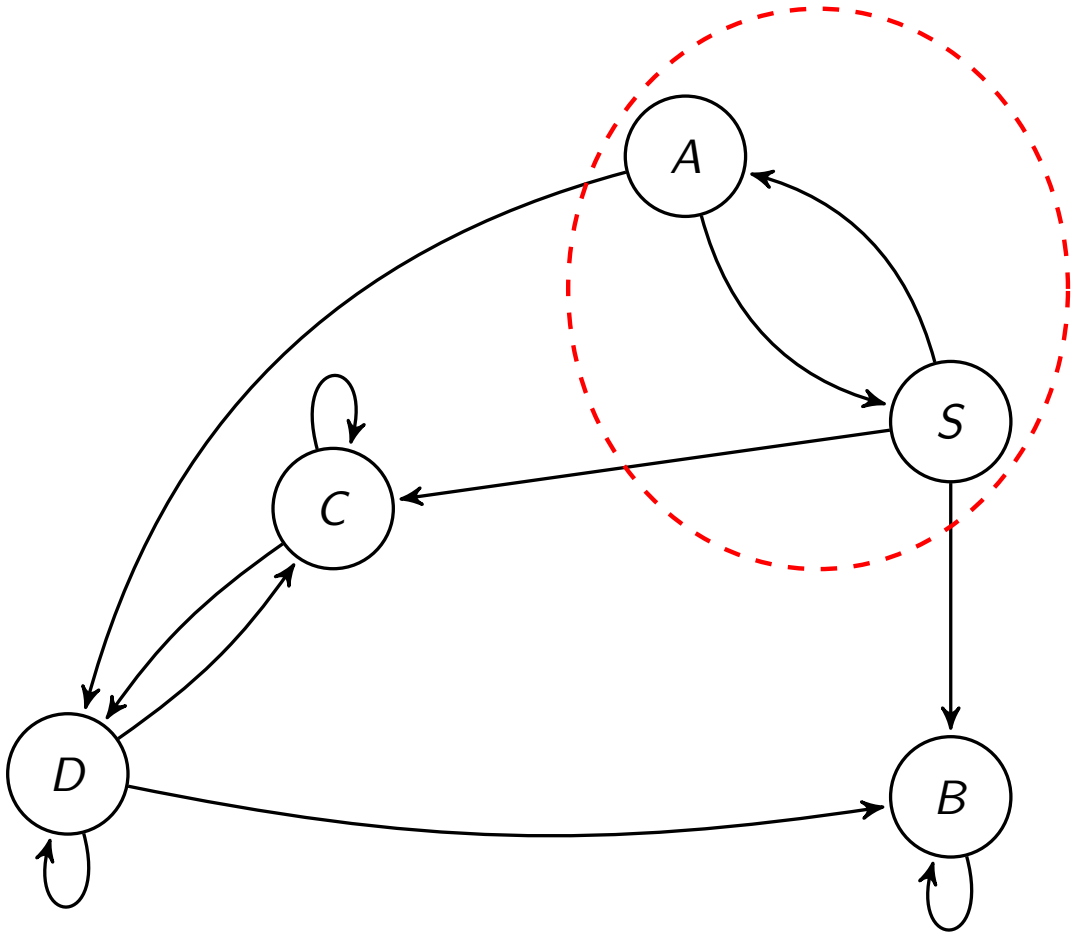
$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

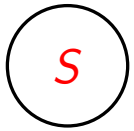
$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

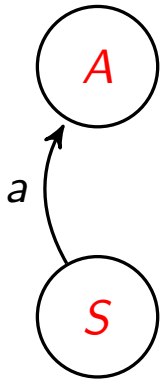
$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

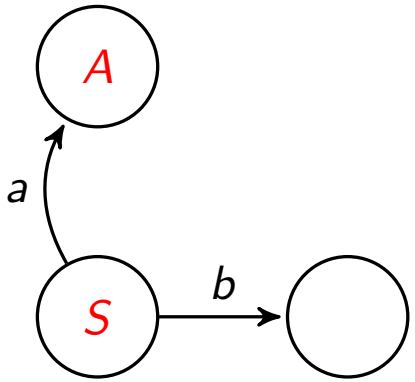
$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

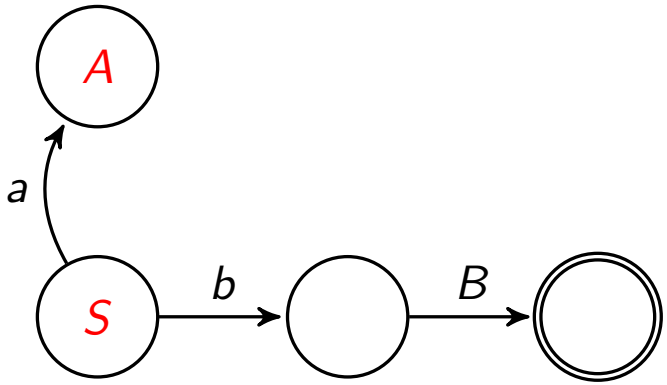
$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

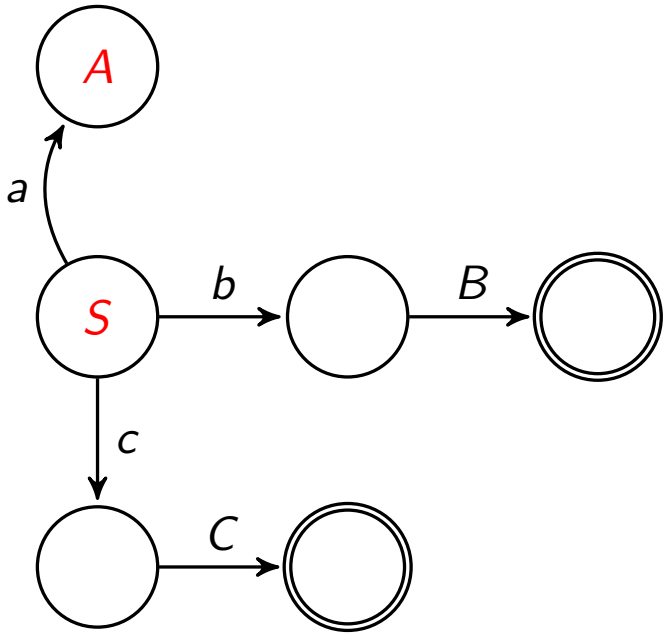
$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

