Efficient Enumeration of Regular Languages

Margareta Ackerman Jeffrey Shallit

School of Computer Science, University of Waterloo

July 5, 2007

Cross-Section Enumeration Problem

- We represent regular languages via nondeterministic finite automata (NFA)
- The n^{th} cross-section of a language is L is $L \cap \Sigma^n$.

Cross-Section Enumeration Problem

- We represent regular languages via nondeterministic finite automata (NFA)
- The n^{th} cross-section of a language is L is $L \cap \Sigma^n$.

Definition

Cross-section Enumeration Problem: List all words in a regular language of length n in lexicographical order.

Cross-Section Enumeration Problem

- We represent regular languages via nondeterministic finite automata (NFA)
- The n^{th} cross-section of a language is L is $L \cap \Sigma^n$.

Definition

Cross-section Enumeration Problem: List all words in a regular language of length n in lexicographical order.

Example

$L(N) = (0 + 1)^*$
The 2^{nd}-cross-section is: $00, 01, 10, 11$.

Cross-Section Enumeration Problem

- We represent regular languages via nondeterministic finite automata (NFA)
- The n^{th} cross-section of a language is L is $L \cap \Sigma^n$.

Definition

Cross-section Enumeration Problem: List all words in a regular language of length n in lexicographical order.

Example

$L(N) = (0 + 1)^*$
The 2^{nd}-cross-section is: $00, 01, 10, 11$.

Example

$L(N) = \{w \in \{0,1\}^* \mid |w|_1 \equiv 0 \mod 3\}$
The 4^{th}-cross-section is: $0000, 0111, 1011, 1101, 1110$.

Enumeration Problem

Definition

Given words $u = u_1 u_2 \cdots u_n$ and $v = v_1 v_2 \cdots v_m$, $u < v$ according to *radix order* if $n < m$ or if $n = m$, $u \neq v$, and $u_i < v_i$ for the minimal i where $u_i \neq v_i$.

Enumeration Problem

Definition

Given words $u = u_1 u_2 \cdots u_n$ and $v = v_1 v_2 \cdots v_m$, $u < v$ according to *radix order* if $n < m$ or if $n = m$, $u \neq v$, and $u_i < v_i$ for the minimal i where $u_i \neq v_i$.

Definition

Enumeration Problem: List the first m words in a regular language in radix order.

Enumeration Problem

Definition

Given words $u = u_1 u_2 \cdots u_n$ and $v = v_1 v_2 \cdots v_m$, $u < v$ according to *radix order* if $n < m$ or if $n = m$, $u \neq v$, and $u_i < v_i$ for the minimal i where $u_i \neq v_i$.

Definition

Enumeration Problem: List the first m words in a regular language in radix order.

Example

$L(N) = (0 + 1)^*$
Enumerate the first 5 words in $L(N)$: $\epsilon, 0, 1, 00, 01$.

Enumeration Problem

Definition

Given words $u = u_1 u_2 \cdots u_n$ and $v = v_1 v_2 \cdots v_m$, $u < v$ according to *radix order* if $n < m$ or if $n = m$, $u \neq v$, and $u_i < v_i$ for the minimal i where $u_i \neq v_i$.

Definition

Enumeration Problem: List the first m words in a regular language in radix order.

Example

$L(N) = (0 + 1)^*$
Enumerate the first 5 words in $L(N)$: $\epsilon, 0, 1, 00, 01$.

Example

$L(N) = \{w \in \{0,1\}^* \mid |w|_1 \equiv 0 \mod 3\}$
Enumerate the first 10 words in $L(N)$:
$\epsilon, 0, 00, 000, 111, 0000, 0111, 1011, 1101, 1110$.

Applications

- Correctness testing, provide evidence that an NFA generates the expected language.

Applications

- Correctness testing, provide evidence that an NFA generates the expected language.
- Ex. Verify that an NFA has the property that the least word not accepted is exponential in length (construction by Shallit).

Applications

- Correctness testing, provide evidence that an NFA generates the expected language.
- Ex. Verify that an NFA has the property that the least word not accepted is exponential in length (construction by Shallit).
- Test if two NFAs accept the same language.

Applications

- Correctness testing, provide evidence that an NFA generates the expected language.
- Ex. Verify that an NFA has the property that the least word not accepted is exponential in length (construction by Shallit).
- Test if two NFAs accept the same language.
- If sufficiently many words are generated, then we can verify if two NFAs accept the same language (Conway, 1971).

Applications

Given an NFA on s states, decide if every word it accepts is a power (a string of the form $x^n, |x| \geq 1, n \geq 2$).

- If every word is a power, then the NFA accepts no more than $7s$ words of each length, and further, if it accepts a non-power, it must accept a non-power of length $< 3s$ (Anderson, Rampersad, Santean, and Shallit.)

Applications

Given an NFA on s states, decide if every word it accepts is a power (a string of the form $x^n, |x| \geq 1, n \geq 2$).

- If every word is a power, then the NFA accepts no more than $7s$ words of each length, and further, if it accepts a non-power, it must accept a non-power of length $< 3s$ (Anderson, Rampersad, Santean, and Shallit.)
- We get an efficient algorithm by enumerating all the words of length $1, 2, \ldots, 3s - 1$ and testing if each is a power, stopping if the length of any cross-section exceeds $7s$.

Previous Work - Grail

- Grail+ 3.0, a symbolic computation environment, implements an enumeration algorithm under the function `fmenum`.

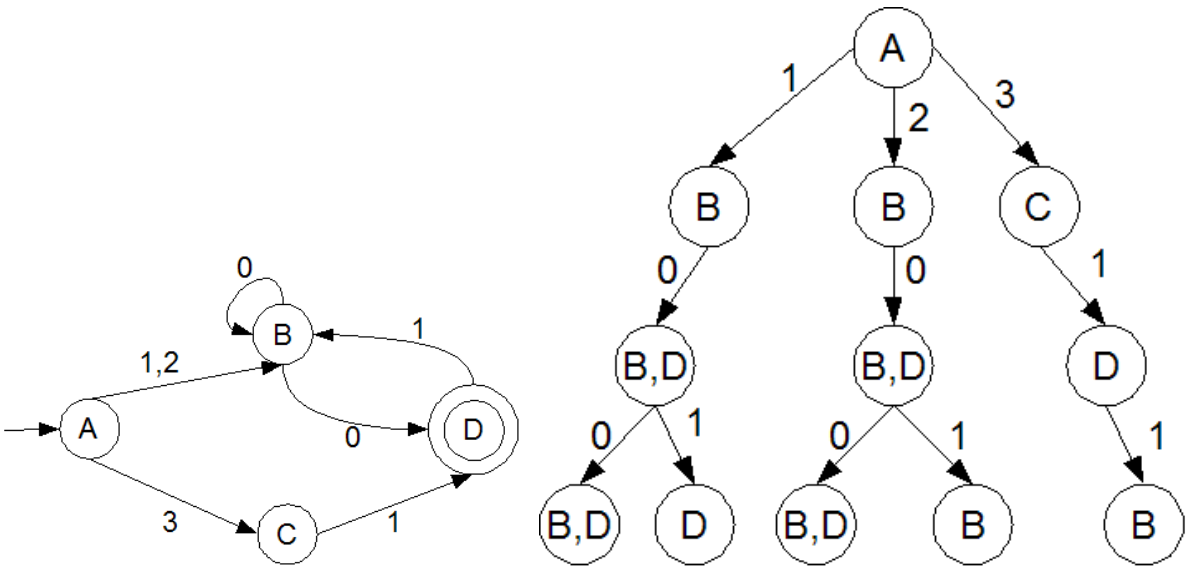
Previous Work - Grail

- Grail+ 3.0, a symbolic computation environment, implements an enumeration algorithm under the function `fmenum`.
- The function `fmenum` performs Breadth-First-Search on the tree of paths that can be traversed on an NFA.

Previous Work - Grail

- Grail+ 3.0, a symbolic computation environment, implements an enumeration algorithm under the function `fmenum`.
- The function `fmenum` performs Breadth-First-Search on the tree of paths that can be traversed on an NFA.

