Unbounded Distributed Graph Automata

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Kushal Prakash
ID No. 2014B4A70624G

Under the supervision of:

Dr. Aiswarya Cyriac
Assistant Professor
Chennai Mathematical Institute

&

Dr. Himadri Mukherjee
Assistant Professor
BITS-Pilani Goa Campus

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS
December 2018

Declaration of Authorship

I, Kushal Prakash, declare that this Undergraduate Thesis titled, 'Unbounded Distributed Graph Automata' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Kushal · P

Date: 08-12-2018

Certificate

This is to certify that the thesis entitled, "*Unbounded Distributed Graph Automata*" and submitted by Kushal PRAKASH ID No. 2014B4A70624G in partial fulfillment of the requirements of BITS F421T Thesis embodies the work done by him under my supervision.

[illegible]

Supervisor
Dr. Aiswarya CYRIAC
Assistant Professor,
Chennai Mathematical Institute
Date: 7/12/2018

Co-Supervisor
Dr. Himadri MUKHERJEE
Assistant Professor,
BITS-Pilani Goa Campus
Date:

Abstract

The definition of Nondeterministic Distributed Graph automata as mentioned in [8] has been extended to include loops and a new generalized definition for Alternating Distributed Graph Automata has been provided. Any NDGA taking only word graphs as input is proved to be equivalent to Linear Bounded Automata. The membership and emptiness properties have been analyzed for each of the Distributed graph automata variants.

Contents

Declaration of Authorship i

Certificate ii

Abstract iii

Contents iv

1 Introduction 1

2 Nondeterministic Distributed Graph Automata 2
- 2.1 Introduction . . . 2
- 2.2 Preliminaries . . . 2
- 2.3 Nondeterministic Distributed Graph Automata . . . 3
- 2.4 Membership Problem . . . 4
- 2.5 Algorithm for membership Problem . . . 5
  - 2.5.1 Complexity . . . 5
- 2.6 Nonemptiness Problem . . . 5
  - 2.6.1 Bounded NDGA . . . 6
- 2.7 Algorithm for Nonemptiness Problem for Bounded NDGA . . . 7
  - 2.7.1 Complexity . . . 7
- 2.8 Proof of correctness . . . 8
- 2.9 Alternating Distributed Graph Automata . . . 10
  - 2.9.1 Run . . . 10
  - 2.9.2 Algorithm for membership problem . . . 11
  - 2.9.3 Complexity . . . 13

3 Nondeterministic Distributed Word Automata 14
- 3.1 Linear Bounded Automata . . . 14
- 3.2 Equivalence of LBA and NDGA on words . . . 14
  - 3.2.1 LBA to NDGA . . . 15
    - 3.2.1.1 Construction of NDGA . . . 15
  - 3.2.2 Example . . . 16
  - 3.2.3 proof of correctness . . . 17

3.2.4 NDGA to LBA . . . 18
3.3 Undecidability of Nonemptiness problem for NDGA . . . 19

4 Closure properties **21**
4.1 Closure Properties . . . 21
4.1.1 ADGA . . . 21
4.1.2 NDGA . . . 23
4.1.3 DDGA . . . 23
4.1.4 NDWA . . . 23
4.2 Summary . . . 24

A Monadic Second Order Logic for a graph language **25**
A.1 Monadic Second Order Logic for a graph language . . . 25
A.1.1 MSO Semantics . . . 25

B Alternating Distributed Graph Automata in [8] **27**
B.1 Alternating Distributed Graph Automata in [8] . . . 27
B.1.1 Mixed and Pure configuration . . . 27
B.1.2 Converting Mixed configuration to pure configuration . . . 28
B.1.3 Algorithm for membership Problem . . . 29

Bibliography **31**

Chapter 1

Introduction

Automata is an abstract machine which performs computation on an input by moving through a series of states or configurations. The set of inputs accepted by the automata form the language of the automata. Distributed Graph Automata(DGA) is an automata which takes a graph as its input and evaluates it in a distributed manner. We study the variants of DGA during this thesis. Them being

- Deterministic Distributed Graph Automata(DDGA).
- Nondeterministic Distributed Graph Automata(NDGA).
- Alternating Distributed Graph Automata(ADGA).

We use the definitions of NDGA and ADGA mentioned in [7],[6] as our inspiration for this thesis. ADGA in [8](recalled in appendix B) is defined to be bounded and is proved to be equivalent to MSO logic (semantics of MSO logic is provided in appendix A). We generalize the definitions of ADGA and NDGA by including loops, thereby making them unbounded. We have analyzed the membership and nonemptiness problems for each of the DGA variants. We define Nondeterminstic Distributed Word Automata as a special case of NDGA and prove its equivalence to Linear Bounded Automata.