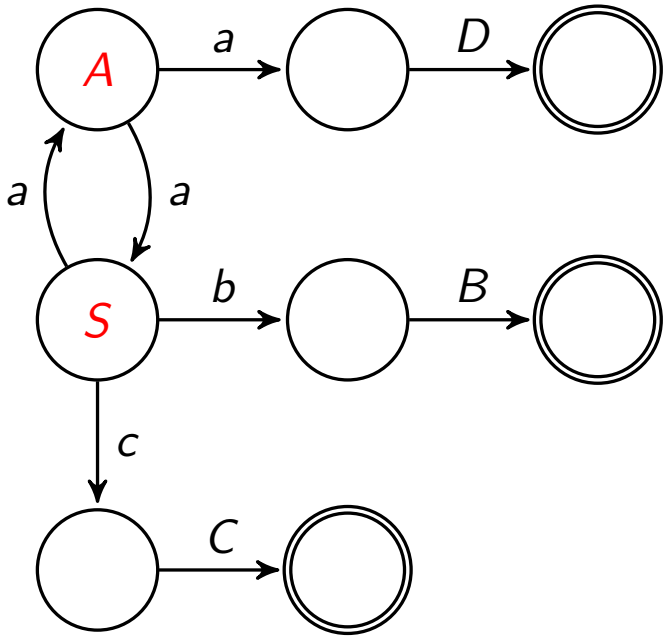
$S \to aA \mid bB \mid cC$

$A \to aS \mid aD$

$B \to bB \mid b$

$C \to cC \mid dD$

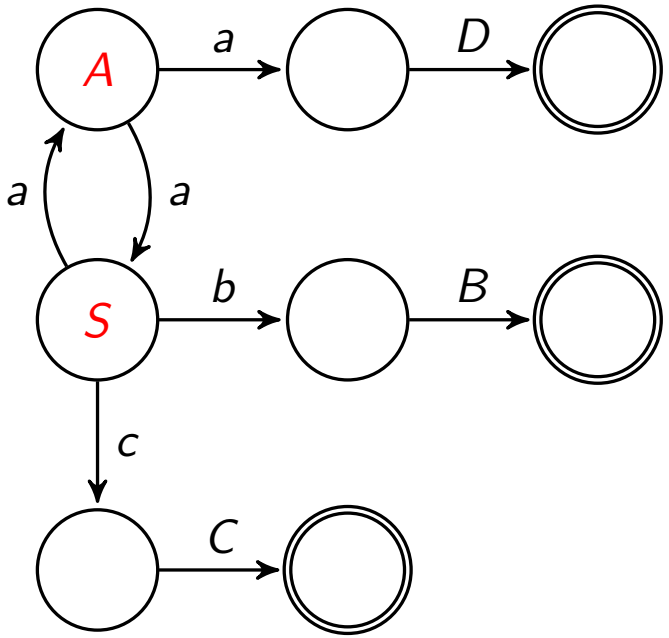
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

$S \to aA \mid bB \mid cC$
$A \to aS \mid aD$
$B \to bB \mid b$
$C \to cC \mid dD$
$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