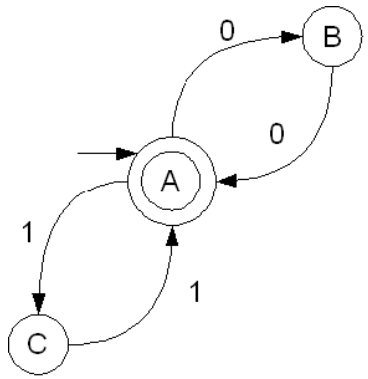
Enumerate the 3^{rd} cross-section of the following NFA.



(e) NFA (f) Illustration of Grail algorithm

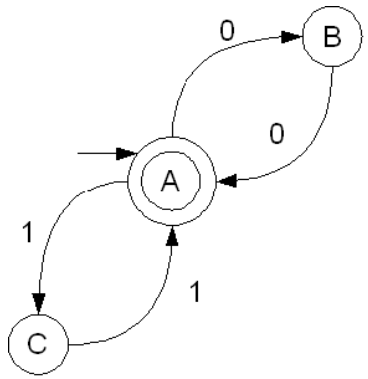
Previous Work - Grail

- Algorithm may do exponential work for empty output. For the NFA below, it will take $\Theta(n2^{n/2})$ operations to enumerate the n^{th}-cross-section, where n is odd.



Previous Work - Grail

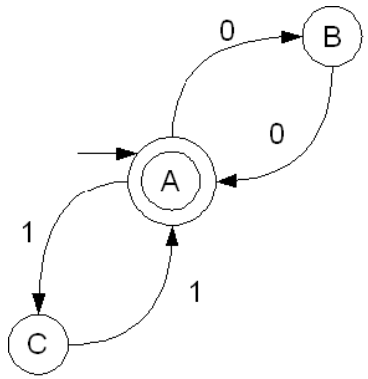
- Algorithm may do exponential work for empty output. For the NFA below, it will take $\Theta(n2^{n/2})$ operations to enumerate the n^{th}-cross-section, where n is odd.



- Takes $O(s^2\sigma^{n+1})$ operations to enumerate the n^{th} cross-section.
  s: number of states. σ: alphabet size.

Previous Work - Grail

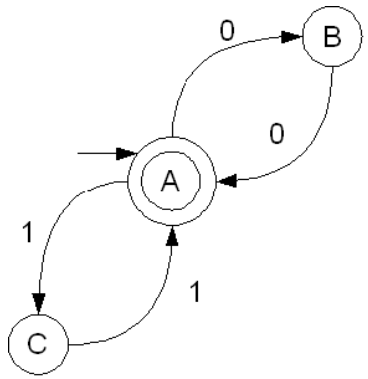
- Algorithm may do exponential work for empty output. For the NFA below, it will take $\Theta(n2^{n/2})$ operations to enumerate the n^{th}-cross-section, where n is odd.



- Takes $O(s^2\sigma^{n+1})$ operations to enumerate the n^{th} cross-section.
  s: number of states. σ: alphabet size.
- Takes $O(s^2\sigma^{k+1})$ operations to enumerate the first m words accepted by the NFA.
  k: length of words in last cross-section examined.

Previous Work - Grail

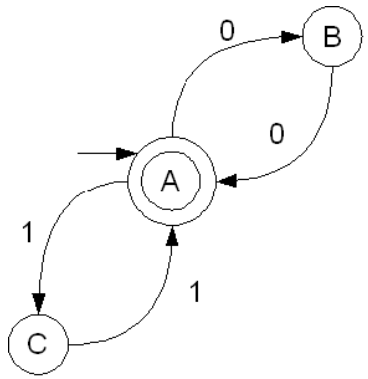
- Algorithm may do exponential work for empty output. For the NFA below, it will take $\Theta(n2^{n/2})$ operations to enumerate the n^{th}-cross-section, where n is odd.



- Takes $O(s^2\sigma^{n+1})$ operations to enumerate the n^{th} cross-section.
  s: number of states. σ: alphabet size.
- Takes $O(s^2\sigma^{k+1})$ operations to enumerate the first m words accepted by the NFA.
  k: length of words in last cross-section examined.
- Works well in practice for small input sizes.

Previous Work - Grail

- Algorithm may do exponential work for empty output. For the NFA below, it will take $\Theta(n2^{n/2})$ operations to enumerate the n^{th}-cross-section, where n is odd.



- Takes $O(s^2\sigma^{n+1})$ operations to enumerate the n^{th} cross-section.
  s: number of states. σ: alphabet size.
- Takes $O(s^2\sigma^{k+1})$ operations to enumerate the first m words accepted by the NFA.
  k: length of words in last cross-section examined.
- Works well in practice for small input sizes.
- We found a few bugs in Grail. Words are not always output in lexicographical order and for some NFAs words are missing from the enumeration.

Previous Work - Mäkinen's algorithm

- Mäkinen finds an efficient algorithm in the unit-cost model ("On lexicographic enumeration of regular and context-free languages." *Acta Cybernetica*, **13** (1997), 55–61.)

Previous Work - Mäkinen's algorithm

- Mäkinen finds an efficient algorithm in the unit-cost model ("On lexicographic enumeration of regular and context-free languages." *Acta Cybernetica*, **13** (1997), 55–61.)
- Mäkinen's algorithm uses dynamic programming. It finds the minimal words of length 1 through n starting at each state.

Previous Work - Mäkinen's algorithm

- Mäkinen finds an efficient algorithm in the unit-cost model ("On lexicographic enumeration of regular and context-free languages." *Acta Cybernetica*, **13** (1997), 55–61.)
- Mäkinen's algorithm uses dynamic programming. It finds the minimal words of length 1 through n starting at each state.
- Originally analyzed in the unit-cost model.

Previous Work - Mäkinen's algorithm

- Mäkinen finds an efficient algorithm in the unit-cost model ("On lexicographic enumeration of regular and context-free languages." *Acta Cybernetica*, **13** (1997), 55–61.)
- Mäkinen's algorithm uses dynamic programming. It finds the minimal words of length 1 through n starting at each state.
- Originally analyzed in the unit-cost model.
- We analyze its running time in the bit-complexity model and modify the algorithm.

Previous Work - Mäkinen's algorithm

- Mäkinen finds an efficient algorithm in the unit-cost model ("On lexicographic enumeration of regular and context-free languages." *Acta Cybernetica*, **13** (1997), 55–61.)
- Mäkinen's algorithm uses dynamic programming. It finds the minimal words of length 1 through n starting at each state.
- Originally analyzed in the unit-cost model.
- We analyze its running time in the bit-complexity model and modify the algorithm.
- Mäkinen's cross-section enumeration algorithm uses $O(s^2n^2 + \sigma s^2 t)$ operations. *Algorithm is quadratic in n.*

Previous Work - Mäkinen's algorithm

- Mäkinen finds an efficient algorithm in the unit-cost model ("On lexicographic enumeration of regular and context-free languages." *Acta Cybernetica*, **13** (1997), 55–61.)
- Mäkinen's algorithm uses dynamic programming. It finds the minimal words of length 1 through n starting at each state.
- Originally analyzed in the unit-cost model.
- We analyze its running time in the bit-complexity model and modify the algorithm.
- Mäkinen's cross-section enumeration algorithm uses $O(s^2n^2 + \sigma s^2 t)$ operations. *Algorithm is quadratic in n.*
- Mäkinen's enumeration algorithm uses $O(s^2e + \sigma s^2 t)$ operations.
  e: number of empty cross-sections. n: length of words in cross-section.
  t: output size. s: number of states in the NFA. σ: size of the alphabet of the NFA.

Previous Work

Pál Dömösi gives a cross-section enumeration algorithm, where finding each consecutive word is superexponential in the size of the cross-section (Dömösi, 1998.)

Contributions

1. New algorithm for cross-section enumeration that is linear in n.

Contributions

1. New algorithm for cross-section enumeration that is linear in n.
2. Variants of Mäkinen's algorithm with best practical performance.

