Accumulating Automata

Shlomi Dolev, Niv Giboa and Ximing Li

June 11, 2013
Ben Gurion University of The Negev

Email: dolev@cs.bgu.ac.il gilboan@cse.bgu.ac.il

ximing@post.bgu.ac.il

Motivation: Cloud computing and private storage

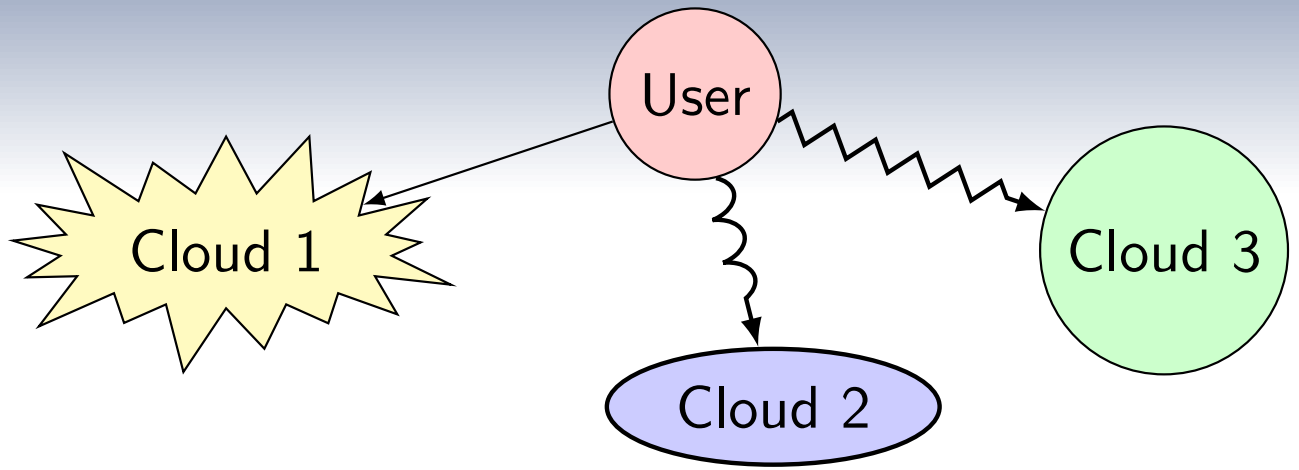

Figure: Cloud computing

Motivation: Cloud computing and private storage

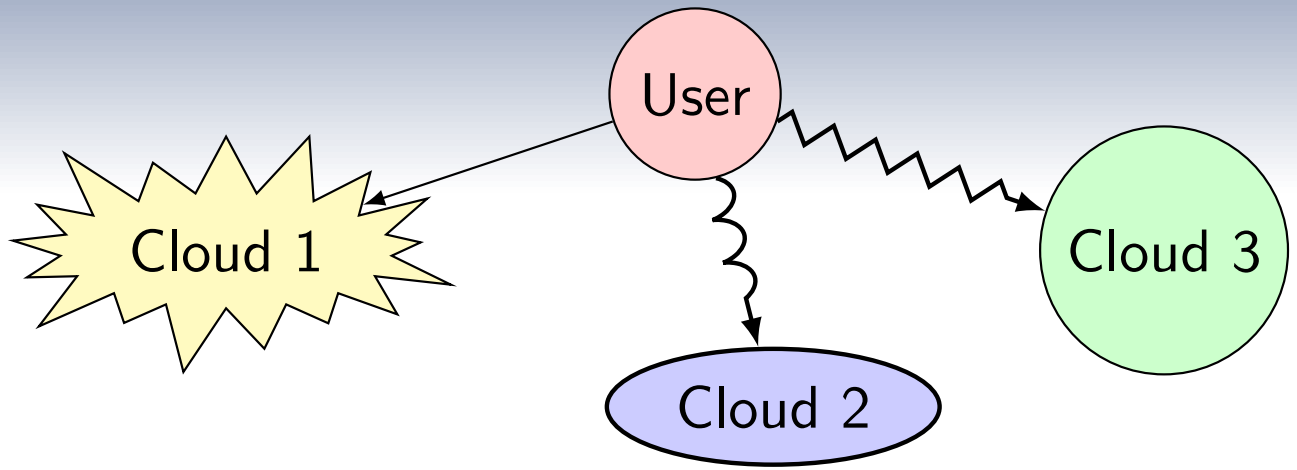

Figure: Cloud computing

Private data storage and computing

Cloud computing model can not really protect user's privacy.

Motivation: Fully homomorphic encryption

Fully Homomorphic Encryption (FHE)

FHE is not good enough.

- Very inefficient in that their per-gate computation;

Gentry, C. (2009). A fully homomorphic encryption scheme (Doctoral dissertation, Stanford University).

Motivation: Fully homomorphic encryption

Fully Homomorphic Encryption (FHE)

FHE is not good enough.

- Very inefficient in that their per-gate computation;
- Only computationally secure.

Gentry, C. (2009). A fully homomorphic encryption scheme (Doctoral dissertation, Stanford University).

Motivation: Multi-Party Computation

Secure Multi-Party Computation (MPC)

A powerful concept in secure distributed computing.

Motivation: Multi-Party Computation

Secure Multi-Party Computation (MPC)

A powerful concept in secure distributed computing.

The goal of secure MPC

Enable a set of m mutually distrusting parties to jointly and securely compute a function f of their private inputs

Even in the presence of an active adversary *Adv*.

Motivation: Multi-Party Computation

Secure Multi-Party Computation (MPC)

A powerful concept in secure distributed computing.

The goal of secure MPC

Enable a set of m mutually distrusting parties to jointly and securely compute a function f of their private inputs

Even in the presence of an active adversary *Adv*.

Secure MPC can be implemented

Based on (1) some mathematical assumption, such as factoring; (2) perfectly information theoretically secure secret sharing schemes, such as Shamir's linear secret sharing based on polynomials.

Normal MPC based on Shamir's secret sharing scheme

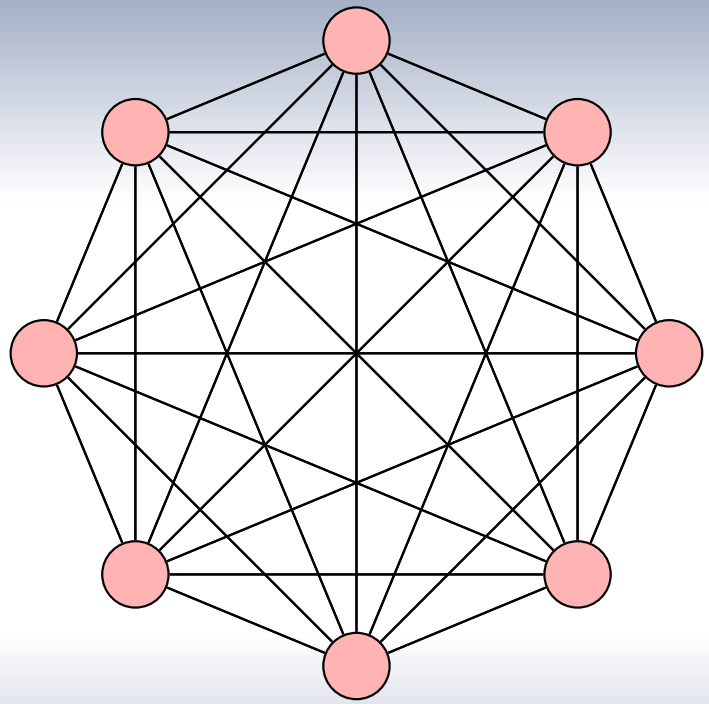

Figure: Communication for MPC

Normal MPC based on Shamir's secret sharing scheme

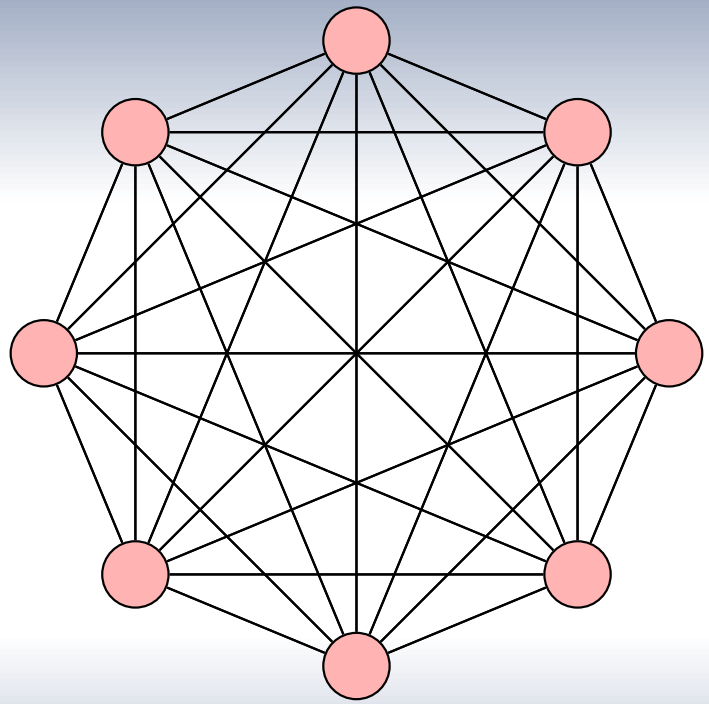

Figure: Communication for MPC

Each channel is either private or secured, computationally secured

Our contribution: Setting

We are concerned with

Communicationless information theoretically secure multi-party computation over Practically infinite input streams;