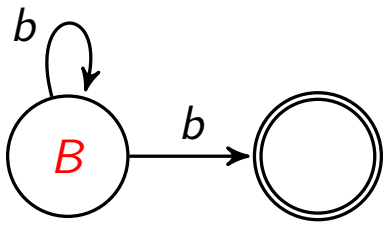
$S \to aA \mid bB \mid cC$

$A \to aS \mid aD$

$B \to bB \mid b$

$C \to cC \mid dD$

$D \to dC \mid dD \mid dB$



Non-Self-Embedding Grammars to Automata

Step II: Converting SCCs

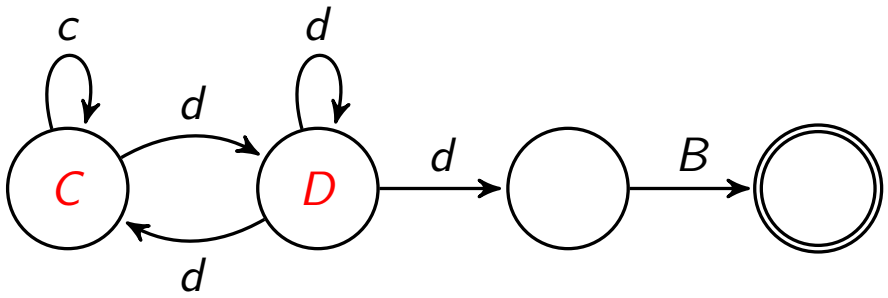
$S \to aA \mid bB \mid cC$

$A \to aS \mid aD$

$B \to bB \mid b$

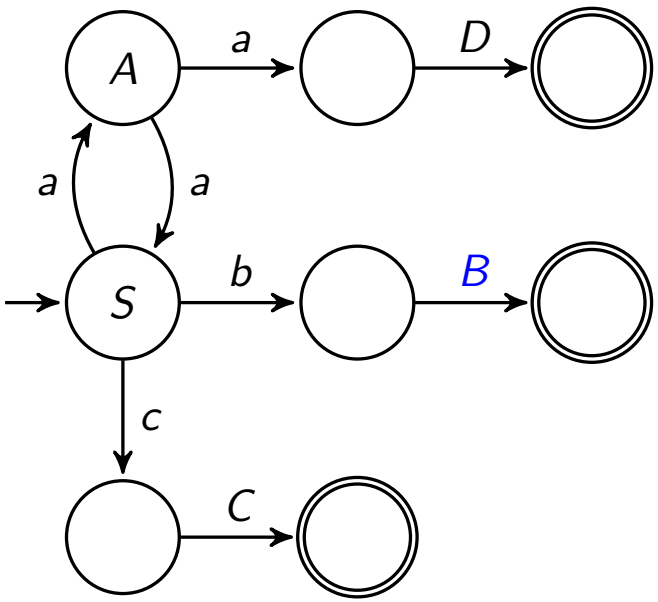
$C \to cC \mid dD$

$D \to dC \mid dD \mid dB$



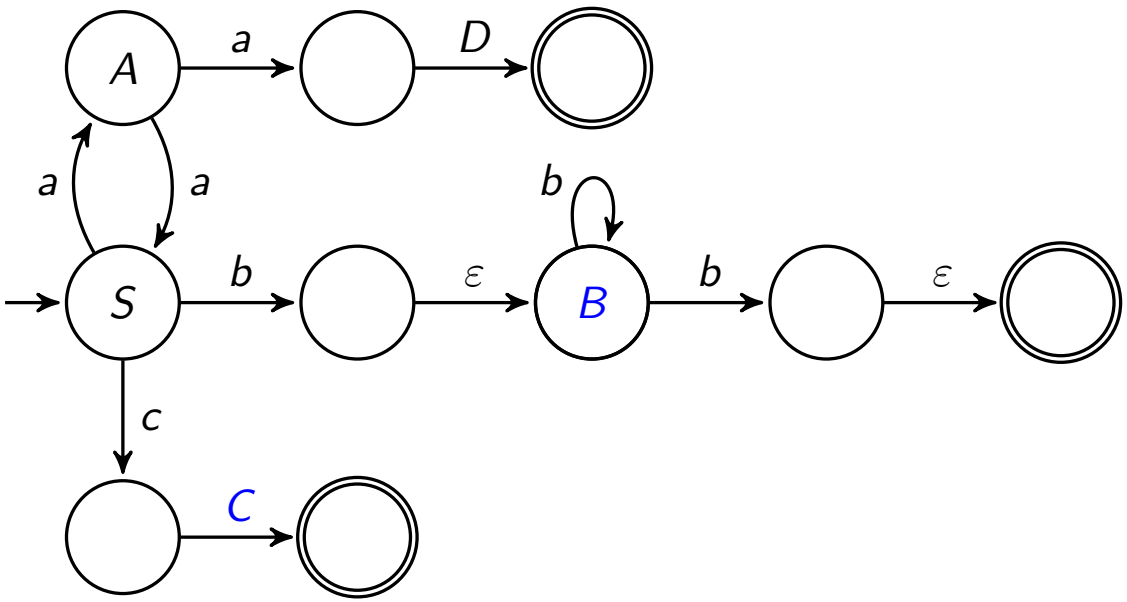
Non-Self-Embedding Grammars to Automata

Step III: Replacing Variables



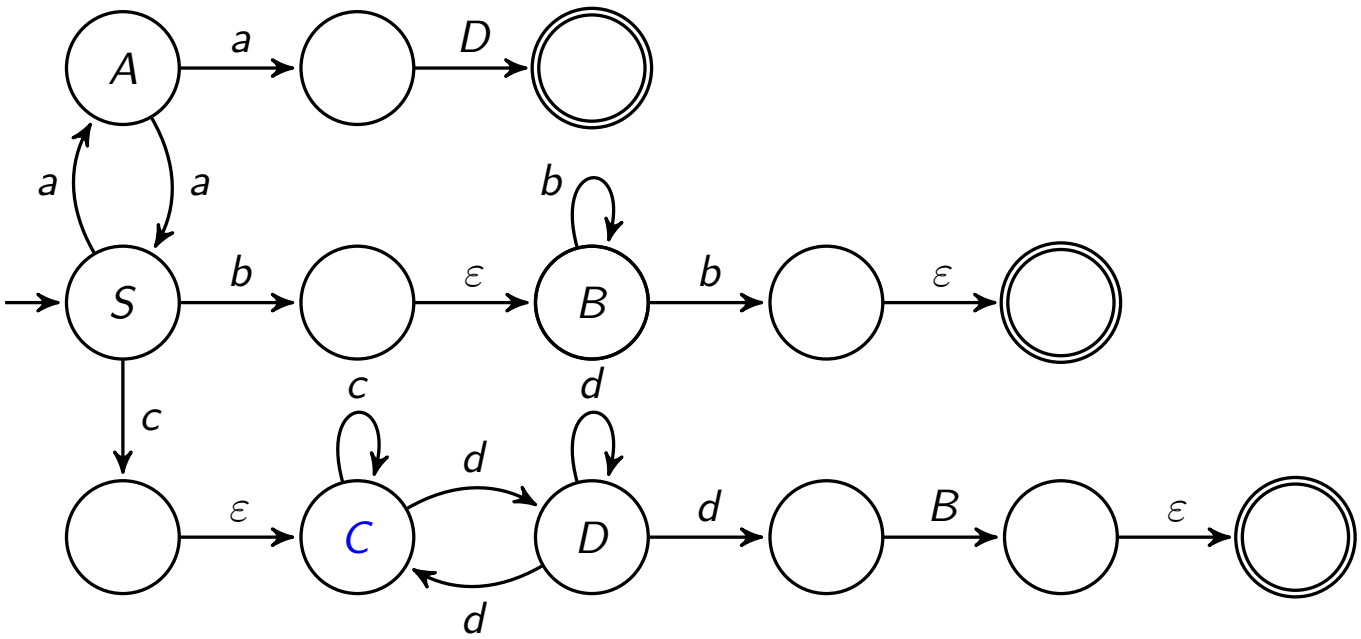
Non-Self-Embedding Grammars to Automata

Step III: Replacing Variables



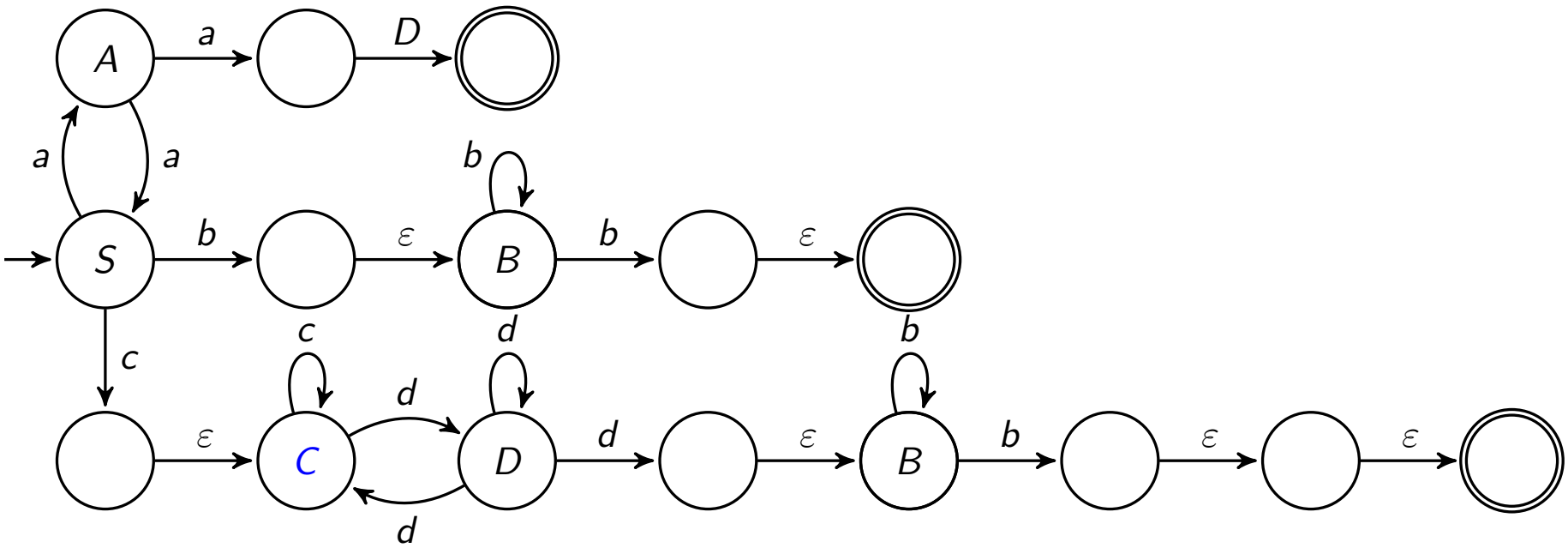
Non-Self-Embedding Grammars to Automata

Step III: Replacing Variables



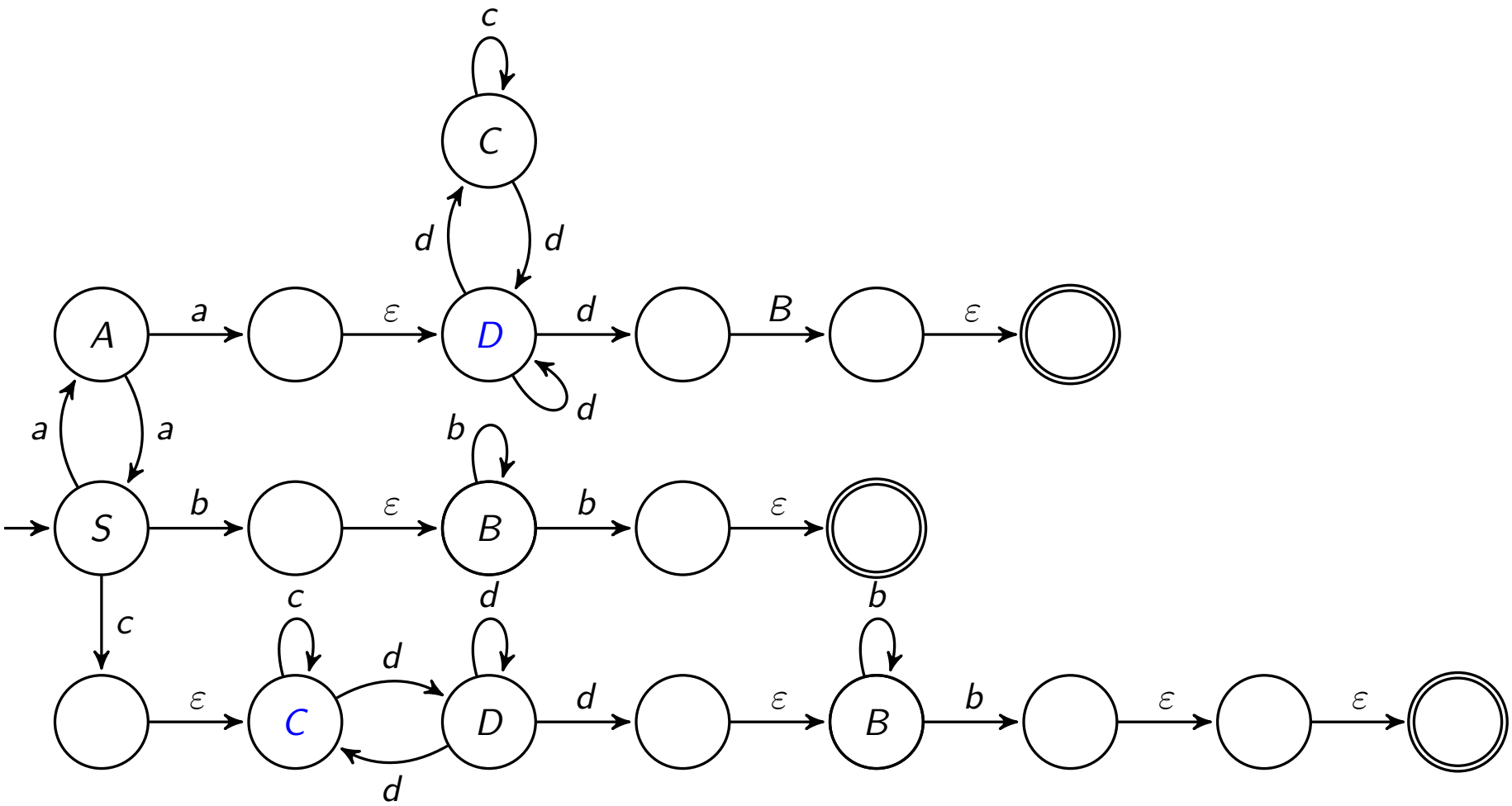
Non-Self-Embedding Grammars to Automata

Step III: Replacing Variables



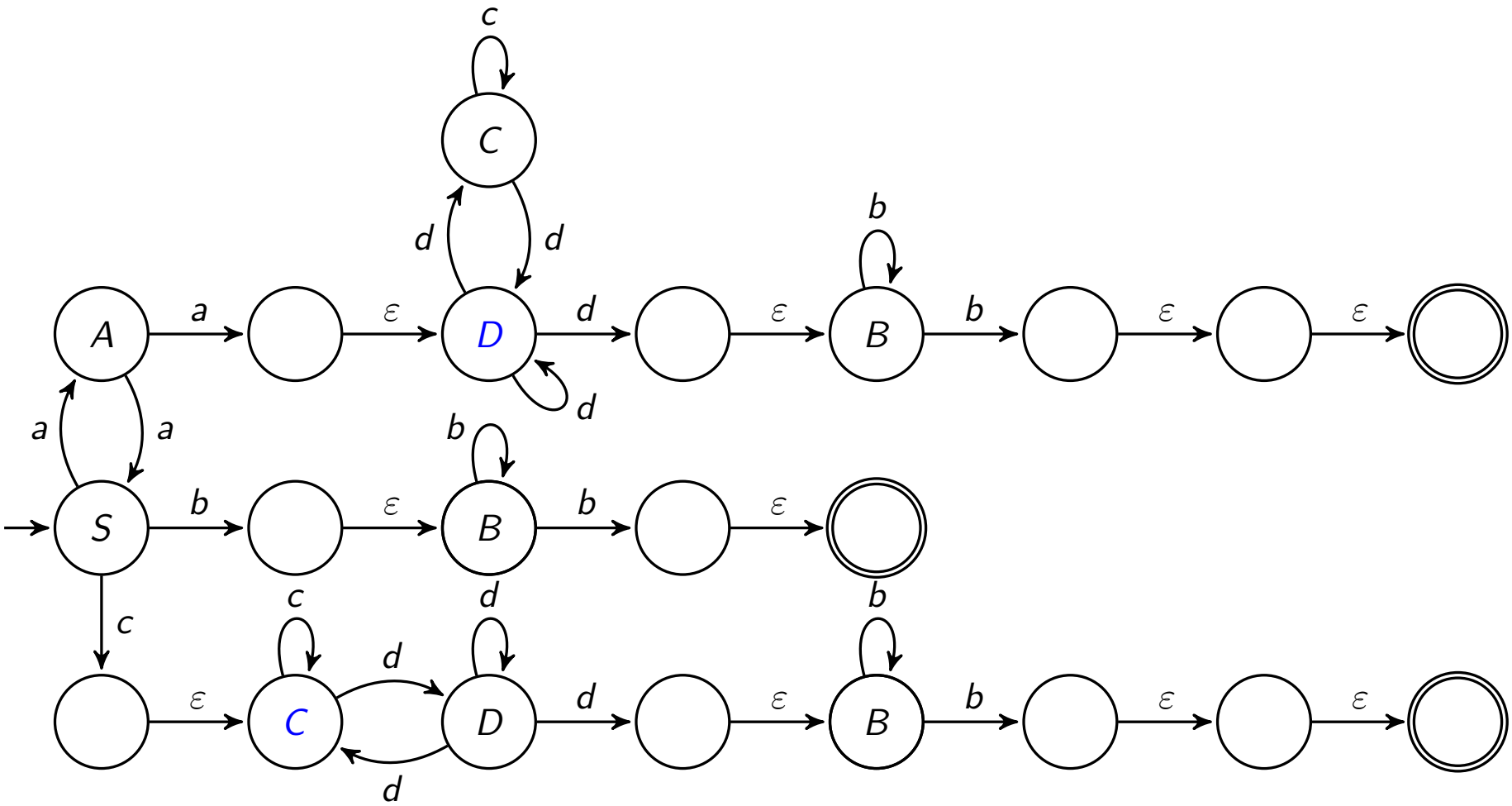
Non-Self-Embedding Grammars to Automata

Step III: Replacing Variables



Non-Self-Embedding Grammars to Automata

Step III: Replacing Variables



Non-Self-Embedding Grammars to Automata

Lower Bound for the Conversion

[Anselmo et al., 2002]

The size of the automaton accepting a regular language L is at least exponential with respect to a NSE grammar generating L.