Contributions

1. New algorithm for cross-section enumeration that is linear in n.
2. Variants of Mäkinen's algorithm with best practical performance.
3. Extensive performance analysis.

Algorithm Framework

- We describe an algorithm for finding the minimal word in a cross-section.

Algorithm Framework

- We describe an algorithm for finding the minimal word in a cross-section.
- Then, we use that algorithm for finding the rest of the words in a cross-section.

Algorithm Framework

- We describe an algorithm for finding the minimal word in a cross-section.
- Then, we use that algorithm for finding the rest of the words in a cross-section.
- We construct an enumeration algorithm through repeated application of the cross-section enumeration algorithm.

Lookahead-Matrix Algorithm For Finding the Minimal Word

- Starting at the start state, we traverse the NFA only down paths that will lead to a final state in the required number of steps.

Lookahead-Matrix Algorithm For Finding the Minimal Word

- Starting at the start state, we traverse the NFA only down paths that will lead to a final state in the required number of steps.
- We need a method of determining whether a word of length i can be completed to a word of length n in $n - i$ steps from a given state.

Lookahead-Matrix Algorithm For Finding the Minimal Word

- Starting at the start state, we traverse the NFA only down paths that will lead to a final state in the required number of steps.
- We need a method of determining whether a word of length i can be completed to a word of length n in $n - i$ steps from a given state.
- Precompute M, the adjacency matrix of the NFA; $M_{p,q} = 1$ if there is a transition from state p to state q, and $M_{p,q} = 0$ otherwise.

Lookahead-Matrix Algorithm For Finding the Minimal Word

- Starting at the start state, we traverse the NFA only down paths that will lead to a final state in the required number of steps.
- We need a method of determining whether a word of length i can be completed to a word of length n in $n - i$ steps from a given state.
- Precompute M, the adjacency matrix of the NFA; $M_{p,q} = 1$ if there is a transition from state p to state q, and $M_{p,q} = 0$ otherwise.
- $M^i_{p,q} = 1$ (using bit-arithmetic) if and only if there is a path from state p to state q of length exactly i.

Lookahead-Matrix Algorithm For Finding the Minimal Word

Definition

A state q in an NFA N is *i-complete* if there is a path in N of length i starting at q and ending at a final state.

- Notice that M^i enables us to determine if a given state is i-complete.

Lookahead-Matrix Algorithm For Finding the Minimal Word

Definition

A state q in an NFA N is *i-complete* if there is a path in N of length i starting at q and ending at a final state.

- Notice that M^i enables us to determine if a given state is i-complete.
- To find the minimal word of length n:
  - Compute $M, M^2, \ldots, M^n$ using boolean matrix multiplication.
  - Find the set of $(n-1)$-complete states, S_1, reachable from the start state on the minimal possible symbol a_1.
  - Then find the set of $(n-2)$-complete states, S_2, reachable from any state in S_1 on the minimal possible symbol a_2.
  - Repeat this process for a total of n iterations.
  - Then $a_1 a_2 \cdots a_n$ is the minimal word of length n.

Finding the Minimal Word

Algorithm

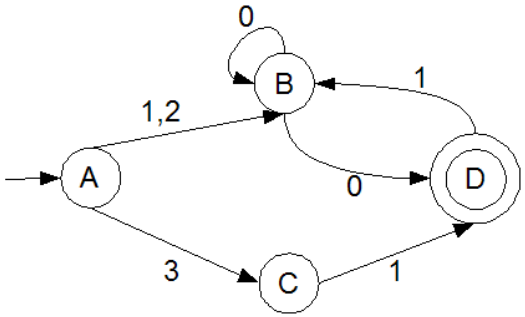
minWordLM(n,N)

INPUT: A nonnegative integer n and an NFA N.
OUTPUT: The minimal word of length n accepted by N. Updates state stack S for a potential subsequent call to minWord or nextWord.

Compute $M, M^2, \ldots, M^n$
$S_0 = \{s_0\}$
IF $M^n_{q,f} = 0$ *for all* $f \in F, q \in S_0$
    return NULL
w = empty word
FOR $i \leftarrow 0 \cdots n-1$
    $a_{i+1} = \min(a \in \Sigma \mid \exists u \in S_i, f \in F$ *where* $M^{n-1-i}_{v,f} = 1$ *for some* $v \in \delta(u,a))$
    $w = wa_{i+1}$
    $S_{i+1} = \{v \in \cup_{u \in S_i} \delta(u, a_{i+1}) \mid M^{n-1-i}_{v,f} = 1$ *for some* $f \in F\}$
    IF $i \neq n-1$
        push(S,S_{i+1}*)*
return w

Finding the Minimal Word

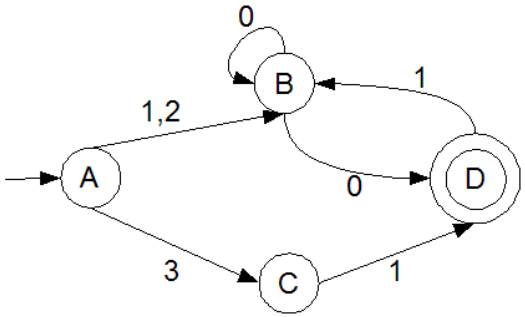
Find the minimal word w of length 4 accepted by the following NFA.



$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, M^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, M^3 = M^4 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Finding the Minimal Word

Find the minimal word w of length 4 accepted by the following NFA.

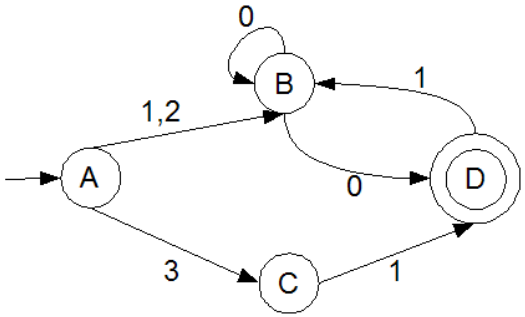


$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, M^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, M^3 = M^4 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

1. State A is 4-complete. Therefore, w exists.
   Set $w = \epsilon$, $S = \{\{A\}\}$.

Finding the Minimal Word

Find the minimal word w of length 4 accepted by the following NFA.

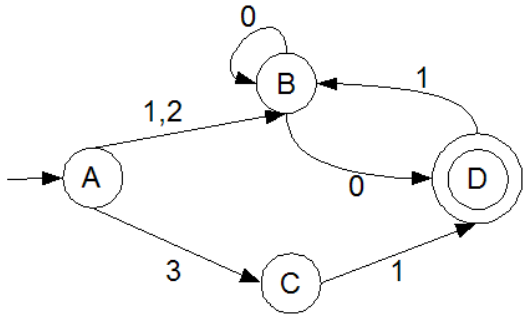


$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, M^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, M^3 = M^4 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

1. State A is 4-complete. Therefore, w exists.
   Set $w = \epsilon$, $S = \{\{A\}\}$.
2. State B is 3-complete and reachable from A on the minimal* char.
   Set $w = 1$, $S = \{\{A\}, \{B\}\}$.

Finding the Minimal Word

Find the minimal word w of length 4 accepted by the following NFA.

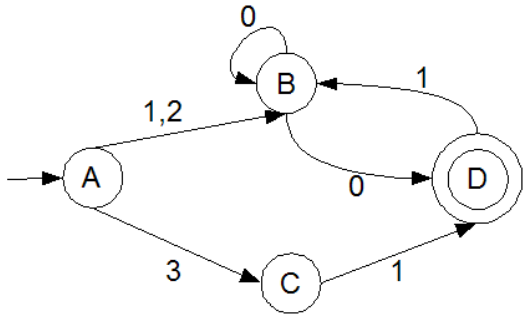


$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, M^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, M^3 = M^4 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

1. State A is 4-complete. Therefore, w exists.
   Set $w = \epsilon$, $S = \{\{A\}\}$.
2. State B is 3-complete and reachable from A on the minimal* char.
   Set $w = 1$, $S = \{\{A\}, \{B\}\}$.
3. States B and D are 2-complete and reachable from B on the minimal* char.
   Set $w = 10$, $S = \{\{A\}, \{B\}, \{B, D\}\}$.