Our Setting

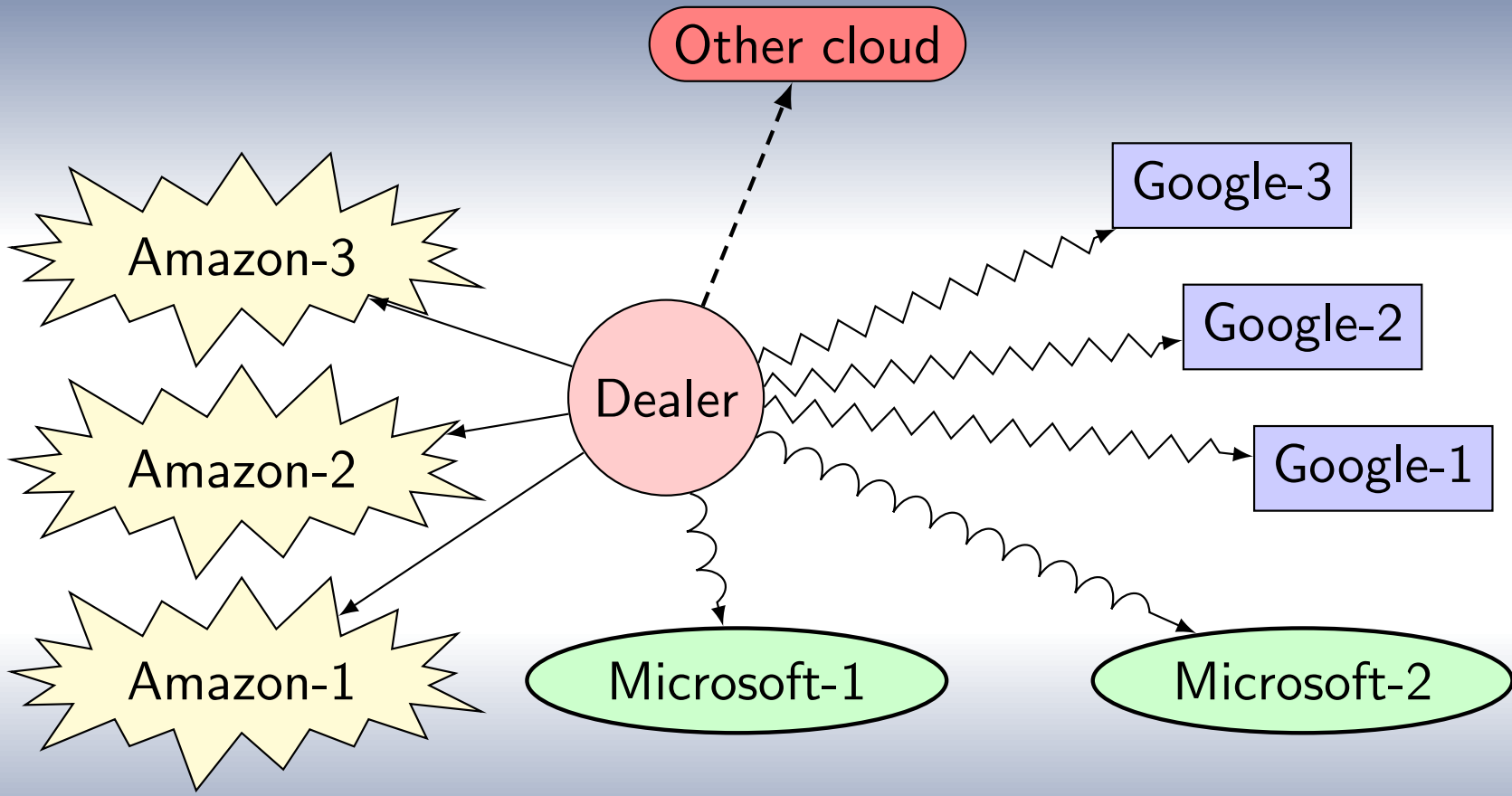

Figure: Communicationless MPC among different cloud suppliers

Our contribution: Setting

We assume

A stateless dealer $\mathcal{D}$, the dealer may temporarily store the input to the system, process the input and send secret shares of the inputs to the participants.

Our contribution: Setting

We assume

A stateless dealer $\mathcal{D}$, the dealer may temporarily store the input to the system, process the input and send secret shares of the inputs to the participants.

No participant returns any information back.

Our contribution: Setting

We assume

A stateless dealer $\mathcal{D}$, the dealer may temporarily store the input to the system, process the input and send secret shares of the inputs to the participants.

No participant returns any information back.

At any point in the execution

The dealer may ask some participants to send their results back, then the dealer can reconstruct the actual result of the algorithm.

Our contribution: Detail

Scheme

- The dealer uses m cloud servers or agents $\mathcal{P}_1, \ldots, \mathcal{P}_m$ which perform a computation over the input stream received from $\mathcal{D}$;

Our contribution: Detail

Scheme

- The dealer uses m cloud servers or agents $\mathcal{P}_1, \ldots, \mathcal{P}_m$ which perform a computation over the input stream received from $\mathcal{D}$;
- The dealer $\mathcal{D}$ sends different input shares to every agent. Agents do not communicate with each other.

Our contribution: Detail

Scheme

- The dealer uses m cloud servers or agents $\mathcal{P}_1, \ldots, \mathcal{P}_m$ which perform a computation over the input stream received from $\mathcal{D}$;
- The dealer $\mathcal{D}$ sends different input shares to every agent. Agents do not communicate with each other.

Properties

- Any agent cannot learn anything about the original inputs that $\mathcal{D}$ partitions to shares;

Our contribution: Detail

Scheme

- The dealer uses m cloud servers or agents $\mathcal{P}_1, \ldots, \mathcal{P}_m$ which perform a computation over the input stream received from $\mathcal{D}$;
- The dealer $\mathcal{D}$ sends different input shares to every agent. Agents do not communicate with each other.

Properties

- Any agent cannot learn anything about the original inputs that $\mathcal{D}$ partitions to shares;
- At any given stage, the dealer $\mathcal{D}$ may collect the state of the agents and obtain the computation result.

Our Contribution

A new kind automaton

Accumulating automaton: for practically unbounded inputs;

Our Contribution

A new kind automaton

Accumulating automaton: for practically unbounded inputs;

Example of Accumulating automaton

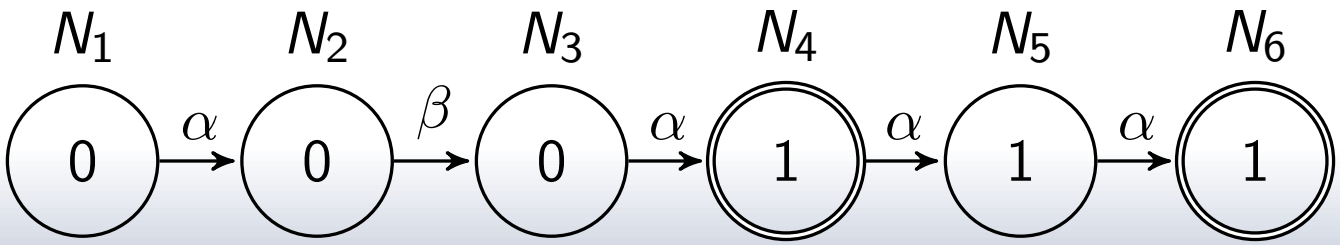

Figure: Accumulating automaton

Accumulating automaton

Definition

An accumulating automaton is a triple $\mathcal{A} = (V, \Sigma, T)$ where:

- V is a set of variables, *nodes*. regular or accumulating;

Accumulating automaton

Definition

An accumulating automaton is a triple $\mathcal{A} = (V, \Sigma, T)$ where:

- V is a set of variables, *nodes*. regular or accumulating;
- Σ is the alphabet;

Accumulating automaton

Definition

An accumulating automaton is a triple $\mathcal{A} = (V, \Sigma, T)$ where:

- V is a set of variables, *nodes*. regular or accumulating;
- Σ is the alphabet;
- T is a set of transitions. A triple $(p_1, \alpha, p_2) \in V \times \Sigma \to V$.

Accumulating automaton

Definition

An accumulating automaton is a triple $\mathcal{A} = (V, \Sigma, T)$ where:

- V is a set of variables, *nodes*. regular or accumulating;
- Σ is the alphabet;
- T is a set of transitions. A triple $(p_1, \alpha, p_2) \in V \times \Sigma \to V$.

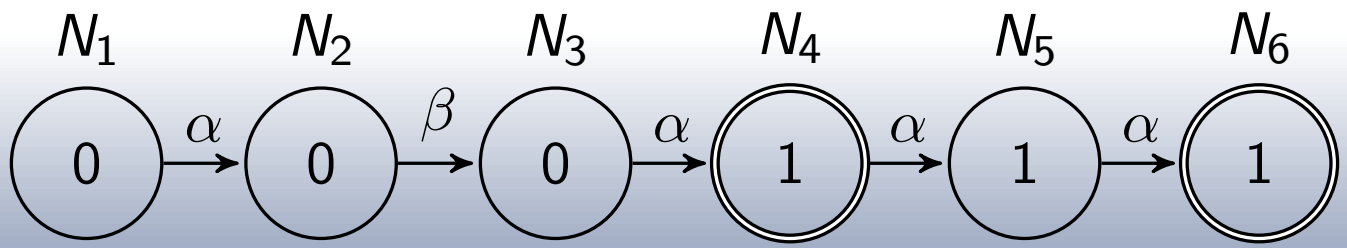

Figure: Accumulating automaton

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;
- String matching algorithm over directed graph;

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;
- String matching algorithm over directed graph;
- General string matching;

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;
- String matching algorithm over directed graph;
- General string matching;

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;
- String matching algorithm over directed graph;
- General string matching;
- Dag Accumulating Automata (DAA);

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;
- String matching algorithm over directed graph;
- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;
- String matching algorithm over directed graph;
- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;
  - Recognizing the regular language $(\alpha\beta\alpha)^*$;

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;
- String matching algorithm over directed graph;
- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;
  - Recognizing the regular language $(\alpha\beta\alpha)^*$;
  - Recognizing the regular language $\alpha(\alpha\beta\alpha)^*$;

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;
- String matching algorithm over directed graph;
- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;
  - Recognizing the regular language $(\alpha\beta\alpha)^*$;
  - Recognizing the regular language $\alpha(\alpha\beta\alpha)^*$;
  - Recognizing the context free language $\alpha^s\beta^s$;

Accumulating automaton: Outline

Outline of accumulating automaton

- String matching, an example;
- String matching algorithm over directed graph;
- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;
  - Recognizing the regular language $(\alpha\beta\alpha)^*$;
  - Recognizing the regular language $\alpha(\alpha\beta\alpha)^*$;
  - Recognizing the context free language $\alpha^s\beta^s$;
  - Recognizing the context sensitive language $\alpha^s\beta^s\gamma^s$.

String matching, an example.

PATTERN $L\ O\ V\ E$

$\downarrow$

TEXT $A\ L\ I\ C\ E \quad L\ O\ V\ E\ S \quad B\ O\ B$

Figure: String matching example

String matching algorithm over directed graph.