$$A_n \to A_{n-1}A_{n-1}$$
$$A_{n-1} \to A_{n-2}A_{n-2}$$
$$\vdots$$
$$A_2 \to A_1 A_1$$
$$A_1 \to A_0 A_0$$
$$A_0 \to a$$

$$L = \{a^{2^n}\}$$

Non-Self-Embedding Grammars to Automata

Lower Bound for the Conversion

[Anselmo et al., 2002]

The size of the automaton accepting a regular language L is at least exponential with respect to a NSE grammar generating L.

$$
\begin{aligned}
A_n &\to A_{n-1}A_{n-1} \\
A_{n-1} &\to A_{n-2}A_{n-2} \\
&\vdots \\
A_2 &\to A_1 A_1 \\
A_1 &\to A_0 A_0 \\
A_0 &\to a
\end{aligned}
$$

$L = \{a^{2^n}\}$

Non-Self-Embedding Grammars to Automata

Lower Bound for the Conversion

[Anselmo et al., 2002]

The size of the automaton accepting a regular language L is at least exponential with respect to a NSE grammar generating L.

$$
\begin{aligned}
&A_n \to A_{n-1}A_{n-1}\\
&A_{n-1} \to A_{n-2}A_{n-2}\\
&\vdots\\
&A_2 \to A_1A_1\\
&A_1 \to A_0A_0\\
&A_0 \to a
\end{aligned}
$$

$$L = \{a^{2^n}\}$$

Non-Self-Embedding Grammars to Automata

Upper Bound for the Conversion

- Grammar G of size $s = s_1 + s_2 + \ldots + s_n$ (Step I)
- Each NFA M_i (Step II) has size $O(s_i)$
- The substitution of the automaton M_i in M_j (Step III) has size $O(s_i s_j)$