Finding the Minimal Word

Find the minimal word w of length 4 accepted by the following NFA.

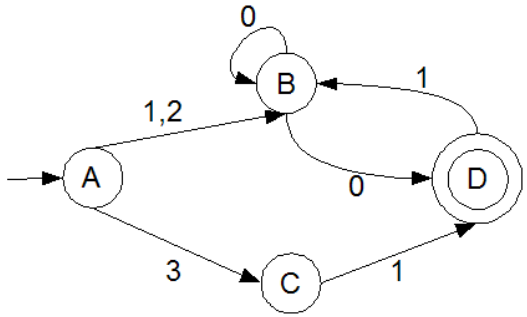


$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, M^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, M^3 = M^4 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

1. State A is 4-complete. Therefore, w exists.
   Set $w = \epsilon$, $S = \{\{A\}\}$.
2. State B is 3-complete and reachable from A on the minimal* char.
   Set $w = 1$, $S = \{\{A\}, \{B\}\}$.
3. States B and D are 2-complete and reachable from B on the minimal* char.
   Set $w = 10$, $S = \{\{A\}, \{B\}, \{B, D\}\}$.
4. State B is 1-complete and reachable from {B, D}.
   Set $w = 100$, $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$.

Finding the Minimal Word

Find the minimal word w of length 4 accepted by the following NFA.



$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, M^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, M^3 = M^4 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

1. State A is 4-complete. Therefore, w exists.
   Set $w = \epsilon$, $S = \{\{A\}\}$.
2. State B is 3-complete and reachable from A on the minimal* char.
   Set $w = 1$, $S = \{\{A\}, \{B\}\}$.
3. States B and D are 2-complete and reachable from B on the minimal* char.
   Set $w = 10$, $S = \{\{A\}, \{B\}, \{B, D\}\}$.
4. State B is 1-complete and reachable from {B, D}.
   Set $w = 100$, $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$.
5. State D is final and reachable from B. Set $w = 1000$.

Complexity of Finding the Minimal Word

- Since the matrices require $O(s^2 n)$ space, `minWordLM` uses $O(s^2 n)$ space.

Complexity of Finding the Minimal Word

- Since the matrices require $O(s^2 n)$ space, `minWordLM` uses $O(s^2 n)$ space.
- The best bound for matrix multiplication is $O(s^{2.376})$ (Coppersmith, 90).

Complexity of Finding the Minimal Word

- Since the matrices require $O(s^2 n)$ space, `minWordLM` uses $O(s^2 n)$ space.
- The best bound for matrix multiplication is $O(s^{2.376})$ (Coppersmith, 90).
- All operations other than the matrix multiplication in the algorithm cost $O(\sigma s^2 n)$.

Complexity of Finding the Minimal Word

- Since the matrices require $O(s^2 n)$ space, `minWordLM` uses $O(s^2 n)$ space.
- The best bound for matrix multiplication is $O(s^{2.376})$ (Coppersmith, 90).
- All operations other than the matrix multiplication in the algorithm cost $O(\sigma s^2 n)$.

Theorem

The Lookahead-Matrix algorithm finds the minimal word of length n in $O(s^{2.376} n + \sigma s^2 n)$ time and $O(s^2 n)$ space.

Finding the Next Word in a Cross-Section

- When looking for the minimal word, store the sets of states $\{s_0\}, S_1, S_2, \ldots, S_{n-1}$ on a state stack.

Finding the Next Word in a Cross-Section

- When looking for the minimal word, store the sets of states $\{s_0\}, S_1, S_2, \ldots, S_{n-1}$ on a state stack.
- Pop S_{n-1} from the state stack and check if it can reach any final state reachable from a character other than a_n.

Finding the Next Word in a Cross-Section

- When looking for the minimal word, store the sets of states $\{s_0\}, S_1, S_2, \ldots, S_{n-1}$ on a state stack.
- Pop S_{n-1} from the state stack and check if it can reach any final state reachable from a character other than a_n.
- If so, complete the word $a_1 a_2 \ldots a_{n-1}$ with the minimal 1-character word $> a_n$ reachable from S_{n-1}.

Finding the Next Word in a Cross-Section

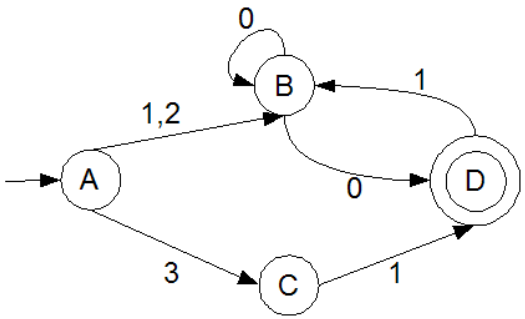
- When looking for the minimal word, store the sets of states $\{s_0\}, S_1, S_2, \ldots, S_{n-1}$ on a state stack.
- Pop S_{n-1} from the state stack and check if it can reach any final state reachable from a character other than a_n.
- If so, complete the word $a_1 a_2 \ldots a_{n-1}$ with the minimal 1-character word $> a_n$ reachable from S_{n-1}.
- If not, pop S_{n-2} and look for 1-complete states reachable from S_{n-2} on a character $> a_{n-1}$.

Finding the Next Word in a Cross-Section

- When looking for the minimal word, store the sets of states $\{s_0\}, S_1, S_2, \ldots, S_{n-1}$ on a state stack.
- Pop S_{n-1} from the state stack and check if it can reach any final state reachable from a character other than a_n.
- If so, complete the word $a_1 a_2 \ldots a_{n-1}$ with the minimal 1-character word $> a_n$ reachable from S_{n-1}.
- If not, pop S_{n-2} and look for 1-complete states reachable from S_{n-2} on a character $> a_{n-1}$.
- Continue until the next word is found or the state stack is empty.

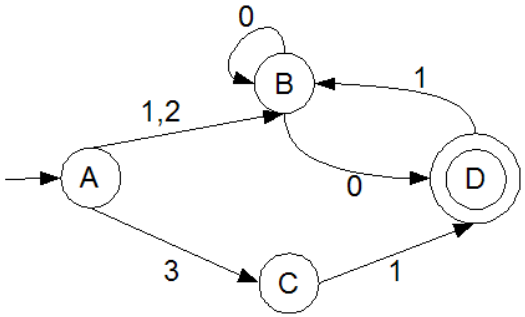
Finding the Next Word

Given the minimal word $w = 1000$, and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the second word u of length 4 accepted by the following NFA.



Finding the Next Word

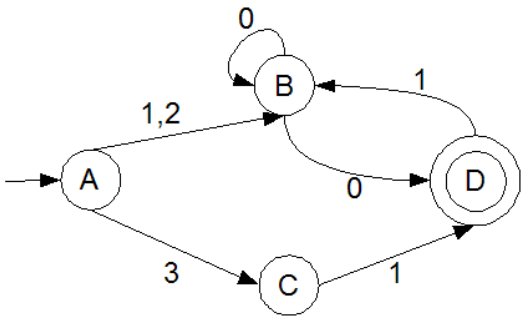
Given the minimal word $w = 1000$, and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the second word u of length 4 accepted by the following NFA.



1. Pop $\{B\}$ from S. Set $u = 100$, $a = 0$. We cannot reach a final state from $\{B\}$ on a character $> a$.

Finding the Next Word

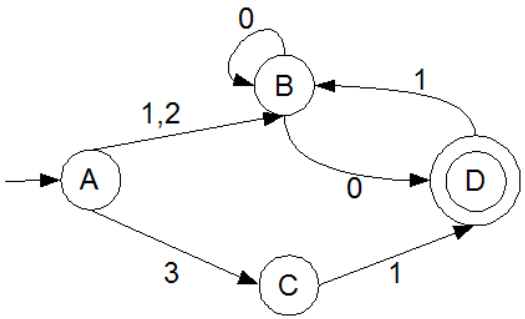
Given the minimal word $w = 1000$, and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the second word u of length 4 accepted by the following NFA.