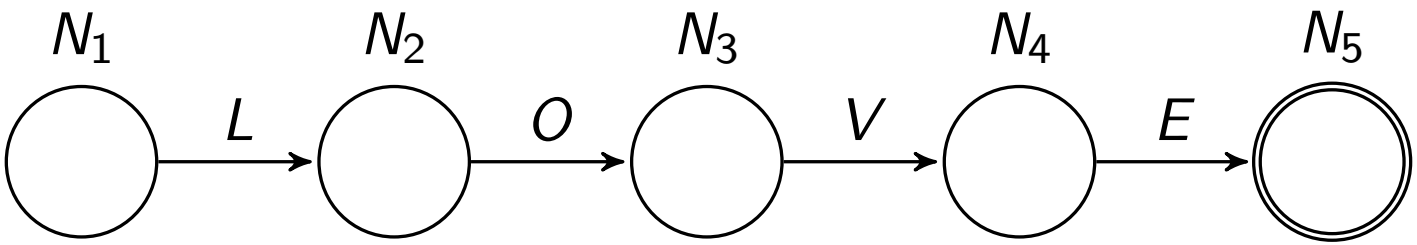

Figure: String matching algorithm over directed graph $\mathbb{G}$

Initial marking of the graph

Initial stage

N_1 is initially set to 1. N_2 to N_5 are initially set to 0. Namely,

$$N_1^{(0)} = 1, N_2^{(0)} = N_3^{(0)} = N_4^{(0)} = N_5^{(0)} = 0$$

Initial marking of the graph

Initial stage

N_1 is initially set to 1. N_2 to N_5 are initially set to 0. Namely,

$$N_1^{(0)} = 1, N_2^{(0)} = N_3^{(0)} = N_4^{(0)} = N_5^{(0)} = 0$$

String matching algorithm over directed graph $\mathbb{G}$

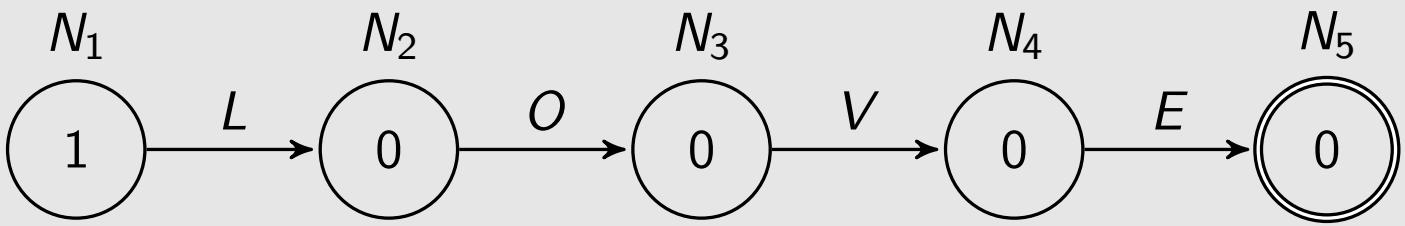

Figure: String matching algorithm over directed graph $\mathbb{G}$

Execution of the string matching algorithm

For each input symbol, define an input vector $\vec{v}$.

- If the input symbol is 'A' the vector $\vec{v} = (0, 0, 0, 0)$;

Execution of the string matching algorithm

For each input symbol, define an input vector $\vec{v}$.

- If the input symbol is 'A' the vector $\vec{v} = (0, 0, 0, 0)$;
- If the input symbol is 'L' the vector $\vec{v} = (1, 0, 0, 0)$;

Execution of the string matching algorithm

For each input symbol, define an input vector $\vec{v}$.

- If the input symbol is 'A' the vector $\vec{v} = (0, 0, 0, 0)$;
- If the input symbol is 'L' the vector $\vec{v} = (1, 0, 0, 0)$;
- If the input symbol is 'O' the vector $\vec{v} = (0, 1, 0, 0)$;

Execution of the string matching algorithm

For each input symbol, define an input vector $\vec{v}$.

- If the input symbol is 'A' the vector $\vec{v} = (0, 0, 0, 0)$;
- If the input symbol is 'L' the vector $\vec{v} = (1, 0, 0, 0)$;
- If the input symbol is 'O' the vector $\vec{v} = (0, 1, 0, 0)$;
- If the input symbol is 'C' the vector $\vec{v} = (0, 0, 0, 0)$;

Execution of the string matching algorithm

For each input symbol, define an input vector $\vec{v}$.

- If the input symbol is 'A' the vector $\vec{v} = (0, 0, 0, 0)$;
- If the input symbol is 'L' the vector $\vec{v} = (1, 0, 0, 0)$;
- If the input symbol is 'O' the vector $\vec{v} = (0, 1, 0, 0)$;
- If the input symbol is 'C' the vector $\vec{v} = (0, 0, 0, 0)$;

Execution of the string matching algorithm

$$N_2^{(i+1)} = N_1^{(i)} \cdot v_1 \quad N_3^{(i+1)} = N_2^{(i)} \cdot v_2$$
$$N_4^{(i+1)} = N_4^{(i)} \cdot v_3 \quad N_5^{(i+1)} = N_5^{(i)} + N_4^{(i)} \cdot v_4$$

N_1 is initialized to be 1 and will keep unchanged.

Execution of the string matching algorithm

For the input symbol A.

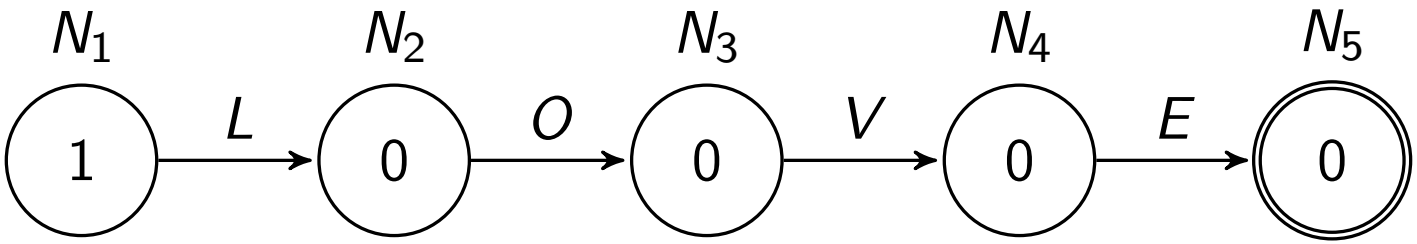

Figure: Marking of $\mathbb{G}$

Execution of the string matching algorithm

For the input symbols AL.

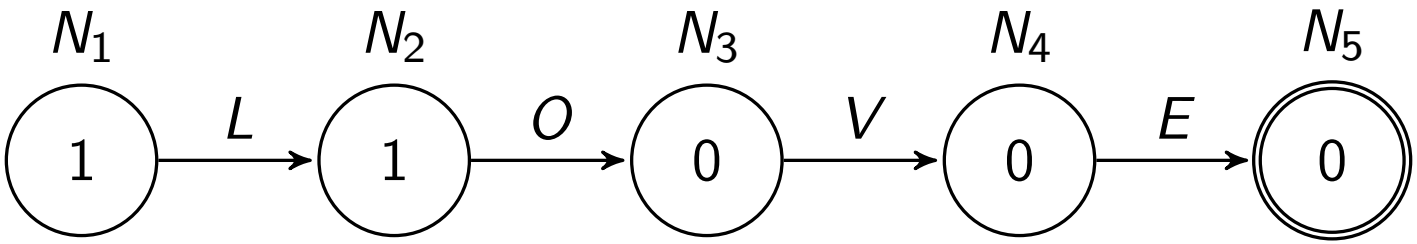

Figure: Marking of $\mathbb{G}$

Execution of the string matching algorithm

For the input symbols *ALI*.

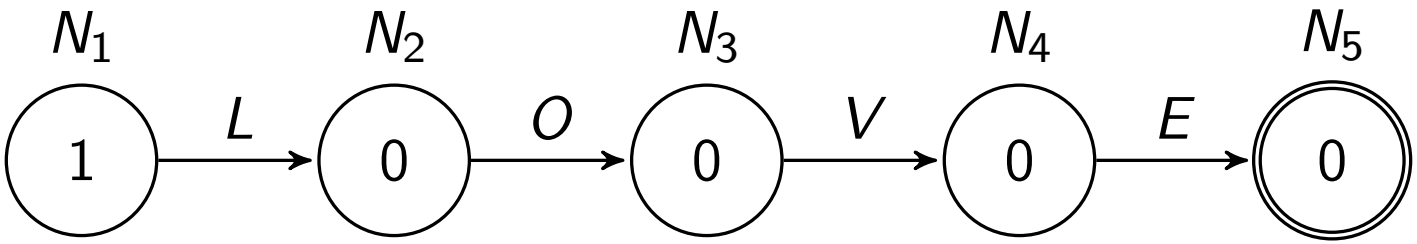

Figure: Marking of $\mathbb{G}$

Execution of the string matching algorithm

For the input symbols *ALICE*.

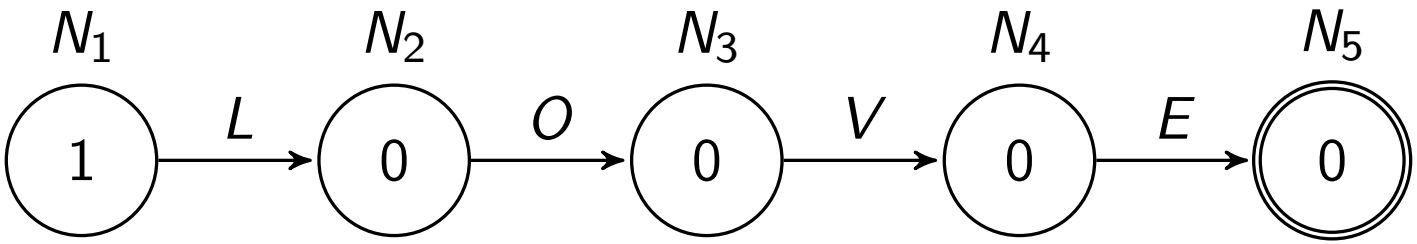

Figure: Marking of $\mathbb{G}$

Execution of the string matching algorithm

For the input stream *ALICE L*.

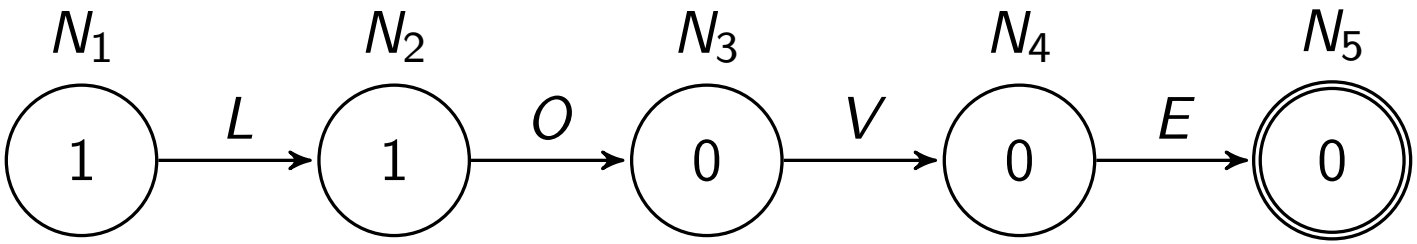

Figure: Marking of $\mathbb{G}$

Execution of the string matching algorithm

For the input stream *ALICE LO*.