$$s_1 \cdots s_n = 2^{\log(s_1 \cdots s_n)} = 2^{\log s_1 + \cdots + \log s_n}$$

- This leads to a 2^{2^s}-state DFA

Non-Self-Embedding Grammars to Automata

Upper Bound for the Conversion

- Grammar G of size $s = s_1 + s_2 + \ldots + s_n$ (Step I)
- Each NFA M_i (Step II) has size $O(s_i)$
- The substitution of the automaton M_i in M_j (Step III) has size $O(s_i s_j)$

$$s_1 \cdots s_n = 2^{\log(s_1 \cdots s_n)} = 2^{\log s_1 + \cdots + \log s_n}$$

- This leads to a 2^{2^s}-state DFA

Non-Self-Embedding Grammars to Automata

Upper Bound for the Conversion

- Grammar G of size $s = s_1 + s_2 + \ldots + s_n$ (Step I)
- Each NFA M_i (Step II) has size $O(s_i)$
- The substitution of the automaton M_i in M_j (Step III) has size $O(s_i s_j)$

$$s_1 \cdots s_n = 2^{\log(s_1 \cdots s_n)} = 2^{\log s_1 + \cdots + \log s_n}$$

- This leads to a 2^{2^s}-state DFA

Non-Self-Embedding Grammars to Automata

Upper Bound for the Conversion

- Grammar G of size $s = s_1 + s_2 + \ldots + s_n$ (Step I)
- Each NFA M_i (Step II) has size $O(s_i)$
- The substitution of the automaton M_i in M_j (Step III) has size $O(s_i s_j)$

$$s_1 \cdots s_n = 2^{\log(s_1 \cdots s_n)} = 2^{\log s_1 + \cdots + \log s_n} \leq 2^s$$