1. Pop $\{B\}$ from S. Set $u = 100$, $a = 0$. We cannot reach a final state from $\{B\}$ on a character $> a$.
2. Pop $\{B, D\}$ from S. Remove last character from u and assign it to a. We get $u = 10$, $a = 0$. State B is 1-complete and reachable from state D on character $1 > a$.

Finding the Next Word

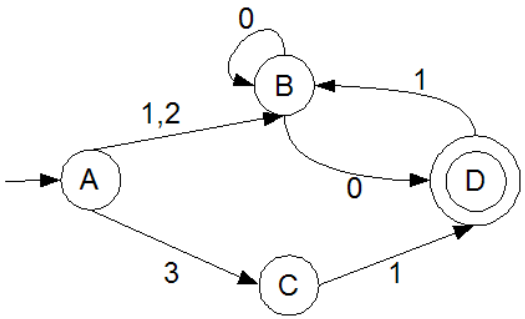
Given the minimal word $w = 1000$, and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the second word u of length 4 accepted by the following NFA.



1. Pop $\{B\}$ from S. Set $u = 100$, $a = 0$. We cannot reach a final state from $\{B\}$ on a character $> a$.
2. Pop $\{B, D\}$ from S. Remove last character from u and assign it to a. We get $u = 10$, $a = 0$. State B is 1-complete and reachable from state D on character $1 > a$.
3. Push $\{B, D\}$ and $\{B\}$ onto S. Append 1 to u, giving $u = 101$.

Finding the Next Word

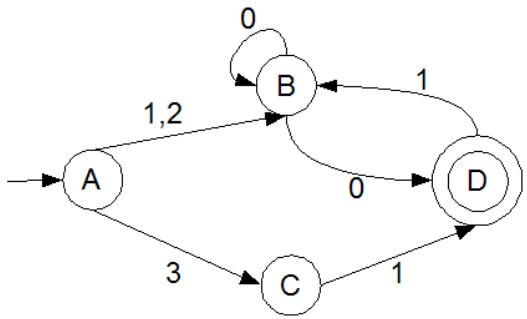
Given the minimal word $w = 1000$, and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the second word u of length 4 accepted by the following NFA.



1. Pop $\{B\}$ from S. Set $u = 100$, $a = 0$. We cannot reach a final state from $\{B\}$ on a character $> a$.
2. Pop $\{B, D\}$ from S. Remove last character from u and assign it to a. We get $u = 10$, $a = 0$. State B is 1-complete and reachable from state D on character $1 > a$.
3. Push $\{B, D\}$ and $\{B\}$ onto S. Append 1 to u, giving $u = 101$.
4. Find the minimal word of length 1 from $\{B\}$ to a final state and append it to u.

Finding the Next Word

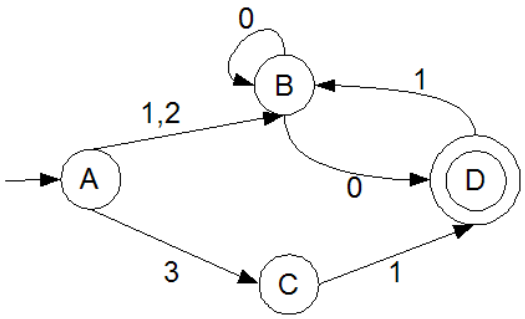
Given the minimal word $w = 1000$, and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the second word u of length 4 accepted by the following NFA.



1. Pop $\{B\}$ from S. Set $u = 100$, $a = 0$. We cannot reach a final state from $\{B\}$ on a character $> a$.
2. Pop $\{B, D\}$ from S. Remove last character from u and assign it to a. We get $u = 10$, $a = 0$. State B is 1-complete and reachable from state D on character $1 > a$.
3. Push $\{B, D\}$ and $\{B\}$ onto S. Append 1 to u, giving $u = 101$.
4. Find the minimal word of length 1 from $\{B\}$ to a final state and append it to u.
5. We get $u = 1010$ and $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$.

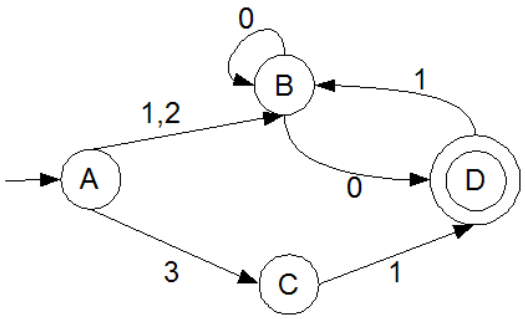
Finding the Next Word

Given word $u = 1010$ and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the next word v of length 4 accepted by the NFA.



Finding the Next Word

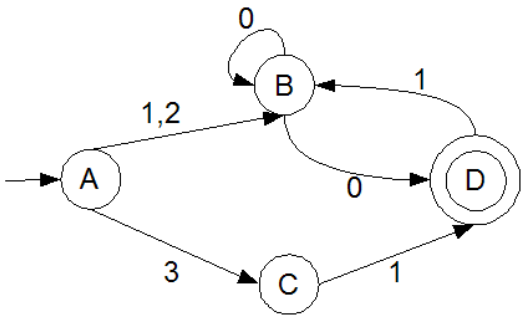
Given word $u = 1010$ and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the next word v of length 4 accepted by the NFA.



1. Pop $\{B\}$ from S. Set $v = 101$, $a = 0$.

Finding the Next Word

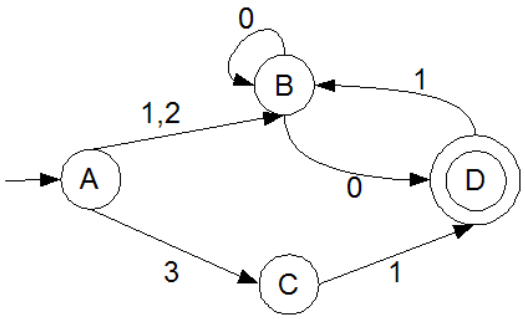
Given word $u = 1010$ and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the next word v of length 4 accepted by the NFA.



1. Pop $\{B\}$ from S. Set $v = 101$, $a = 0$.
2. Pop $\{B, D\}$ from S. Now $v = 10$, $a = 1$.

Finding the Next Word

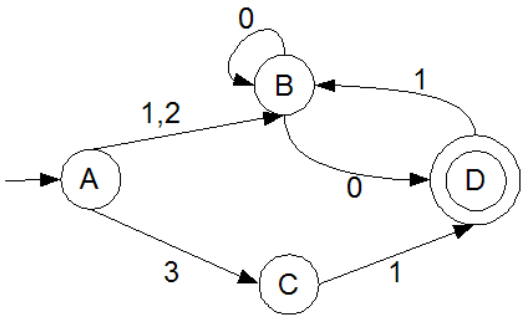
Given word $u = 1010$ and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the next word v of length 4 accepted by the NFA.



1. Pop $\{B\}$ from S. Set $v = 101$, $a = 0$.
2. Pop $\{B, D\}$ from S. Now $v = 10$, $a = 1$.
3. Pop $\{B\}$ from S. Now $v = 1$, $a = 0$.

Finding the Next Word

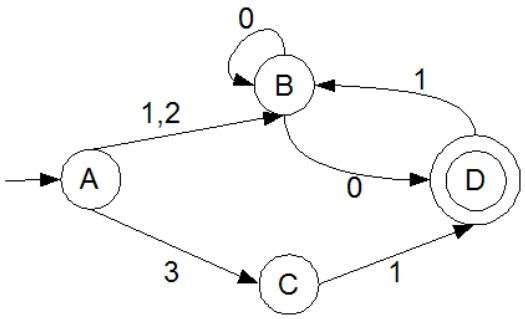
Given word $u = 1010$ and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the next word v of length 4 accepted by the NFA.