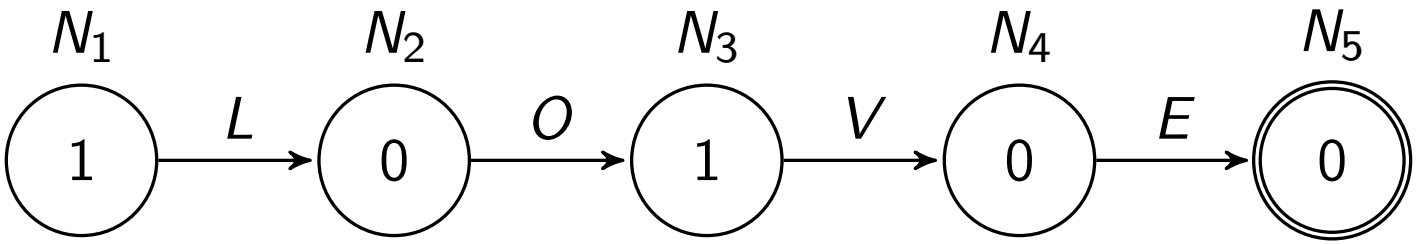

Figure: Marking of $\mathbb{G}$

Execution of the string matching algorithm

For the input stream *ALICE LOV*.

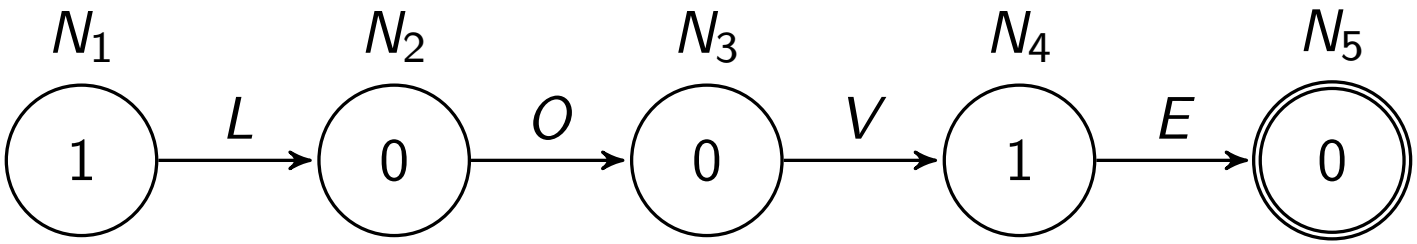

Figure: Marking of $\mathbb{G}$

Execution of the string matching algorithm

For the input stream *ALICE LOVES*.

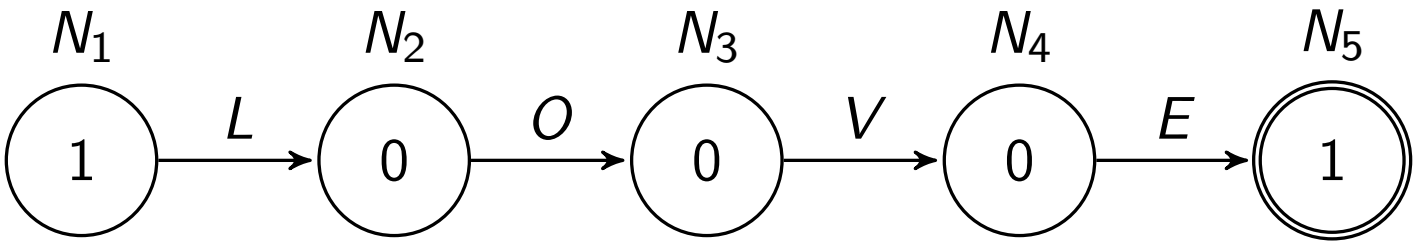

Figure: Marking of $\mathbb{G}$

Execution of the string matching algorithm

For the input stream *ALICE LOVES BOB*.

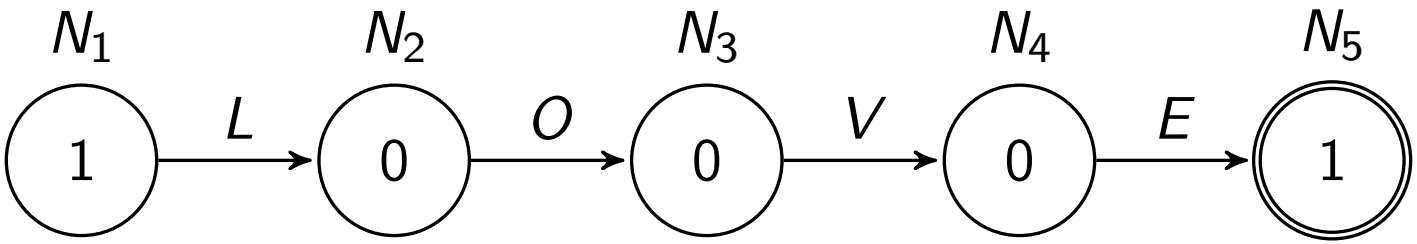

Figure: Marking of $\mathbb{G}$

Execution of the string matching algorithm

Execution of the string matching algorithm

- Each input value assigns a new (integer) value to every node;

Execution of the string matching algorithm

Execution of the string matching algorithm

- Each input value assigns a new (integer) value to every node;
- Define an input vector $\vec{v}$. Each element matches one corresponding element in the pattern;

Execution of the string matching algorithm

Execution of the string matching algorithm

- Each input value assigns a new (integer) value to every node;
- Define an input vector $\vec{v}$. Each element matches one corresponding element in the pattern;
- If the input character does not appear, $\vec{v}$ is set to $(0, 0, 0, 0)$;

Execution of the string matching algorithm

Execution of the string matching algorithm

- Each input value assigns a new (integer) value to every node;
- Define an input vector $\vec{v}$. Each element matches one corresponding element in the pattern;
- If the input character does not appear, $\vec{v}$ is set to $(0, 0, 0, 0)$;
- For any (v_1, v_2, v_3, v_4), the marking of nodes of the graph are simultaneously computed.

Result of the string matching algorithm

Gaining result

- $N_5 > 0$ means there is at least one match;

Result of the string matching algorithm

Gaining result

- $N_5 > 0$ means there is at least one match;
- The value of the node N_5 encodes the number of times that the pattern has occurred in the input stream.

Result of the string matching algorithm

Gaining result

- $N_5 > 0$ means there is at least one match;
- The value of the node N_5 encodes the number of times that the pattern has occurred in the input stream.

Assume that the number of occurrence does not exceed the maximal integer that the system can represent for N_5.

Communicationless secure private multi-party string matching protocol

MPC string matching protocol

- Initial stage: initializes all the participants;

Communicationless secure private multi-party string matching protocol

MPC string matching protocol

- Initial stage: initializes all the participants;
- Execution stage: sends shares to all participants, and each participant compute independently;

Communicationless secure private multi-party string matching protocol

MPC string matching protocol

- Initial stage: initializes all the participants;
- Execution stage: sends shares to all participants, and each participant compute independently;
- Collection stage: gets the shares back and reconstruct the result.

Communicationless secure private multi-party string matching protocol

Initial stage

For simplicity, assume there are six participants $\mathcal{P}_1, \ldots, \mathcal{P}_6$.

- Nodes' values of the vector are secret shared;

Communicationless secure private multi-party string matching protocol

Initial stage

For simplicity, assume there are six participants $\mathcal{P}_1, \ldots, \mathcal{P}_6$.

- Nodes' values of the vector are secret shared;
- Each participant $\mathcal{P}_i$ receives one share;

Communicationless secure private multi-party string matching protocol

Initial stage

For simplicity, assume there are six participants $\mathcal{P}_1, \ldots, \mathcal{P}_6$.

- Nodes' values of the vector are secret shared;
- Each participant $\mathcal{P}_i$ receives one share;
- The initial share of the node N_j that is maintained by the participants $\mathcal{P}_i$ by $S^{(0)}_{\mathcal{P}_i, N_j}$.

Communicationless secure private multi-party string matching protocol

Execution stage

- Each symbol α is mapped to an input vector $\vec{v}$. Each element in the input vector $\vec{v}$ is secret shared into six parts by a random polynomial of degree 1;

Communicationless secure private multi-party string matching protocol

Execution stage

- Each symbol α is mapped to an input vector $\vec{v}$. Each element in the input vector $\vec{v}$ is secret shared into six parts by a random polynomial of degree 1;
- Each share of the input vector is then sent to one of the participants;

Communicationless secure private multi-party string matching protocol

Execution stage

- Each symbol α is mapped to an input vector $\vec{v}$. Each element in the input vector $\vec{v}$ is secret shared into six parts by a random polynomial of degree 1;
- Each share of the input vector is then sent to one of the participants;
- Immediately after processing the k^{th} input symbol, the value of the (share of) node N_j that is stored by the participant $\mathcal{P}_i$ is denoted $S^{(k)}_{\mathcal{P}_i,N_j}$.

Communicationless secure private multi-party string matching protocol

Execution stage

Communicationless secure private multi-party string matching protocol

Execution stage

When the dealer sends a vector as follows

$$(S_{i,v_0}, S_{i,v1}, S_{i,v2}, S_{i,v3}, S_{i,v4})$$

$\mathcal{P}_i$ executes the following transitions:

$$\begin{aligned}
S^{(k+1)}_{\mathcal{P}_i,N_1} &= S_{i,v_0} \\
S^{(k+1)}_{\mathcal{P}_i,N_2} &= S^{(k)}_{\mathcal{P}_i,N_1} \cdot S_{i,v_1} \\
S^{(k+1)}_{\mathcal{P}_i,N_3} &= S^{(k)}_{\mathcal{P}_i,N_2} \cdot S_{i,v_2} \\
S^{(k+1)}_{\mathcal{P}_i,N_4} &= S^{(k)}_{\mathcal{P}_i,N_3} \cdot S_{i,v_3} \\
S^{(k+1)}_{\mathcal{P}_i,N_5} &= S^{(k)}_{\mathcal{P}_i,N_5} + S^{(k)}_{\mathcal{P}_i,N_4} \cdot S_{i,v_4}
\end{aligned}$$

Communicationless secure private multi-party string matching protocol

Collection stage

- Ask all the participants to send the value that corresponds to N_5 back;

Communicationless secure private multi-party string matching protocol

Collection stage

- Ask all the participants to send the value that corresponds to N_5 back;
- Construct the actual value of N_5 using Lagrange interpolation;

Communicationless secure private multi-party string matching protocol

Collection stage

- Ask all the participants to send the value that corresponds to N_5 back;
- Construct the actual value of N_5 using Lagrange interpolation;
- The value obtained indicates whether the search is successful or not.

