Outline

- Introduction
- How to execute a MA Schema
- Examples
- Formal Definition
- Formal Algorithm
- Other Examples
Andrey A. Markov (1950s): Russian mathematician.

MA is equivalent to the Turing Machine TM.

MA can simulate the Universal Turing Machine U.

Accept or Reject.

MA can compute the set of computable function.

May return a value.

MA is based on rewrite rules.

Ordered list of rules for rewriting strings.
Terminology

❖ **Markov Algorithm:**
  ➢ The substitution process described by the production rules.

❖ **Markov Algorithm Schema:**
  ➢ The set of production rules themselves.
  ➢ A set of rules for rewriting strings.
How to execute a MA Schema

- Apply the **first rule** whose **LHS** matches a substring of the current string.
  - (Repeat, until the procedure halts.)
  - Apply the rule at the **leftmost** position where it matches.
- If **no** rule is applicable, **halt**.
- If the **RHS** of the applied rule contains a **dot (.)**
  - **halt** after applying that rule.
**Example 1:**

Sort a string so that its letters appear in alphabetical order, $\Sigma = \{a, b, c, d\}$.

E.g., $cbadb \rightarrow^{*} abbcd$

(i) $ba \rightarrow ab$
(ii) $ca \rightarrow ac$
(iii) $da \rightarrow ad$
(iv) $cb \rightarrow bc$
(v) $db \rightarrow bd$
(vi) $dc \rightarrow cd$

Application to $cbadb$:

- $cbadb$ $\rightarrow$ $bcadtb$
- $bacdcb$ $\rightarrow$ $abcdcb$
- $abcdcb$ $\rightarrow$ $abcdb$
- $abcdb$ $\rightarrow$ $abcbd$
- $abcbd$ $\rightarrow$ $abbcdb$
- $abbcdb$ $\rightarrow$ $abbbcd$
Duplicate each letter in a string. $\Sigma = \{a, b, c\}$.
E.g., $\text{acbb} \rightarrow * \text{aacchbblbb}$

(i) $a \rightarrow aa$
(ii) $b \rightarrow bb$
(iii) $c \rightarrow cc$

Bad first draft! Why?
Problem: Never Halts:
$\text{acbb} \rightarrow \text{aacbb} \rightarrow \text{aaaacbblbb} \rightarrow \ldots$
Second draft: Use a placeholder, $\Sigma = \{a, b, c, \#\}$.

(i) $\#a \rightarrow aa\#$
(ii) $\#b \rightarrow bb\#$
(iii) $\#c \rightarrow cc\#$
(iv) $\# \rightarrow \varepsilon \varepsilon$
(v) $\varepsilon \rightarrow \#$

Application to $acbb$:
- $acbb$
- $\#acbb$
- $aa\#cbb$
- $aacc\#bb$
- $aaccbb\#b$
- $aaccbbbb\#$
- $aaccbbbb$
MA is a triple \(<V, \Sigma, R>\), where:

- \(V\): is the rule alphabet, which contains both working symbols and input symbols (Accept, Reject).
- \(\Sigma\): is the set of input symbols (subset of \(V\)).
- \(R\): is an ordered list of rules (Rules), continuing and terminating rules.

- we will write continuing rules as \((X \rightarrow Y)\).
- if a terminating rule is applied, the algorithm halts as \((X \rightarrow \bullet Y)\).
Important Notices

- No start symbol
- The job of the algorithm is to examine an input string and return the appropriate result.
- MA is completely deterministic.
- Main focus on function computation and language acceptance.
- Prolog language executes program in the same way of the MA concepts.
Formal Algorithm

**Markov algorithm** (*M*: Markov algorithm, *w*: input) =

1. Until *no* rules apply or the process has been terminated by executing a terminal rule do:
   1.1. Find the **first rule** in the list *R* that matches against *w*.
      1.1.1. If that rule matches *w* in more than one place, choose the **leftmost** match.
   1.2. If *no* rule matches then **exit**.
   1.3. Apply the matched rule to *w* by replacing the **substring** that matched the rule’s **LHS** with rule’s **RHS**.
   1.4. If the matched rule is a terminating rule, **exit**.
2. If *w* contains the symbol **Accept** then accept.
3. If *w* contains the symbol **Reject** then reject.
4. Otherwise, return *w*. 
Example 3: (MA as language acceptors)

**Transform** input string into 1 (to accept);
Any other actions (including not halting) constitute non-acceptance.

*E.g.,* $a^*bc^*$. Let $M = \langle \{a, b, c, \#, \$, 1\}, \{a, b, c\}, R \rangle$

where $R =$

1. $\#a \rightarrow \#$
2. $\#b \rightarrow \$
3. $\$c \rightarrow \$
4. $\$ \rightarrow .1$
5. $\varepsilon \rightarrow \#$
Example 3: cont..

Tests of algorithm:

aaabcc
#aaabcc
#aabcc
#abcc
#bcc
$cc
$c
$

But

aac
#aac
#ac
#c
###c

abbc
#abbc
#bbc
$bc
1bc
Example 4: MA as language recognizers

Transform input string into 1 (to accept) or 0 (to reject).

E.g., \( L = \{ a^n b^n \mid n \geq 0 \} \), \( \Sigma = \{a, b\} \), and \( R = \)

1. \( ba \rightarrow 0 \) /* **Reject** if any symbols are out of order
2. \( a0 \rightarrow 0 \) /* Once the string has been rejected,
3. \( 0a \rightarrow 0 \) /* **Erase** other symbols
4. \( b0 \rightarrow 0 \)
5. \( 0b \rightarrow 0 \)
6. \( ab \rightarrow \varepsilon \) /* **Erase** ab pairs in the middle of the string
7. \( a \rightarrow 0 \) /* If any extra \( a \)s or \( b \)s, **reject**
8. \( b \rightarrow 0 \)
9. \( 0 \rightarrow .0 \)
10. \( \varepsilon \rightarrow 1 \) /* **Accept** if erasing \( ab \) erases whole string
Example 4: cont...

Tests of algorithm:

Accept:
- aaabbb
- aabb
- ab
- $\varepsilon$
- 1

Reject:
- $aabb$
- $ab$
- $b$
- 0
- 0

Reject:
- $aaa$
- $0aa$
- $0a$
- 0
- 0
We show a MA to decide the language $A^nB^nC^n = \{ a^n b^n c^n \mid n \geq 0 \}$. Let $M = < \{a, b, c, \#, \%, \?, \text{Accept, Reject}\}, \{a,b,c\}, R>$, where $R =$

1. $\#a \rightarrow \%$ /* If the first char. = $a$, erase it and look for a $b$ next.
2. $\#b \rightarrow \text{Reject}$ /* if the first char. = $b$, reject.
3. $\#c \rightarrow \text{Reject}$ /* if the first char. = $c$, reject.
4. $\%a \rightarrow a\%$ /* Move the $\%$ past the $a$’s until it finds a $b$.
5. $\%b \rightarrow ?$ /* If it finds a $b$, erase it and look for a $c$ next.
6. $\% \rightarrow \text{Reject}$ /* No $b$ found. Just $c$’s or end of string. Reject.
7. $?b \rightarrow b?$ /* Move the ? past the $b$’s until it finds a $c$.
8. $?c \rightarrow \varepsilon$ /* If it finds a $c$. Then only rule11 can apply next.
9. $? \rightarrow \text{Reject}$ /* No $c$ found. Just $a$’s or $b$’s or end of string. Reject.
10. $\# \rightarrow \text{Accept}$ /* # was created but No input char. left. Accept.
11. $\varepsilon \rightarrow \#$ /* This one goes first since none of the others can.
References