1. Pop $\{B\}$ from S. Set $v = 101$, $a = 0$.
2. Pop $\{B, D\}$ from S. Now $v = 10$, $a = 1$.
3. Pop $\{B\}$ from S. Now $v = 1$, $a = 0$.
4. Pop $\{A\}$ from S. Now $v = \epsilon$, $a = 1$. State B is a 3-complete state reachable from $\{A\}$ on character $2 > a$.

Finding the Next Word

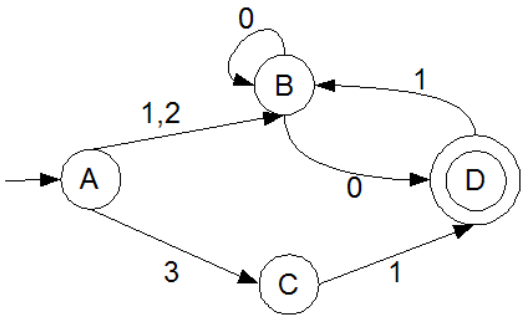
Given word $u = 1010$ and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the next word v of length 4 accepted by the NFA.



1. Pop $\{B\}$ from S. Set $v = 101$, $a = 0$.
2. Pop $\{B, D\}$ from S. Now $v = 10$, $a = 1$.
3. Pop $\{B\}$ from S. Now $v = 1$, $a = 0$.
4. Pop $\{A\}$ from S. Now $v = \epsilon$, $a = 1$. State B is a 3-complete state reachable from $\{A\}$ on character $2 > a$.
5. Push $\{A\}$ and $\{B\}$ onto S. Append 2 to v, giving $v = 2$.

Finding the Next Word

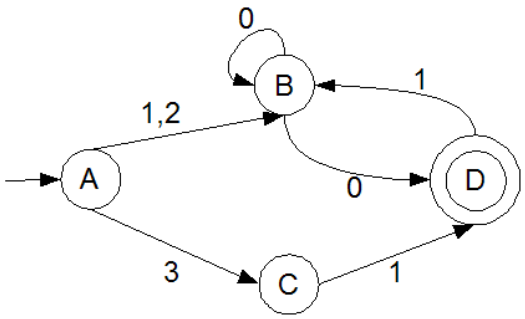
Given word $u = 1010$ and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the next word v of length 4 accepted by the NFA.



1. Pop $\{B\}$ from S. Set $v = 101$, $a = 0$.
2. Pop $\{B, D\}$ from S. Now $v = 10$, $a = 1$.
3. Pop $\{B\}$ from S. Now $v = 1$, $a = 0$.
4. Pop $\{A\}$ from S. Now $v = \epsilon$, $a = 1$. State B is a 3-complete state reachable from $\{A\}$ on character $2 > a$.
5. Push $\{A\}$ and $\{B\}$ onto S. Append 2 to v, giving $v = 2$.
6. Find the minimal word of length 3 from $\{B\}$ to a final state and append it to v.

Finding the Next Word

Given word $u = 1010$ and state stack $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$, find the next word v of length 4 accepted by the NFA.



1. Pop $\{B\}$ from S. Set $v = 101$, $a = 0$.
2. Pop $\{B, D\}$ from S. Now $v = 10$, $a = 1$.
3. Pop $\{B\}$ from S. Now $v = 1$, $a = 0$.
4. Pop $\{A\}$ from S. Now $v = \epsilon$, $a = 1$. State B is a 3-complete state reachable from $\{A\}$ on character $2 > a$.
5. Push $\{A\}$ and $\{B\}$ onto S. Append 2 to v, giving $v = 2$.
6. Find the minimal word of length 3 from $\{B\}$ to a final state and append it to v.
7. Thus, $v = 2000$ and $S = \{\{A\}, \{B\}, \{B, D\}, \{B\}\}$.

Lookahead-Matrix Cross-Section Enumeration Algorithm

- The algorithm `CrossSectionLM` finds the minimal word in the cross-section and repeatedly applies the algorithm for finding the next word in the cross-section, until the state stack is empty.

Lookahead-Matrix Cross-Section Enumeration Algorithm

- The algorithm `CrossSectionLM` finds the minimal word in the cross-section and repeatedly applies the algorithm for finding the next word in the cross-section, until the state stack is empty.
- Finding the minimal word costs $O(s^{2.376}n + \sigma s^2 n)$.

Lookahead-Matrix Cross-Section Enumeration Algorithm

- The algorithm `CrossSectionLM` finds the minimal word in the cross-section and repeatedly applies the algorithm for finding the next word in the cross-section, until the state stack is empty.
- Finding the minimal word costs $O(s^{2.376}n + \sigma s^2 n)$.
- Finding the remaining words costs $O(\sigma s^2 t)$.

Lookahead-Matrix Cross-Section Enumeration Algorithm

- The algorithm `CrossSectionLM` finds the minimal word in the cross-section and repeatedly applies the algorithm for finding the next word in the cross-section, until the state stack is empty.
- Finding the minimal word costs $O(s^{2.376}n + \sigma s^2 n)$.
- Finding the remaining words costs $O(\sigma s^2 t)$.

Theorem

The algorithm `crossSectionLM` uses $O(s^{2.376}n + \sigma s^2 t)$ operations.

t: output size. s: number of states. σ: alphabet size. n: length of words in cross-section.

Lookahead-Matrix Enumeration Algorithm

- The algorithm `enumLM` consists of repeated applications of `CrossSectionLM`.

Lookahead-Matrix Enumeration Algorithm

- The algorithm `enumLM` consists of repeated applications of `CrossSectionLM`.
- If an empty cross-section is encountered in `enumLM`, the algorithm performs $O(s^{2.376})$ operations to determine that.

Lookahead-Matrix Enumeration Algorithm

- The algorithm `enumLM` consists of repeated applications of `CrossSectionLM`.
- If an empty cross-section is encountered in `enumLM`, the algorithm performs $O(s^{2.376})$ operations to determine that.
- The algorithm `enumLM` does $O(s^{2.376} + \sigma s^2 t)$ operations for each non-empty cross-section.

Lookahead-Matrix Enumeration Algorithm

- The algorithm `enumLM` consists of repeated applications of `CrossSectionLM`.
- If an empty cross-section is encountered in `enumLM`, the algorithm performs $O(s^{2.376})$ operations to determine that.
- The algorithm `enumLM` does $O(s^{2.376} + \sigma s^2 t)$ operations for each non-empty cross-section.

Theorem

The algorithm `enumLM` uses $O(s^{2.376}(c + e) + \sigma s^2 t)$ operations.

e: number of empty cross-sections encountered throughout the enumeration.
c: number of non-empty cross-sections encountered throughout the enumeration.
t: output size. s: number of states. σ: alphabet size.

- The number of consecutive empty cross-sections is at most $s - 1$. Thus, $e < cs < ts$.

Mäkinen's Algorithms

- Uses dynamic programming to find the minimal word in each cross-section.

Mäkinen's Algorithms

- Uses dynamic programming to find the minimal word in each cross-section.
- Originally, algorithms were analyzed in the unit-cost model.

Mäkinen's Algorithms

- Uses dynamic programming to find the minimal word in each cross-section.
- Originally, algorithms were analyzed in the unit-cost model.
- We give variants of Mäkinen's algorithms which we analyze in the bit-complexity model.