Communicationless secure private multi-party string matching protocol

Analysis of the protocol

- The greatest value is associated with the node N_5 is bounded by the length of the input text;

Communicationless secure private multi-party string matching protocol

Analysis of the protocol

- The greatest value is associated with the node N_5 is bounded by the length of the input text;
- The participants do not know the inputs and the results during the entire execution;

Communicationless secure private multi-party string matching protocol

Analysis of the protocol

- The greatest value is associated with the node N_5 is bounded by the length of the input text;
- The participants do not know the inputs and the results during the entire execution;
- One can also secure the pattern by executing such string matching over all possible strings.

Matching several strings simultaneously

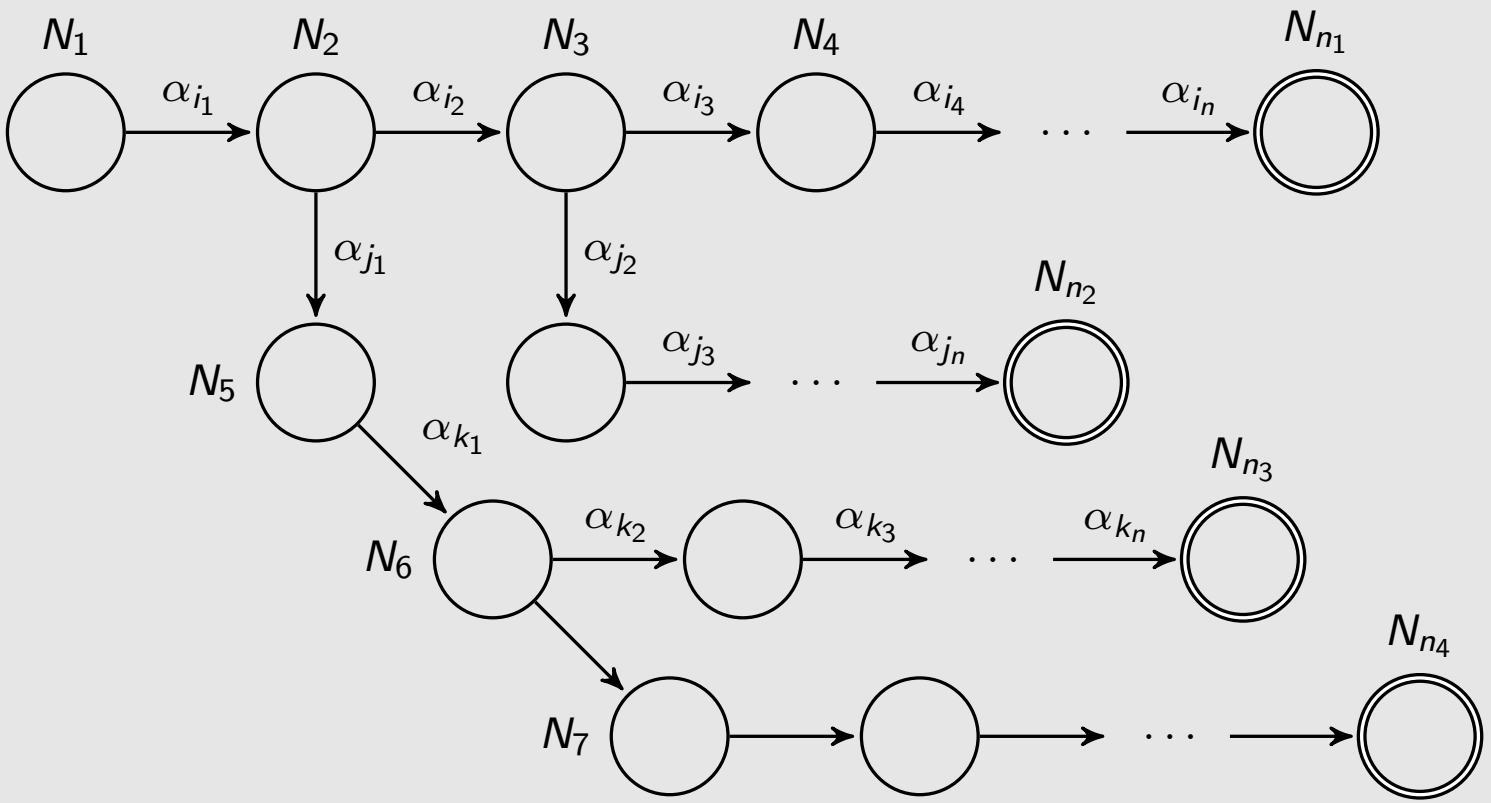

Figure: A directed graph for multiple string matching

String matching algorithm with a question wildcard in the pattern

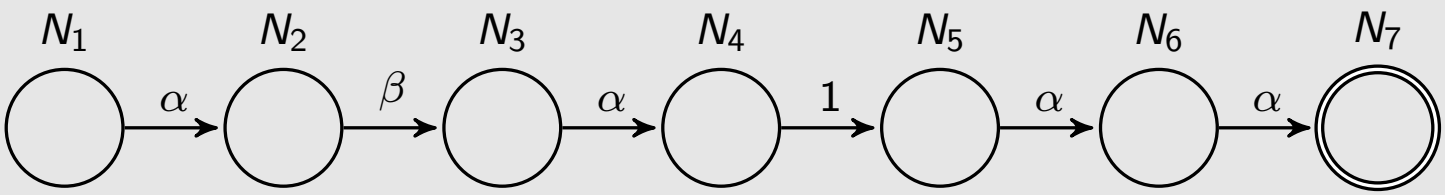

Figure: String matching algorithm with one wildcard "?" in the pattern $(\alpha\beta\alpha?\alpha\alpha)$

String matching algorithm with a question wildcard in the pattern

String matching algorithm with '?' in the pattern

For each node N_i, compute as follows

$$N_i^{(k+1)} = \begin{cases} v_0 & \text{if } i = 0 \\ N_{i-1}^{(k)} & \text{if the former edge is labeled by 1} \\ N_{i-1}^{(k)} \cdot v_i & \text{if } N_i \text{ is not an accumulating node;} \\ & \text{the former edge is not labeled by 1} \\ N_i^{(k)} + N_{i-1}^{(k)} \cdot v_i & \text{if } N_i \text{ is an accumulating node;} \\ & \text{the former edge is not labeled by 1} \end{cases}$$

String matching algorithm with a star wildcard in the pattern

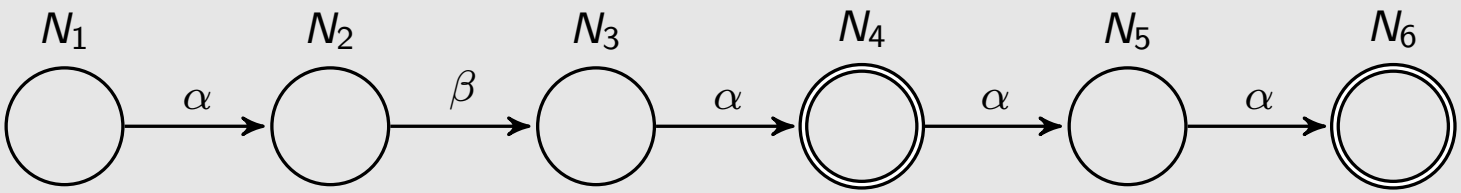

Figure: Algorithm with one wildcard "*" in the pattern $(\alpha\beta\alpha * \alpha\alpha)$

String matching algorithm with a star wildcard in the pattern

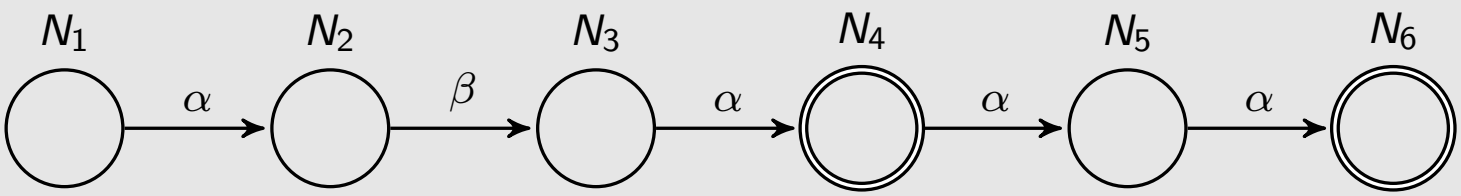

Figure: Algorithm with one wildcard "*" in the pattern $(\alpha\beta\alpha * \alpha\alpha)$

For each node N_i, compute as follows

$$N_i^{(k+1)} = \begin{cases} v_0 & \text{if } i = 0 \\ N_{i-1}^{(k)} \cdot v_i & \text{if } N_i \text{ is not an accumulating node} \\ N_i^{(k)} + N_{i-1}^{(k)} \cdot v_i & \text{if } N_i \text{ is an accumulating node} \end{cases}$$

Our Setting

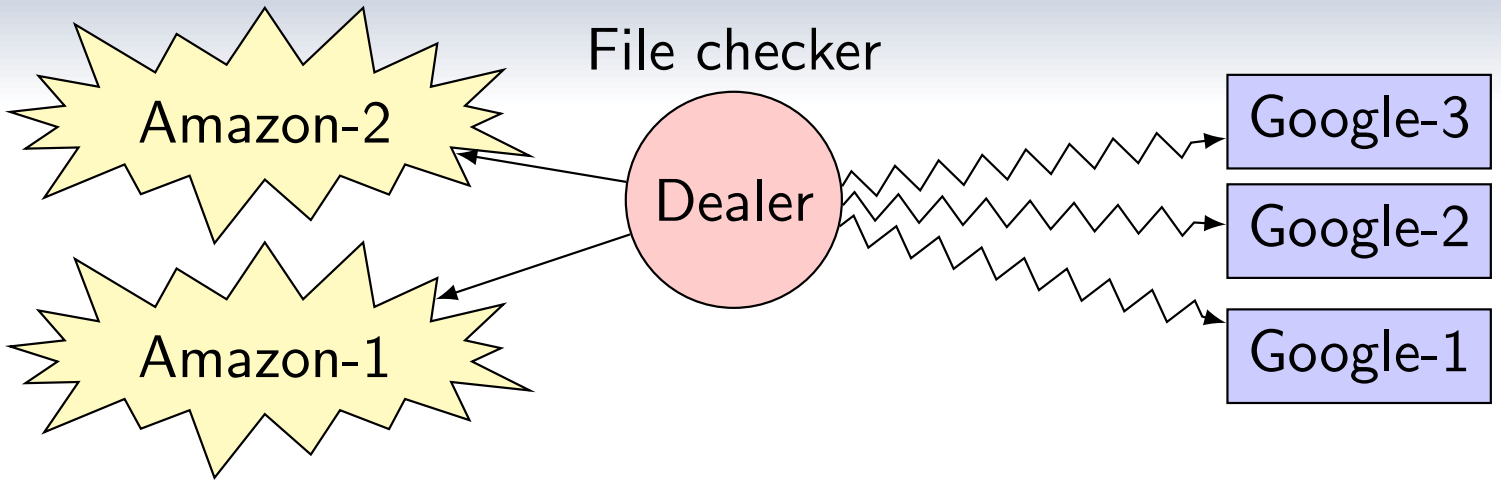

Figure: Communicationless file checker on Amazon and Google clouds

Our Setting

Properties of our setting

- A very long file can be secret shared and stored;

Our Setting

Properties of our setting

- A very long file can be secret shared and stored;
- None of which can gain any information about the files;

Our Setting

Properties of our setting

- A very long file can be secret shared and stored;
- None of which can gain any information about the files;
- String matching (searches) can be repeatedly performed by these cloud agents without them knowing anything concerning the search result.