- This leads to a 2^{2^s}-state DFA

Non-Self-Embedding Grammars to Automata

Upper Bound for the Conversion

- Grammar G of size $s = s_1 + s_2 + \ldots + s_n$ (Step I)
- Each NFA M_i (Step II) has size $O(s_i)$
- The substitution of the automaton M_i in M_j (Step III) has size $O(s_i s_j)$

$$s_1 \cdots s_n = 2^{\log(s_1 \cdots s_n)} = 2^{\log s_1 + \cdots + \log s_n} \leq 2^s$$

- This leads to a 2^{2^s}-state DFA

Non-Self-Embedding Grammars to Automata

Upper Bound for the Conversion

- Grammar G of size $s = s_1 + s_2 + \ldots + s_n$ (Step I)
- Each NFA M_i (Step II) has size $O(s_i)$
- The substitution of the automaton M_i in M_j (Step III) has size $O(s_i s_j)$

$$s_1 \cdots s_n = 2^{\log(s_1 \cdots s_n)} = 2^{\log s_1 + \cdots + \log s_n} \leq 2^s$$

- This leads to a 2^{2^s}-state DFA

Non-Self-Embedding Grammars to Automata

Upper Bound for the Conversion

- Grammar G of size $s = s_1 + s_2 + \ldots + s_n$ (Step I)
- Each NFA M_i (Step II) has size $O(s_i)$
- The substitution of the automaton M_i in M_j (Step III) has size $O(s_i s_j)$

$$s_1 \cdots s_n = 2^{\log(s_1 \cdots s_n)} = 2^{\log s_1 + \cdots + \log s_n} \leq 2^s$$

- This leads to a 2^{2^s}-state DFA

Tightness: the Witness Language

$$L_n = a_1 \quad a_2 \quad \cdots \quad a_n \quad \cdots \; a_{j+1} \; a_{j+2} \; \cdots \; a_{j+n} \; \cdots \quad a_{\ldots} \quad \cdots a_{kn-1} \; a_{kn}$$

$$L_n = \{x_1 \cdots x_k \quad | \quad x_i \in \{a,b\}^n, \; k > 1, \; i = 1, \ldots, k, \; \exists j, 1 \leq j < k, \text{ s.t. } x_j = x_k^R\}$$

The grammar generating L_n has size $O(n)$

Tightness: the Witness Language

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\ldots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

$$L_n = \{x_1 \cdots x_k \quad | \quad x_i \in \{a,b\}^n, \\ k > 1,\ i = 1, \ldots, k, \\ \exists j, 1 \leq j < k,\ \text{s.t. } x_j = x_k^R\}$$

The grammar generating L_n has size $O(n)$

Tightness: the Witness Language

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\ldots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

$$L_n = \{x_1 \cdots x_k \quad | \quad x_i \in \{a,b\}^n, \\ k > 1,\ i = 1, \ldots, k, \\ \exists j, 1 \leq j < k,\ \text{s.t. } x_j = x_k^R\}$$

The grammar generating L_n has size $O(n)$

Tightness: the Witness Language

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\ldots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

$$L_n = \{x_1 \cdots x_k \quad | \quad x_i \in \{a,b\}^n, \\ k > 1,\ i = 1, \ldots, k, \\ \exists j, 1 \leq j < k,\ \text{s.t. } x_j = x_k^R\}$$

The grammar generating L_n has size $O(n)$

Recognizing L_n

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots\ a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- With an NFA:
  1. Guess which is the block x_j
  2. Store x_j
  3. Guess which is the last block
  4. Verify $x_k^R = x_j$
- $O(2^n)$ states
- By a fooling set argument, any equivalent DFA requires 2^{2^n} states

Recognizing L_n

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- With an NFA:
  1. Guess which is the block x_j
  2. Store x_j
  3. Guess which is the last block
  4. Verify $x_k^R = x_j$
- $O(2^n)$ states
- By a fooling set argument, any equivalent DFA requires 2^{2^n} states

Recognizing L_n

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- With an NFA:
  1. Guess which is the block x_j
  2. Store x_j
  3. Guess which is the last block
  4. Verify $x_k^R = x_j$
- $O(2^n)$ states
- By a fooling set argument, any equivalent DFA requires 2^{2^n} states

Recognizing L_n

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots\ a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- With an NFA:
  1. Guess which is the block x_j
  2. Store x_j
  3. Guess which is the last block
  4. Verify $x_k^R = x_j$
- $O(2^n)$ states
- By a fooling set argument, any equivalent DFA requires 2^{2^n} states

Recognizing L_n

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots\ a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- With an NFA:
  1. Guess which is the block x_j
  2. Store x_j
  3. Guess which is the last block
  4. Verify $x_k^R = x_j$
- $O(2^n)$ states
- By a fooling set argument, any equivalent DFA requires 2^{2^n} states

Recognizing L_n

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- With an NFA:
  1. Guess which is the block x_j
  2. Store x_j
  3. Guess which is the last block
  4. Verify $x_k^R = x_j$
- $O(2^n)$ states
- By a fooling set argument, any equivalent DFA requires 2^{2^n} states

Recognizing L_n

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- With an NFA:
  1. Guess which is the block x_j
  2. Store x_j
  3. Guess which is the last block
  4. Verify $x_k^R = x_j$
- $O(2^n)$ states
- By a fooling set argument, any equivalent DFA requires 2^{2^n} states

Recognizing L_n

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- With an NFA:
  1. Guess which is the block x_j
  2. Store x_j
  3. Guess which is the last block
  4. Verify $x_k^R = x_j$
- $O(2^n)$ states
- By a fooling set argument, any equivalent DFA requires 2^{2^n} states

Recognizing L_n

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- With an NFA:
  1. Guess which is the block x_j
  2. Store x_j
  3. Guess which is the last block
  4. Verify $x_k^R = x_j$
- $O(2^n)$ states
- By a fooling set argument, any equivalent DFA requires 2^{2^n} states

The double exponential bound NSE $\rightarrow$ DFA is tight

Quasi-NSE Grammars: Definition

- Unary CFGs only generate regular languages

  [Ginsburg and Rice, 1962]

- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β are not empty iff A only generates a unary language
- qNSE grammars only generate regular languages [Andrei et al., 2003]

Quasi-NSE Grammars: Definition

- Unary CFGs only generate regular languages [Ginsburg and Rice, 1962]
- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{*}{\Longrightarrow} \alpha A \beta$
- α, β are not empty iff A only generates a unary language
- qNSE grammars only generate regular languages [Andrei et al., 2003]