Mäkinen's Minimal Word Algorithm

Algorithm

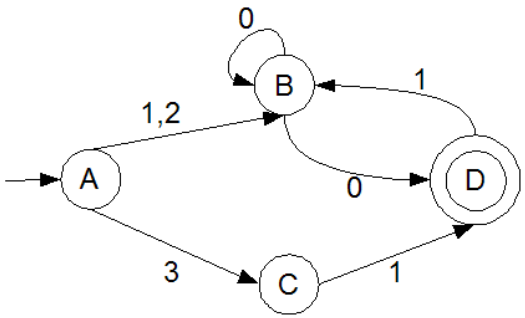
minWordMäkinen(n, N)

INPUT: A positive integer n and an NFA N.
OUTPUT: Table $A^{min}[1 \cdots n]$ *for each state* $A \in Q$ *where* $A^{min}[i]$ *is the minimal word of length i starting at state A.*

FOR each $A \in Q$
    IF for all $a \in \Sigma$, $\delta(A, a) \cap F = \emptyset$
        $A^{min}[1] =$ *NULL*
    ELSE
        $A^{min}[1] = \min\{a \in \Sigma \mid \delta(A, a) \cap F \neq \emptyset\}$
FOR $i \leftarrow 2 \cdots n$
    FOR each $A \in Q$
        $min =$ *NULL*
        FOR each $B \in Q$ *and minimal* $a \in \Sigma$ *such that* $B \in \delta(A, a)$
            IF $B^{min}[i-1] \neq$ *NULL*
                IF $aB^{min}[i-1] < min$ *OR* $min =$ *NULL*
                $min \leftarrow aB^{min}[i-1]$
        $A^{min}[i] = min$
RETURN $\{A^{min} \mid A \in Q\}$

Mäkinen's Minimal Word Algorithm

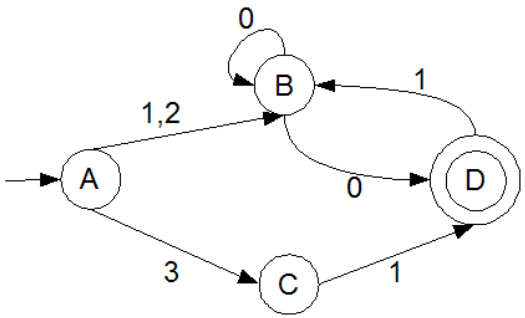
Find the minimal word of length 4 in the NFA.



states/length	1	2	3	4
A	x			
B	0			
C	1			
D	x			

Mäkinen's Minimal Word Algorithm

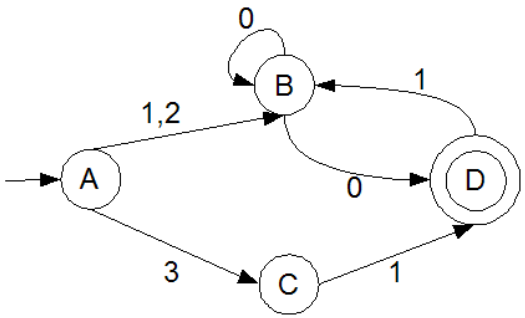
Find the minimal word of length 4 in the NFA.



states/length	1	2	3	4
A	x	10		
B	0	00		
C	1	x		
D	x	10		

Mäkinen's Minimal Word Algorithm

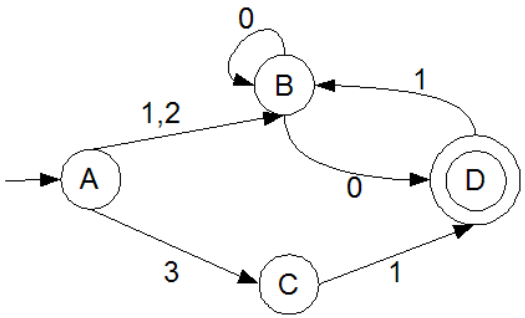
Find the minimal word of length 4 in the NFA.



states/length	1	2	3	4
A	x	10	100	
B	0	00	000	
C	1	x	110	
D	x	10	100	

Mäkinen's Minimal Word Algorithm

Find the minimal word of length 4 in the NFA.



states/length	1	2	3	4
A	x	10	100	1000
B	0	00	000	0000
C	1	x	110	1100
D	x	10	100	1000

Mäkinen's Minimal Word Algorithm

- Finding the minimal word using Mäkinen's algorithm costs $O(n)$ and uses $O(n)$ space in the *unit-cost model* and when the size of the NFA is constant.

Mäkinen's Minimal Word Algorithm

- Finding the minimal word using Mäkinen's algorithm costs $O(n)$ and uses $O(n)$ space in the *unit-cost model* and when the size of the NFA is constant.
- We analyze the algorithm in the *bit-complexity model* and integrate other parameters into our analysis.

Mäkinen's Minimal Word Algorithm

- Finding the minimal word using Mäkinen's algorithm costs $O(n)$ and uses $O(n)$ space in the *unit-cost model* and when the size of the NFA is constant.
- We analyze the algorithm in the *bit-complexity model* and integrate other parameters into our analysis.
- Concatenation of words can be performed in constant time by changing the mode of storage: Instead of storing a word w of length i in $A^{min}[i]$, store the pair (a, B) such that $w = aB^{min}[i-1]$.

states/length	1	2	3	4
A	x	(1, B)	(1, B)	(1, B)
B	0	(0, B)	(0, B)	(0, B)
C	1	x	x	x
D	x	(1, B)	(1, B)	(1, B)

- With this modification, Mäkinen's algorithm uses $\Theta(sn)$ space to find the minimal word.

Mäkinen's Minimal Word Algorithm

- Since the states of an NFA can form a complete graph, the worst-case running time is $O(s^2 n^2)$.

Mäkinen's Minimal Word Algorithm

- Since the states of an NFA can form a complete graph, the worst-case running time is $O(s^2 n^2)$.
- This worst-case is reached in the figure below.

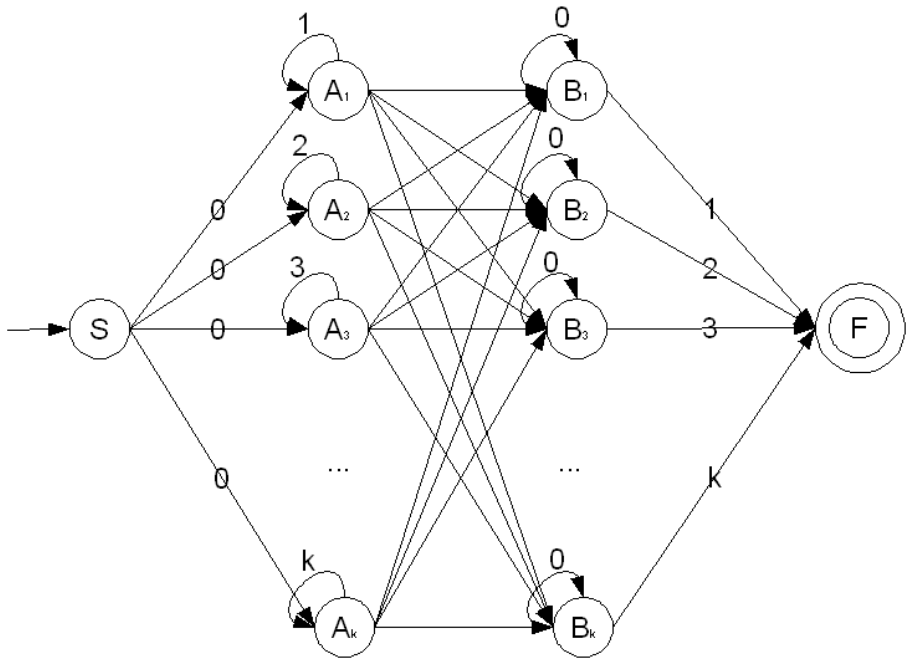


Figure: $\delta(A_i, a_i) = \{B_1, B_2, \ldots, B_k\}$ for all distinct a_i.

Mäkinen's Cross-Section Enumeration Algorithms

- Mäkinen's original cross-section enumeration algorithm requires subset-construction to work on NFAs.

Mäkinen's Cross-Section Enumeration Algorithms

- Mäkinen's original cross-section enumeration algorithm requires subset-construction to work on NFAs.
- To overcome this constraint, find consecutive words in a cross-section using the same method as in the Lookahead-Matrix algorithm.

Mäkinen's Cross-Section Enumeration Algorithms

- Mäkinen's original cross-section enumeration algorithm requires subset-construction to work on NFAs.
- To overcome this constraint, find consecutive words in a cross-section using the same method as in the Lookahead-Matrix algorithm.
- Mäkinen's original cross-section algorithm determines when a cross-section has been fully enumerated by precomputing the maximal word in the cross-section.