Dag Accumulating Automata (DAA)

Definition

- Accumulating automaton that defines a graph $\mathbb{G}$ that is acyclic is a dag accumulating automaton;

Dag Accumulating Automata (DAA)

Definition

- Accumulating automaton that defines a graph $\mathbb{G}$ that is acyclic is a dag accumulating automaton;
- DAA is an accumulating automaton for which it holds for any p in V, there does not exist $\alpha_1, \ldots, \alpha_n \in \Sigma$ and $\delta_1, \ldots, \delta_n \in T$, such that

$$\delta_n(\cdots \delta_2(\delta_1(p, \alpha_1)), \alpha_2) \cdots) = p$$

Dag Accumulating Automata (DAA)

Definition

- Accumulating automaton that defines a graph $\mathbb{G}$ that is acyclic is a dag accumulating automaton;
- DAA is an accumulating automaton for which it holds for any p in V, there does not exist $\alpha_1, \ldots, \alpha_n \in \Sigma$ and $\delta_1, \ldots, \delta_n \in T$, such that

$$\delta_n(\cdots \delta_2(\delta_1(p, \alpha_1)), \alpha_2) \cdots) = p$$

- For every p and q in V, if there exists $\alpha \in \Sigma$ such that $\delta(p, \alpha) = q$ then $p \neq q$.

Marking of accumulating automata

Definition

- A marking of an accumulating automaton $\mathcal{A} = (V, \Sigma, T)$ is a vector of values, one integer value for each node in V;

Marking of accumulating automata

Definition

- A marking of an accumulating automaton $\mathcal{A} = (V, \Sigma, T)$ is a vector of values, one integer value for each node in V;
- A marked automaton $\mathcal{A}$ is a 4-tuple (V, Σ, T, M), where M is the marking vector.

Execution semantics of *AA*

After the j step, node p_i has the value $n_{p_i}^{(j)}$;
In the $(j+1)^{st}$ step, the value of p_i is computed as

- If p_i is a regular node, then

$$n_{p_i}^{(j+1)} = \sum_{\substack{\delta(p_t, \alpha_i) = p_i, \\ \forall p_t \in V;\ \alpha_i \in \Sigma}} n_{p_t}^{(j)} \cdot \alpha_i + \sum_{\substack{\delta(p_t, \alpha_i) = p_i, \\ \forall p_t \in V;\ \alpha_i = 1}} n_{p_t}^{(j)}$$

Execution semantics of *AA*

After the j step, node p_i has the value $n_{p_i}^{(j)}$;
In the $(j+1)^{st}$ step, the value of p_i is computed as

- If p_i is a regular node, then

$$n_{p_i}^{(j+1)} = \sum_{\substack{\delta(p_t,\alpha_i)=p_i,\\ \forall p_t \in V;\ \alpha_i \in \Sigma}} n_{p_t}^{(j)} \cdot \alpha_i + \sum_{\substack{\delta(p_t,\alpha_i)=p_i,\\ \forall p_t \in V;\ \alpha_i = 1}} n_{p_t}^{(j)}$$

- If p_i is an accumulating node, then

$$n_{p_i}^{(j+1)} = n_{p_i}^{(j)} + \sum_{\substack{\delta(p_t,\alpha_i)=p_i,\\ \forall p_t \in V;\ \alpha_i \in \Sigma}} n_{p_t}^{(j)} \cdot \alpha_i + \sum_{\substack{\delta(p_t,\alpha_i)=p_i,\\ \forall p_t \in V;\ \alpha_i = 1}} n_{p_t}^{(j)}$$

$\mathcal{DAA}^{\alpha\beta\gamma}$ for recognizing the regular language for $\alpha\beta\gamma$

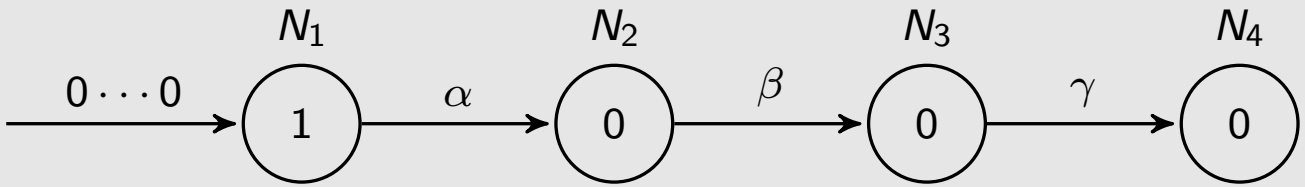

Figure: $\mathcal{DAA}^{\alpha\beta\gamma}$ and the initial marking

Executing $\mathcal{DAA}^{\alpha\beta\gamma}$

Assume the input symbol is α, the input vector is

$$\vec{v} = (v_0, v_1, v_2, v_3) = (0, 1, 0, 0)$$

Executing $\mathcal{DAA}^{\alpha\beta\gamma}$

Assume the input symbol is α, the input vector is

$$\vec{v} = (v_0, v_1, v_2, v_3) = (0, 1, 0, 0)$$

The transitions are computed as follows

$$\begin{aligned} N_1^{(1)} &= v_0 = 0 \\ N_2^{(1)} &= N_1^{(0)} \cdot v_1 = 1 \\ N_3^{(1)} &= N_2^{(0)} \cdot v_2 = 0 \\ N_4^{(1)} &= N_3^{(0)} \cdot v_3 = 0 \end{aligned}$$

Executing $\mathcal{DAA}^{\alpha\beta\gamma}$

The new marking is:

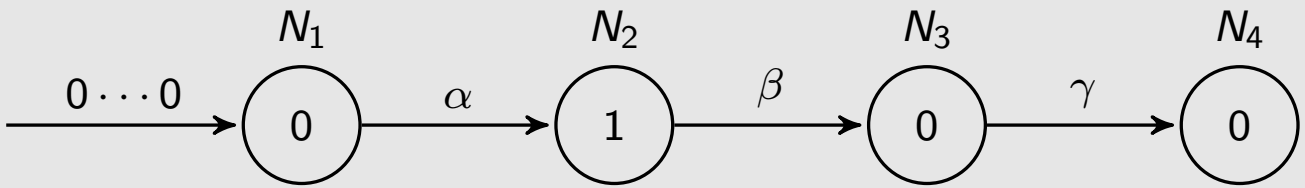

Figure: Marking of $\mathcal{DAA}^{\alpha\beta\gamma}$

Correctness of $\mathcal{DAA}^{\alpha\beta\gamma}$

Result of $\mathcal{DAA}^{\alpha\beta\gamma}$

Only if the marking is $(0, 0, 0, 1)$, the input stream is accepted, otherwise rejected. It means $N_4 = 0$.

Correctness of $\mathcal{DAA}^{\alpha\beta\gamma}$

Result of $\mathcal{DAA}^{\alpha\beta\gamma}$

Only if the marking is $(0, 0, 0, 1)$, the input stream is accepted, otherwise rejected. It means $N_4 = 0$.

Correctness of $\mathcal{DAA}^{\alpha\beta\gamma}$

- If and only if the first three input symbols are $\alpha\beta\gamma$, then the marking of $\mathcal{DAA}^{\alpha\beta\gamma}$ is $(0, 0, 0, 1)$. The state of the automaton is "accepted";

Correctness of $\mathcal{DAA}^{\alpha\beta\gamma}$

Result of $\mathcal{DAA}^{\alpha\beta\gamma}$

Only if the marking is $(0, 0, 0, 1)$, the input stream is accepted, otherwise rejected. It means $N_4 = 0$.

Correctness of $\mathcal{DAA}^{\alpha\beta\gamma}$

- If and only if the first three input symbols are $\alpha\beta\gamma$, then the marking of $\mathcal{DAA}^{\alpha\beta\gamma}$ is $(0, 0, 0, 1)$. The state of the automaton is "accepted";
- Any extra input(s) will change the marking of the automaton to $(0, 0, 0, 0)$, and the state of the automaton is also changed to "rejected".