Quasi-NSE Grammars: Definition

- Unary CFGs only generate regular languages

  [Ginsburg and Rice, 1962]
- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β are not empty iff A only generates a unary language
- qNSE grammars only generate regular languages [Andrei et al., 2003]

Quasi-NSE Grammars: Definition

- Unary CFGs only generate regular languages

  [Ginsburg and Rice, 1962]
- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β are not empty iff A only generates a unary language
- qNSE grammars only generate regular languages [Andrei et al., 2003]

Quasi-NSE Grammars: Definition

- Unary CFGs only generate regular languages [Ginsburg and Rice, 1962]
- Context-Free Grammar $G = (V, \Sigma, P, S)$
- $A \stackrel{\star}{\Longrightarrow} \alpha A \beta$
- α, β are not empty iff A only generates a unary language
- qNSE grammars only generate regular languages [Andrei et al., 2003]

Quasi-NSE Grammars

Bounds for unary CFGs grammars [Pighizzini et al., 2002]

- CFG with size s
- equivalent NFA: $2^{O(s)}$ states
- equivalent DFA: $2^{O(s^2)}$ states

Quasi-NSE Grammars

Bounds for unary CFGs grammars [Pighizzini et al., 2002]

- CFG with size s
- equivalent NFA: $2^{O(s)}$ states
- equivalent DFA: $2^{O(s^2)}$ states

Quasi-NSE Grammars

Bounds for unary CFGs grammars [Pighizzini et al., 2002]

- CFG with size s
- equivalent NFA: $2^{O(s)}$ states
- equivalent DFA: $2^{O(s^2)}$ states

Quasi-NSE Grammars

Bounds for unary CFGs grammars [Pighizzini et al., 2002]

- CFG with size s
- equivalent NFA: $2^{O(s)}$ states
- equivalent DFA: $2^{O(s^2)}$ states

Quasi-NSE Grammars

Bounds for unary CFGs grammars [Pighizzini et al., 2002]

- CFG with size s
- equivalent NFA: $2^{O(s)}$ states
- equivalent DFA: $2^{O(s^2)}$ states

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is $\mathrm{NSE} \rightarrow 2^{O(s_n)}$ states NFA N_n
- $G_u \rightarrow 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \rightarrow 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is NSE $\rightarrow 2^{O(s_n)}$ states NFA N_n
- $G_u \rightarrow 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \rightarrow 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is NSE $\rightarrow 2^{O(s_n)}$ states NFA N_n
- $G_u \rightarrow 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \rightarrow 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is $\mathrm{NSE} \to 2^{O(s_n)}$ states NFA N_n
- $G_u \to 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \to 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is NSE $\rightarrow 2^{O(s_n)}$ states NFA N_n
- $G_u \rightarrow 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \rightarrow 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is NSE $\rightarrow 2^{O(s_n)}$ states NFA N_n
- $G_u \rightarrow 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \rightarrow 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is NSE $\rightarrow 2^{O(s_n)}$ states NFA N_n
- $G_u \rightarrow 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \rightarrow 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is NSE $\rightarrow 2^{O(s_n)}$ states NFA N_n
- $G_u \rightarrow 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \rightarrow 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is NSE $\rightarrow 2^{O(s_n)}$ states NFA N_n
- $G_u \rightarrow 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \rightarrow 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

qNSE $\rightarrow$ NFA exponential

Quasi-NSE Grammars: Conversion

- Let G be a qNSE grammar with size s
- Split G in G_n (size s_n) and G_u (size s_u)
- G_n is NSE $\rightarrow 2^{O(s_n)}$ states NFA N_n
- $G_u \rightarrow 2^{O(s_u)}$ states NFA N_u
- Substitution of N_u in $N_n \rightarrow 2^{O(s_u)}2^{O(s_n)} = 2^{O(s)}$

qNSE $\rightarrow$ NFA exponential

qNSE $\rightarrow$ DFA double-exponential

Back to the Witness Language

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- Each block could have 2^n different configurations
- What happens if we consider some restrictions on the structure of the strings?

Back to the Witness Language

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- Each block could have 2^n different configurations
- What happens if we consider some restrictions on the structure of the strings?

Back to the Witness Language

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- Each block could have 2^n different configurations
- What happens if we consider some restrictions on the structure of the strings?

Back to the Witness Language

$$L_n = \underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{j+1}\ a_{j+2} \quad \cdots \quad a_{j+n}}_{x_j} \quad \cdots \quad \underbrace{a_{\dots} \quad \cdots a_{kn-1}\ a_{kn}}_{x_k}$$

x_k is equal to the reversal of some x_j

- Each block could have 2^n different configurations
- What happens if we consider some restrictions on the structure of the strings?

Study the letter-bounded case

The Letter-Bounded Case

- G is a qNSE of size s
- $L(G) \subseteq a_1^* a_2^* \cdots a_m^*$, for $\Sigma = \{a_1, a_2, \ldots, a_m\}$
- The equivalent NFA has $2^{O(s)}$
- [Herrmann et al., 2016] $\mathrm{NFA} \to \mathrm{DFA}$ bounded case: subexponential but still superpolynomial cost
- $\mathrm{qNSE} \to \mathrm{DFA}$ bouded case: superexponential upper bound

We can do better!

- Each "unary SCC" of size s' can be replaced by a $2^{O(s'^2)}$-state DFA.

The Letter-Bounded Case

- G is a qNSE of size s
- $L(G) \subseteq a_1^* a_2^* \cdots a_m^*$, for $\Sigma = \{a_1, a_2, \ldots, a_m\}$
- The equivalent NFA has $2^{O(s)}$
- [Herrmann et al., 2016] $\mathrm{NFA} \to \mathrm{DFA}$ bounded case: subexponential but still superpolynomial cost
- $\mathrm{qNSE} \to \mathrm{DFA}$ bouded case: superexponential upper bound

We can do better!

- Each "unary SCC" of size s' can be replaced by a $2^{O(s'^2)}$-state DFA.

The Letter-Bounded Case

- G is a qNSE of size s
- $L(G) \subseteq a_1^* a_2^* \cdots a_m^*$, for $\Sigma = \{a_1, a_2, \ldots, a_m\}$
- The equivalent NFA has $2^{O(s)}$
- [Herrmann et al., 2016] $\mathrm{NFA} \to \mathrm{DFA}$ bounded case: subexponential but still superpolynomial cost
- $\mathrm{qNSE} \to \mathrm{DFA}$ bouded case: superexponential upper bound

We can do better!