As our results, we obtain that the membership problem is decidable for each of our DGA variants and provide algorithm for membership checking. We also prove that the nonemptiness problem for NDGA if undecidable by providing a construction from Linear Bounded Automata(LBA) to NDWA. We then analyzed the closure properties of the DGA variants on set operations(union,intersection,complement).

Chapter 2

Nondeterministic Distributed Graph Automata

2.1 Introduction

Nondeterministic distributed graph automata (NDGA) is an automata which takes a graph as its input and evaluates the input graph in a distributed manner. Each node of the input graph is assigned with a state. At each step, transition function determines the next state for each node of the graph. The graph is either accepted or rejected depending on the accepting set defined for the NDGA.

Figure 2.1 is an example of NDGA to accept any disconnected graph. Given an input graph, all the nodes in the graph are initially assigned with state q_{in}, the nodes then undergo transition as shown using their current state and the incoming neighbor states. Acceptance set $F = \{\{q^1_{yes}, q^2_{yes}\}\}$ for the NDGA in 2.1. The input graph to the NDGA is accepted if the graph reaches a configuration which contains the states exactly equal to any set in F.

Any input word for a finite state machine can be converted to a straight line graph by including directed edges between the positions and its neighbors. We define any NDGA taking straight line graph input as *Nondeterministic Distributed Word Automata.*

2.2 Preliminaries

An NDGA is defined on an input graph with each of the graph's node and edge labelled with the elements in set Σ and Γ respectively. Σ is a finite set of node labels and Γ is a finite set of edge labels.

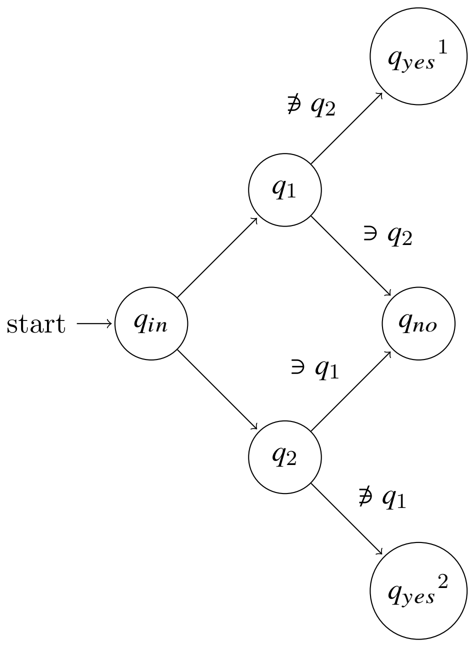

$$\begin{tikzcd}
 & & & q_{yes}{}^1 \\
 & & q_1 \arrow[ur, "\not\ni q_2"] \arrow[dr, "\ni q_2"] & \\
\text{start} \arrow[r] & q_{in} \arrow[ur] \arrow[dr] & & q_{no} \\
 & & q_2 \arrow[ur, "\ni q_1"] \arrow[dr, "\not\ni q_1"] & \\
 & & & q_{yes}{}^2
\end{tikzcd}$$

Figure 2.1: NDGA to accept all disconnected graphs.

Input graph for an NDGA The input graph for an NDGA is of the form $G = (V, \lambda, E)$ where V is a nonempty set of nodes, $\lambda : V \longrightarrow \Sigma$ maps each node to a node label and $E \subseteq V \times V \times \Gamma$.

2.3 Nondeterministic Distributed Graph Automata

Nondeterminitic Distributed Graph Automata takes an input graph and evaluates it over a sequence of steps. The NDGA accepts the input graph if the graph in the final step contains only states in the accepting set F. Let $A = (Q, \delta, \sigma, F)$ be an NDGA where

- Q is a finite set of states.
- σ is the initialization function defined as $\sigma : \Sigma \longrightarrow Q$.
- $F \subseteq 2^Q$, F is a finite set containing subsets of state set Q which describes accepting configurations.
- δ is the transition function defined as $\delta : Q \times 2^{\Gamma \times Q} \longrightarrow 2^Q$.

Run A run of a graph $G = (V, \lambda, E)$ on $A = (Q, \delta, \sigma, F)$ is a sequence $\rho = \rho_0\rho_1\rho_2.....\rho_l$ where $\rho_i : V \longrightarrow Q$ for $i \in \mathbb{N}$, $\rho_0 = \sigma \circ \lambda$. We define the length of the run ρ(denoted by $length(\rho)$) to be l. $\rho_{i+1}(v) \in \delta(\rho_i(v), S)$ where $S = \{(\gamma, q) \mid \exists u \in V, (u, v, \gamma) \in E, \rho_i(u) = q\}$

Accepting Run A run ρ of a graph $G = (V, \lambda, E)$ on $A = (Q, \delta, \sigma, F)$ is accepting if $\{\rho_l(v) \mid v \in V\} \in F$ where l is the $length(\rho)$.