$\mathcal{DAA}^{(\alpha\beta\alpha)^*}$ for recognizing the regular language $(\alpha\beta\alpha)^*$

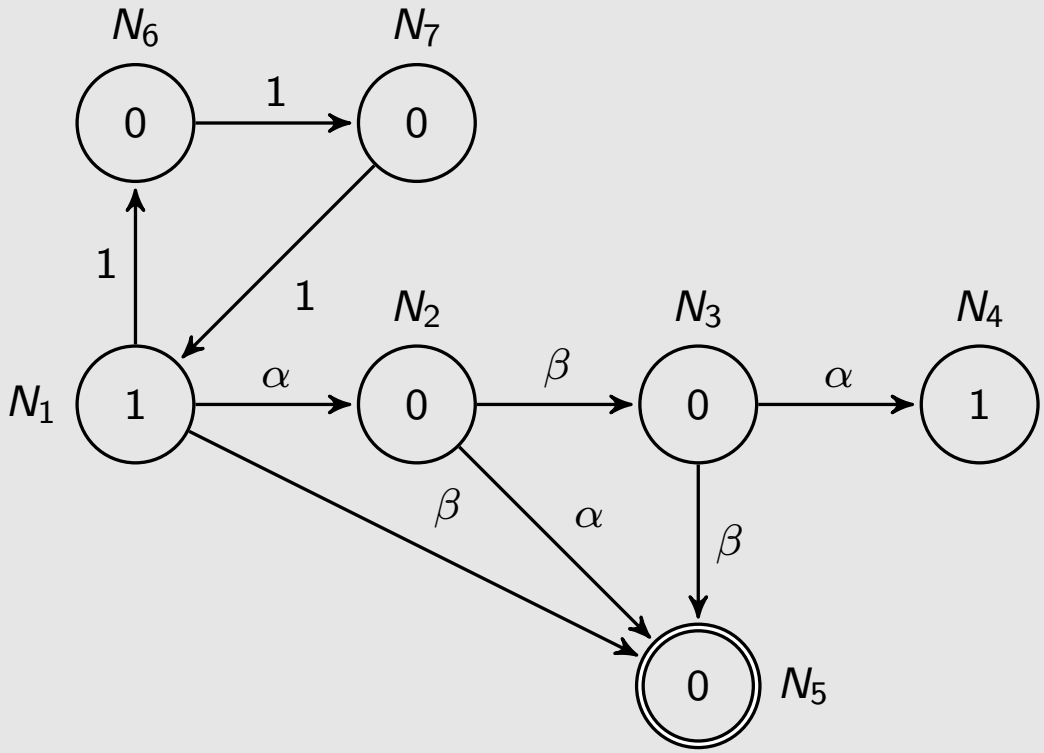

Figure: $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$ and the initial marking

Execution of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

How to compute

Compute the new value of all the regular nodes as follows

$$\begin{aligned} N_1^{(k+1)} &= N_7^{(k)} \\ N_2^{(k+1)} &= N_1^{(k)} \cdot v_1 \\ N_3^{(k+1)} &= N_2^{(k)} \cdot v_2 \\ N_4^{(k+1)} &= N_3^{(k)} \cdot v_1 \\ N_6^{(k+1)} &= N_1^{(k)} \\ N_7^{(k+1)} &= N_6^{(k)} \end{aligned}$$

Compute the new value of accumulating node N_5 as follows

$$N_5^{(k+1)} = N_5^{(k)} + N_1^{(k)} \cdot v_2 + N_2^{(k)} \cdot v_1 + N_3^{(k)} \cdot v_2$$

Correctness of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

Result of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

After any input symbol, check the marking of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$. Only if $N_1 = 1$ and $N_5 = 0$, the input stream is accepted, otherwise rejected.

Correctness of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

Result of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

After any input symbol, check the marking of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$. Only if $N_1 = 1$ and $N_5 = 0$, the input stream is accepted, otherwise rejected.

Correctness of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

According to the transitions of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

- In the initial marking of the automaton, N_4 is set to 1, N_5 is set to 0;

Correctness of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

Result of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

After any input symbol, check the marking of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$. Only if $N_1 = 1$ and $N_5 = 0$, the input stream is accepted, otherwise rejected.

Correctness of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

According to the transitions of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

- In the initial marking of the automaton, N_4 is set to 1, N_5 is set to 0;
- If the input stream is $(\alpha\beta\alpha)^*$, N_4 will be set to 1, N_5 stay 0;

Correctness of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

Result of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

After any input symbol, check the marking of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$. Only if $N_1 = 1$ and $N_5 = 0$, the input stream is accepted, otherwise rejected.

Correctness of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

According to the transitions of $\mathcal{DAA}^{(\alpha\beta\alpha)^*}$

- In the initial marking of the automaton, N_4 is set to 1, N_5 is set to 0;
- If the input stream is $(\alpha\beta\alpha)^*$, N_4 will be set to 1, N_5 stay 0;
- If the input stream is not $(\alpha\beta\alpha)^*$, N_4 will be set to 0 and/or N_5 will not be 0.

$\mathcal{DAA}^{\alpha(\alpha\beta\alpha)^*}$ for recognizing the regular language $\alpha(\alpha\beta\alpha)^*$

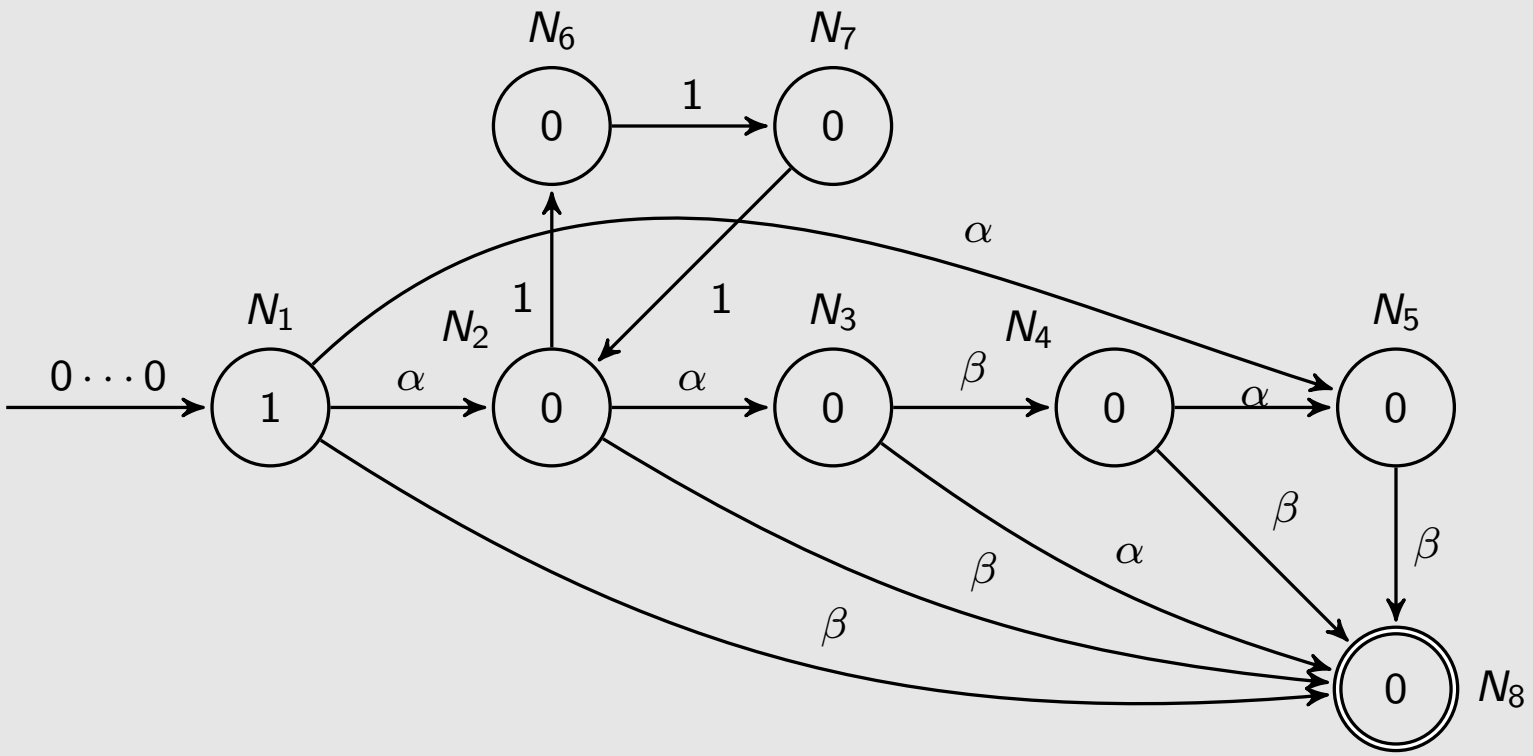

Figure: $\mathcal{DAA}^{\alpha(\alpha\beta\alpha)^*}$ and the initial marking

$\mathcal{DAA}^{\alpha^s\beta^s}$ for recognizing the context free language $\alpha^s\beta^s$

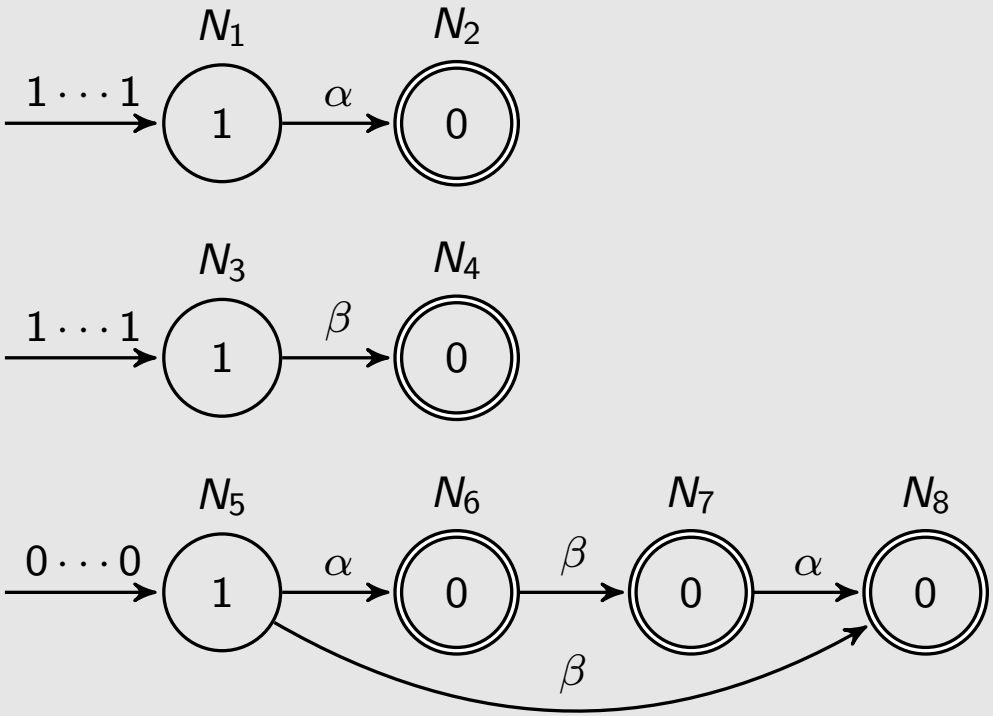

Figure: $\mathcal{DAA}^{\alpha^s\beta^s}$ and the initial marking

$\mathcal{DAA}^{\alpha^s\beta^s}$ for recognizing the context free language $\alpha^s\beta^s$