Mäkinen's Cross-Section Enumeration Algorithms

- Mäkinen's original cross-section enumeration algorithm requires subset-construction to work on NFAs.
- To overcome this constraint, find consecutive words in a cross-section using the same method as in the Lookahead-Matrix algorithm.
- Mäkinen's original cross-section algorithm determines when a cross-section has been fully enumerated by precomputing the maximal word in the cross-section.
- Recall that Lookahead-Matrix determines that a cross-section has been enumerated when the state stack is empty.

Mäkinen's Cross-Section Enumeration Algorithms

- When Mäkinen's original cross-section termination method is used, we call the algorithm `crossSectionMäkinenI`.

Mäkinen's Cross-Section Enumeration Algorithms

- When Mäkinen's original cross-section termination method is used, we call the algorithm `crossSectionMäkinenI`.
- When the Lookahead-Matrix cross-section termination is used on Mäkinen's algorithm, we call the algorithm `crossSectionMäkinenII`.

Mäkinen's Cross-Section Enumeration Algorithms

- When Mäkinen's original cross-section termination method is used, we call the algorithm `crossSectionMäkinenI`.
- When the Lookahead-Matrix cross-section termination is used on Mäkinen's algorithm, we call the algorithm `crossSectionMäkinenII`.
- Both `crossSectionMäkinenI` and `crossSectionMäkinenII` use $O(s^2n^2 + \sigma s^2 t)$ operations.
  t: output size. s: number of states. σ: alphabet size.

Mäkinen's Cross-Section Enumeration Algorithms

- When Mäkinen's original cross-section termination method is used, we call the algorithm `crossSectionMäkinenI`.
- When the Lookahead-Matrix cross-section termination is used on Mäkinen's algorithm, we call the algorithm `crossSectionMäkinenII`.
- Both `crossSectionMäkinenI` and `crossSectionMäkinenII` use $O(s^2n^2 + \sigma s^2 t)$ operations.
  t: output size. s: number of states. σ: alphabet size.
- In practice, `crossSectionMäkinenI` and `crossSectionMäkinenII` perform differently.

Mäkinen's Enumeration Algorithm

- The algorithm `enumMäkinenI` consists of repeated calls to `crossSectionMäkinenI`.

Mäkinen's Enumeration Algorithm

- The algorithm `enumMäkinenI` consists of repeated calls to `crossSectionMäkinenI`.
- Similarly, `enumMäkinenII` consists of repeated calls to `crossSectionMäkinenII`.

Mäkinen's Enumeration Algorithm

- The algorithm `enumMäkinenI` consists of repeated calls to `crossSectionMäkinenI`.
- Similarly, `enumMäkinenII` consists of repeated calls to `crossSectionMäkinenII`.
- Both algorithms use $O(\sigma s^2 t + s^2 e)$ operations.
  e: number of empty cross-sections. t: output size.
  s: number of states. σ: alphabet size.

Complexity Summary

	Cross-Section	Enum
Lookahead-Matrix	$O(s^{2.376}n + \sigma s^2 t)$	$O(s^{2.376}(c+e) + \sigma s^2 t)$
Mäkinen	$O(s^2 n^2 + \sigma s^2 t)$	$O(s^2 e + \sigma s^2 t)$
Grail	$O(s^2 \sigma^{n+1})$	$O(s^2 \sigma^{k+1})$

e: number of empty cross-sections. c: number of non-empty cross-sections.
t: output size. s: number of states. σ: alphabet size.
n: length of words in cross-section. k: length of words in last cross-section examined.

Experimental Framework

- We ran tests to determine the algorithms' average performance in practice.

Experimental Framework

- We ran tests to determine the algorithms' average performance in practice.
- For each test, 100 NFAs were randomly generated.

Experimental Framework

- We ran tests to determine the algorithms' average performance in practice.
- For each test, 100 NFAs were randomly generated.
- For each NFA, we ran each algorithm up to 10 times and recorded the average running time.

Experimental Framework

- We ran tests to determine the algorithms' average performance in practice.
- For each test, 100 NFAs were randomly generated.
- For each NFA, we ran each algorithm up to 10 times and recorded the average running time.
- To test worst-case practical performance, we also tested the algorithms on the NFA that accepts 1^*, the NFA that accepts $(0+1)^*$, and NFAs that cause Mäkinen's cross-section algorithms to run in quadratic time.

Experimental Framework

- We ran tests to determine the algorithms' average performance in practice.
- For each test, 100 NFAs were randomly generated.
- For each NFA, we ran each algorithm up to 10 times and recorded the average running time.
- To test worst-case practical performance, we also tested the algorithms on the NFA that accepts 1^*, the NFA that accepts $(0+1)^*$, and NFAs that cause Mäkinen's cross-section algorithms to run in quadratic time.
- For comparison and correctness testing, we implemented a naive algorithm that generates all words over Σ^* and checks which are accepted by the input NFA.

Experimental Results

- The naive algorithms perform reasonably well on small NFAs when the alphabet is of size less than 3, but usually slower than the other algorithms.
- With an alphabet size greater than 3, the naive algorithms are unreasonably slow.

Experimental Results

- The naive algorithms perform reasonably well on small NFAs when the alphabet is of size less than 3, but usually slower than the other algorithms.
- With an alphabet size greater than 3, the naive algorithms are unreasonably slow.
- The Grail algorithms tend to perform well on small input size.
- The Grail algorithms outperform the other enumeration algorithms on 1^*.

Experimental Results

- The naive algorithms perform reasonably well on small NFAs when the alphabet is of size less than 3, but usually slower than the other algorithms.
- With an alphabet size greater than 3, the naive algorithms are unreasonably slow.
- The Grail algorithms tend to perform well on small input size.
- The Grail algorithms outperform the other enumeration algorithms on 1^*.
- Naive and Grail algorithms are significantly slower than the Lookahead-Matrix algorithm and Mäkinen's algorithms on most NFAs.

Experimental Results

- Mäkinen's algorithms and lookahead-matrix are poorly suited for sparse languages, like 1^*.

Experimental Results

- Mäkinen's algorithms and lookahead-matrix are poorly suited for sparse languages, like 1^*.
- MäkinenII is significantly more efficient than MäkinenI on NFAs with unary alphabets.

Experimental Results

- Mäkinen's algorithms and lookahead-matrix are poorly suited for sparse languages, like 1^*.
- MäkinenII is significantly more efficient than MäkinenI on NFAs with unary alphabets.
- On NFAs where Mäkinen's cross-section enumeration algorithms are quadratic in n, `crossSectionLM` performs significantly better than Mäkinen's cross-section algorithms (at times over 50 times faster).

Experimental Results

- Mäkinen's algorithms and lookahead-matrix are poorly suited for sparse languages, like 1^*.
- MäkinenII is significantly more efficient than MäkinenI on NFAs with unary alphabets.
- On NFAs where Mäkinen's cross-section enumeration algorithms are quadratic in n, `crossSectionLM` performs significantly better than Mäkinen's cross-section algorithms (at times over 50 times faster).
- On average, the Matrix-Lookahead algorithms perform almost as well as the MäkinenII algorithms and better than the MäkinenI algorithms.
- On average, MäkinenII performs best.

Conclusions

- The algorithm `crossSectionLM` has the best worst-case complexity and the best worst-case running time in practice.

Conclusions

- The algorithm `crossSectionLM` has the best worst-case complexity and the best worst-case running time in practice.
- The MäkinenII algorithms for both enumeration problems have the best average-case running times in practice.

Future Work

- Improve on the running time of enumMäkinen for the enumeration problem.

Future Work

- Improve on the running time of enumMäkinen for the enumeration problem.
- Find heuristics to further improve the running time of the algorithms in practice. For instance, check the density of the language and select algorithm based on the density.

Future Work

- Improve on the running time of enumMäkinen for the enumeration problem.
- Find heuristics to further improve the running time of the algorithms in practice. For instance, check the density of the language and select algorithm based on the density.
- Prove lower bounds for the enumeration and cross-section enumeration problems.