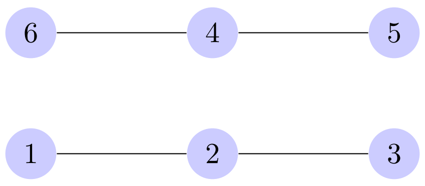


FIGURE 2.2: Input graph G

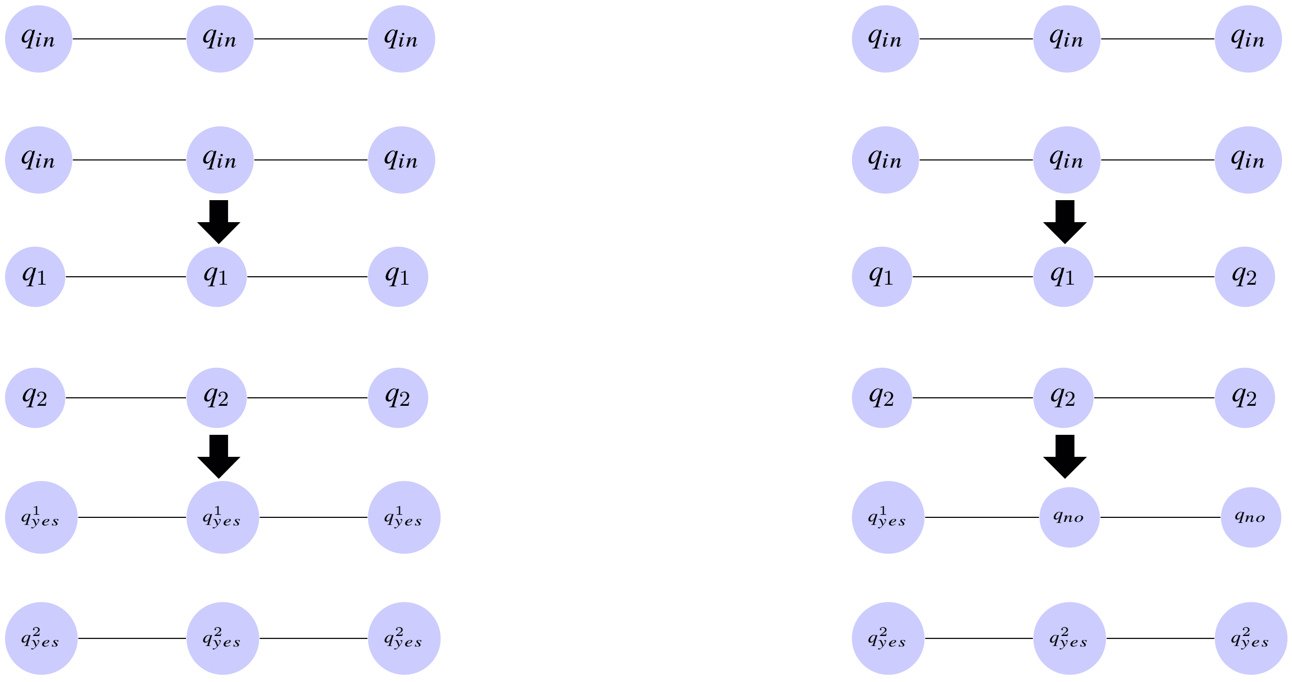


FIGURE 2.3: Accepting(left) and Rejecting(right) runs of the input graph G for the NDGA 2.1

Example Figure 2.3 shows accepting and rejecting runs for the input graph(figure 2.2) on the NDGA in figure 2.1.

Language of NDGA An input graph G is accepted by the NDGA if there exists an accepting run ρ of G on A. Language of an NDGA $A = (Q, \delta, \sigma, F)$ is the set of all graphs which are accepted by A.
$L(A) = \{G \mid \exists \text{ an accepting run } \rho \text{ of } G \text{ on } A\}$

2.4 Membership Problem

A Graph $G = (V, \lambda, E)$ is a member of an NDGA $A = (Q, \delta, \sigma, F)$ if $G \in L(A)$.
Input: $A = (Q, \delta, \sigma, F), \Sigma, \Gamma, G$ where A is an NDGA over Σ and Γ $G = (V, \lambda, E)$

$$\textbf{Output}: \begin{cases} \text{Yes , if G} \in L(A). \\ \text{No , if G} \notin L(A). \end{cases}$$

2.5 Algorithm for membership Problem

The algorithm takes a graph $G = (V, \lambda, E)$ and NDGA $A = (Q, \delta, \sigma, F)$ as inputs. We construct a pruned tree T with the root being $\sigma \circ \lambda$. For a node ρ in the tree, we look for all the incoming edges to any node $v \in V$, if u is an incoming edge, then we add $\rho(u)$ to the set S. We obtain a sets S for each node $v \in V$. A node ρ' is a child node of ρ if for every $v \in V$, $\rho'(v)$ belongs to $\delta(\rho(v), S)$ where S is the set obtained for the node v. If any node ρ' is already present in the tree T during construction, then we do not add ρ' to the tree.