- Each "unary SCC" of size s' can be replaced by a $2^{O(s'^2)}$-state DFA.

The Letter-Bounded Case

- G is a qNSE of size s
- $L(G) \subseteq a_1^* a_2^* \cdots a_m^*$, for $\Sigma = \{a_1, a_2, \ldots, a_m\}$
- The equivalent NFA has $2^{O(s)}$

- [Herrmann et al., 2016] $\mathrm{NFA} \to \mathrm{DFA}$ bounded case:
  subexponential but still superpolynomial cost
- $\mathrm{qNSE} \to \mathrm{DFA}$ bouded case: superexponential upper bound
  We can do better!
- Each "unary SCC" of size s' can be replaced by a $2^{O(s'^2)}$-state DFA.

The Letter-Bounded Case

- G is a qNSE of size s
- $L(G) \subseteq a_1^* a_2^* \cdots a_m^*$, for $\Sigma = \{a_1, a_2, \ldots, a_m\}$
- The equivalent NFA has $2^{O(s)}$

- [Herrmann et al., 2016] $\mathrm{NFA} \to \mathrm{DFA}$ bounded case:
  subexponential but still superpolynomial cost
- $\mathrm{qNSE} \to \mathrm{DFA}$ bouded case: superexponential upper bound

We can do better!

- Each "unary SCC" of size s' can be replaced by a $2^{O(s'^2)}$-state DFA.

The Letter-Bounded Case

- G is a qNSE of size s
- $L(G) \subseteq a_1^* a_2^* \cdots a_m^*$, for $\Sigma = \{a_1, a_2, \ldots, a_m\}$
- The equivalent NFA has $2^{O(s)}$

- [Herrmann et al., 2016] $\mathrm{NFA} \to \mathrm{DFA}$ bounded case:
  subexponential but still superpolynomial cost
- $\mathrm{qNSE} \to \mathrm{DFA}$ bouded case: superexponential upper bound

  We can do better!
- Each "unary SCC" of size s' can be replaced by a $2^{O(s'^2)}$-state DFA.

The Letter-Bounded Case

- G is a qNSE of size s
- $L(G) \subseteq a_1^* a_2^* \cdots a_m^*$, for $\Sigma = \{a_1, a_2, \ldots, a_m\}$
- The equivalent NFA has $2^{O(s)}$

- [Herrmann et al., 2016] $\mathrm{NFA} \to \mathrm{DFA}$ bounded case:
  subexponential but still superpolynomial cost
- $\mathrm{qNSE} \to \mathrm{DFA}$ bouded case: superexponential upper bound
  We can do better!
- Each "unary SCC" of size s' can be replaced by a $2^{O(s'^2)}$-state DFA.

The Letter-Bounded Case

- G is a qNSE of size s
- $L(G) \subseteq a_1^* a_2^* \cdots a_m^*$, for $\Sigma = \{a_1, a_2, \ldots, a_m\}$
- The equivalent NFA has $2^{O(s)}$

- [Herrmann et al., 2016] $\mathrm{NFA} \to \mathrm{DFA}$ bounded case:
  subexponential but still superpolynomial cost
- $\mathrm{qNSE} \to \mathrm{DFA}$ bouded case: superexponential upper bound

We can do better!

- Each "unary SCC" of size s' can be replaced by a $2^{O(s'^2)}$-state DFA.

letter-bounded $\mathrm{qNSE} \to \mathrm{NFA}/\mathrm{DFA}$ exponential

Conclusion

	NFA	DFA
(q)NSE	exponential	double-exponential
l.b. qNSE	exponential	exponential

Thanks for your attention

Conclusion

	NFA	DFA
(q)NSE	exponential	double-exponential
l.b. qNSE	exponential	exponential

Thanks for your attention

Conclusion

	NFA	DFA
(q)NSE	exponential	double-exponential
l.b. qNSE	exponential	exponential

Thanks for your attention

Conclusion

	NFA	DFA
(q)NSE	exponential	double-exponential
l.b. qNSE	exponential	exponential

Thanks for your attention

Conclusion

	NFA	DFA
(q)NSE	exponential	double-exponential
l.b. qNSE	exponential	exponential

Thanks for your attention

Conclusion

	NFA	DFA
(q)NSE	exponential	double-exponential
l.b. qNSE	exponential	exponential

Thanks for your attention

Conclusion

	NFA	DFA
(q)NSE	exponential	double-exponential
l.b. qNSE	exponential	exponential

Thanks for your attention

Conclusion

	NFA	DFA
(q)NSE	exponential	double-exponential
l.b. qNSE	exponential	exponential

Thanks for your attention

Conclusion

	NFA	DFA
(q)NSE	exponential	double-exponential
l.b. qNSE	exponential	exponential

Thanks for your attention

Andrei, S., Cavadini, S. V., and Chin, W. (2003).
A new algorithm for regularizing one-letter context-free grammars.
Theor. Comput. Sci., 306(1-3):113–122.

Anselmo, M., Giammarresi, D., and Varricchio, S. (2002).
Finite automata and non-self-embedding grammars.
In Champarnaud, J. and Maurel, D., editors, *Implementation and Application of Automata, 7th International Conference, CIAA 2002, Tours, France, July 3-5, 2002, Revised Papers*, volume 2608 of *Lecture Notes in Computer Science*, pages 47–56. Springer.

Chomsky, N. (1959a).
On certain formal properties of grammars.
Information and Control, 2(2):137–167.

Chomsky, N. (1959b).

a note on phrase structure grammars.
Information and Control, 2(4):393–395.

Ginsburg, S. and Rice, H. G. (1962).

Two families of languages related to ALGOL.
J. ACM, 9(3):350–371.


Herrmann, A., Kutrib, M., Malcher, A., and Wendlandt, M. (2016).
Descriptional complexity of bounded regular languages.
In Câmpeanu, C., Manea, F., and Shallit, J., editors, *Descriptional Complexity of Formal Systems - 18th IFIP WG 1.2 International Conference, DCFS 2016, Bucharest, Romania, July 5-8, 2016. Proceedings*, volume 9777 of *Lecture Notes in Computer Science*, pages 138–152. Springer.


Pighizzini, G., Shallit, J., and Wang, M. (2002).
Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds.
J. Comput. Syst. Sci., 65(2):393–414.