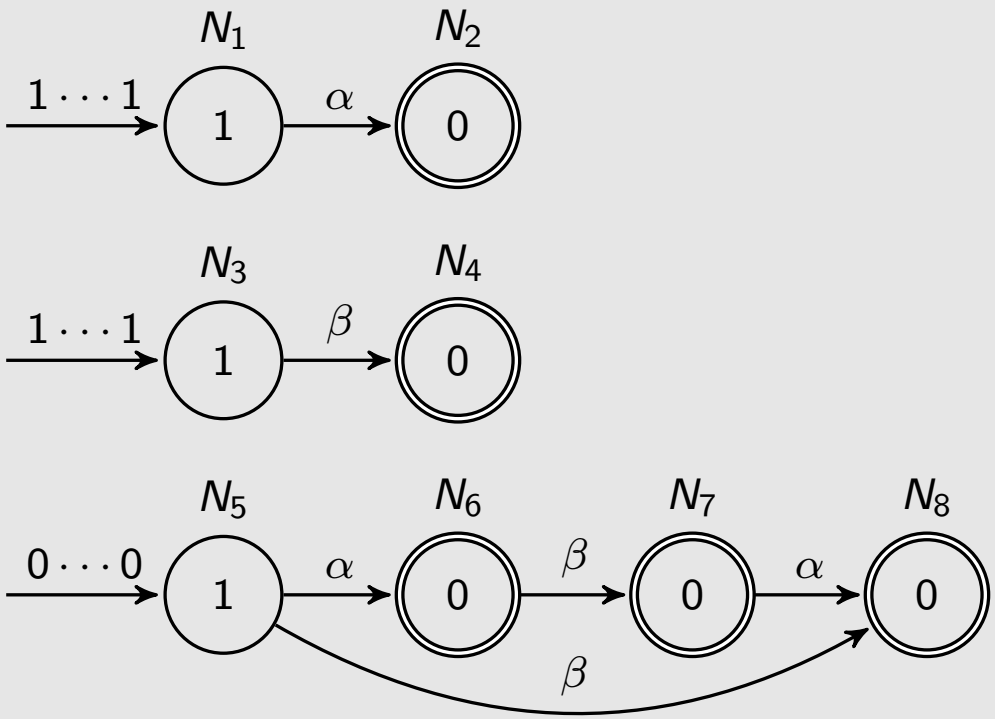

Figure: $\mathcal{DAA}^{\alpha^s\beta^s}$ and the initial marking

$\mathcal{DAA}^{\alpha^s\beta^s}$ for recognizing the context free language $\alpha^s\beta^s$

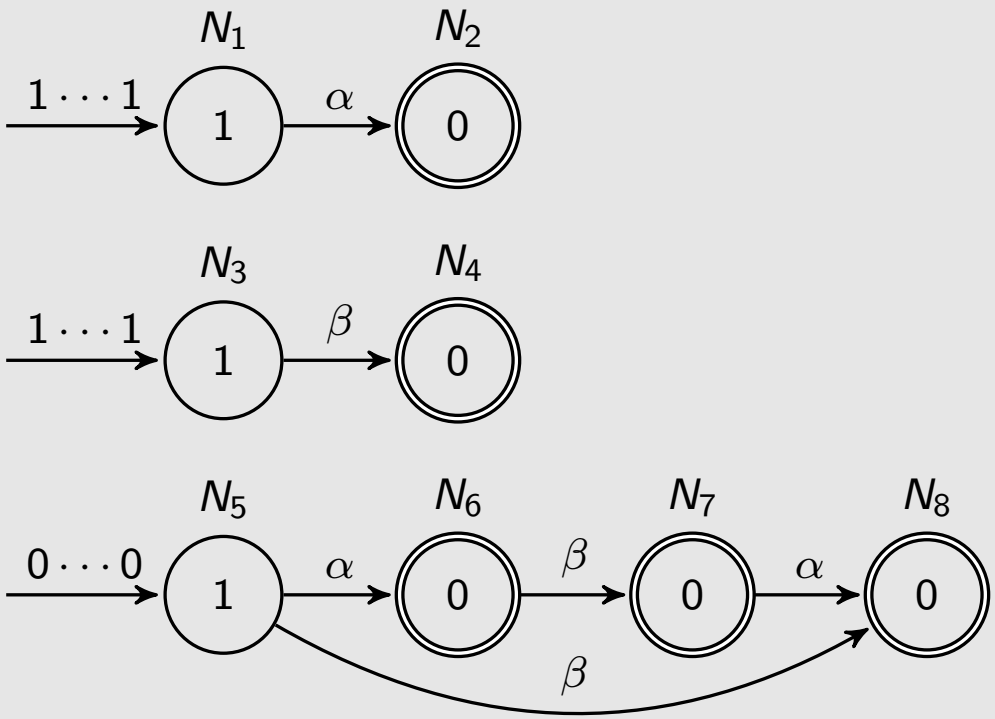

Figure: $\mathcal{DAA}^{\alpha^s\beta^s}$ and the initial marking

Hiding the structure

To separate the automaton among several clouds.

$\mathcal{DAA}^{\alpha^s\beta^s\gamma^s}$ for recognizing context sensitive language $\alpha^s\beta^s\gamma^s$

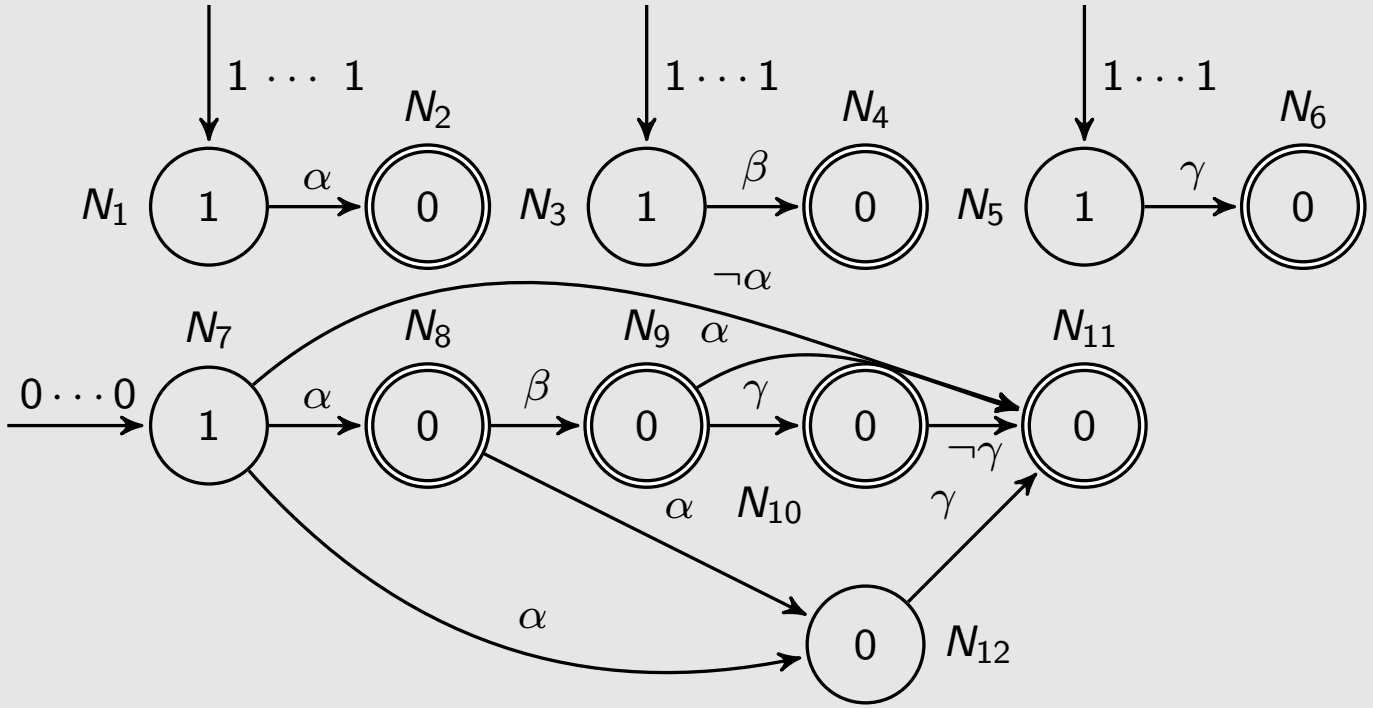

Figure: $\mathcal{DAA}^{\alpha^s\beta^s\gamma^s}$ and the initial marking

Accumulating automaton

String matching and recognizing some languages

- General string matching;

Accumulating automaton

String matching and recognizing some languages

- General string matching;

Accumulating automaton

String matching and recognizing some languages

- General string matching;
- Dag Accumulating Automata (DAA);

Accumulating automaton

String matching and recognizing some languages

- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;

Accumulating automaton

String matching and recognizing some languages

- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;
  - Recognizing the regular language $(\alpha\beta\alpha)^*$;

Accumulating automaton

String matching and recognizing some languages

- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;
  - Recognizing the regular language $(\alpha\beta\alpha)^*$;
  - Recognizing the regular language $\alpha(\alpha\beta\alpha)^*$;

Accumulating automaton

String matching and recognizing some languages

- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;
  - Recognizing the regular language $(\alpha\beta\alpha)^*$;
  - Recognizing the regular language $\alpha(\alpha\beta\alpha)^*$;
  - Recognizing the context free language $\alpha^s\beta^s$;

Accumulating automaton

String matching and recognizing some languages

- General string matching;
- Dag Accumulating Automata (DAA);
  - Recognizing the regular language $\alpha\beta\gamma$;
  - Recognizing the regular language $(\alpha\beta\alpha)^*$;
  - Recognizing the regular language $\alpha(\alpha\beta\alpha)^*$;
  - Recognizing the context free language $\alpha^s\beta^s$;
  - Recognizing the context sensitive language $\alpha^s\beta^s\gamma^s$.

Conclusions and discussions

A new automaton: Accumulating automaton, for practically unbounded inputs

- Execute any string matching privately and securely in terms of information theoretically security;

Conclusions and discussions

A new automaton: Accumulating automaton, for practically unbounded inputs

- Execute any string matching privately and securely in terms of information theoretically security;
- Other canonical samples of regular languages, context free languages and context sensitive languages.

Conclusions and discussions

Discussions

- Design a general accumulating automata in which each original symbol is mapped to several symbols;

Conclusions and discussions

Discussions

- Design a general accumulating automata in which each original symbol is mapped to several symbols;
- The dealer can send input shares not perfectly in synchronous version;

Conclusions and discussions

Discussions

- Design a general accumulating automata in which each original symbol is mapped to several symbols;
- The dealer can send input shares not perfectly in synchronous version;
- The leaked information may be eliminated by using standard (error correcting) schemes such as the Berlekamp Welch method.

Thanks