The tree T constructed is accepted if any of the leaf node configuration in the tree is accepted. The algorithm is represented in 1. The function $Combination(M, V)$ returns all possible mappings from V to Q in M.

2.5.1 Complexity

Membership algorithm involves constructing a pruned tree and then searching for an accepting branch in the tree. Constructing the tree takes the same time as BFS as this involves adding all vertices to the tree by checking the transitions from each node. The tree constructed will contain maximum of $|Q|^{|V|}$ vertices and $|Q|^{2|V|}$ edges as we do not add the repeat vertices. Searching for an accepting branch in the tree again consumes the same time as BFS on the tree.

Time(Membership)= Time(construction of pruned tree) + Time(Searching for an accepting branch)

$= O(|V| + |E|)$ (V and E are the number of vertices and edges in the pruned tree).

$= O(|Q|^{|V|} + |Q|^{2|V|})$

$= O(Q^{2|V|})$

2.6 Nonemptiness Problem

Nonemptiness problem intends to check if the language of an NDGA is empty or not.

Input:$A = (Q, \delta, \sigma, F), \Sigma, \Gamma$ where A is an NDGA over Σ and Γ.

Output: $\begin{cases} \text{Yes , if L(A) is non empty.} \\ \text{No , if L(A) is empty.} \end{cases}$

Theorem: Nonemptiness problem is undecidable for an NDGA. Undecidability of Nonemptiness problem for NDGA is proved in 3.

Algorithm 1 Algorithm for Membership problem

```
Data: NDGA $A = (Q, \delta, \sigma, F)$, $\Sigma$, $\Gamma$ and Graph $G = (V, \lambda, E)$
Initialize root of Tree $T$ to $\sigma \circ \lambda$ and set $R$ to $\{\sigma \circ \lambda\}$
while $R \neq \emptyset$ do
    Initialize $\rho$ with any element in $R$
    initialize map $M$ to $\emptyset$
    foreach $v \in V$ do
        Initialize set $S$ to $\emptyset$
        foreach $(u_1, u_2, \gamma) \in E$ do
            if $u_2 == v$ then
                Add $\rho(u_1)$ to $S$
            end
        end
        Assign $\delta(\rho(v), S)$ to $M(v)$
    end
    Assign the result of $Combination(M, V)$ to $R$
    foreach $\rho' \in R$ do
        if ($\rho'$ not equal to any node in the tree $T$) then
            Add child node $\rho'$ to node $\rho$ in the tree $T$
            Add $\rho'$ to the set $R$
        end
    end
    Remove $\rho$ from set $R$
end
foreach node $t \in T$ do
    if $t$ is a leaf node then
        Initialize set $E$ to $\emptyset$
        foreach $v$ in $V$ do
            Add $t(v)$ to E
        end
        if $E \in F$ then
            return $YES$
        end
    end
end
return $NO$
```

2.6.1 Bounded NDGA

An NDGA $A = (Q, \delta, \sigma, F)$ is said to be bounded if there exists an $n \in \mathbb{N}$ which is computable from A such that for all graphs G and for all runs ρ of G on A, $length(\rho) \leq n$. The length of the NDGA A is defined as the minimum value of n such that $length(\rho) \leq n$ denoted by $len(A)$=n.

Claim: There exists an algorithm for nonemptiness problem on input A if the NDGA A is bounded([1]).

2.7 Algorithm for Nonemptiness Problem for Bounded NDGA

The algorithm takes a bounded NDGA $A = (Q, \delta, \sigma, F)$, finite set containing node labels Σ and finite set containing edge labels Γ as inputs. All possible Σ labelled Γ graphs with its number of nodes less than or equal to $|Q|^{len(A)+1}$ are checked for accepting run ρ in the NDGA A. If any graph G has an accepting run, then Yes is returned denoting that the NDGA A is Nonempty. If no such graph is found, then No is returned denoting that the NDGA A is empty.

$LabelledGraph(A, \Sigma, \Gamma, i)$ returns a set containing all possible Σ labelled Γ graphs with i number of nodes, $Membership(A, G)$ calls the membership algorithm defined earlier.

Algorithm 2 Algorithm for Nonemptiness problem

```
Data: ADGA $A = (Q, \delta, \sigma, F)$, $\Sigma$, $\Gamma$
Result: Returns $YES$ if L($A$) is non empty, else returns $NO$
assign $len(A)$ to n
assign 1 to i
while $i \leq |Q|^{n+1}$ do
    Assign the result of $LabelledGraph(A, \Sigma, \Gamma, i)$ to set variable $R$
    foreach $G$ in $R$ do
        if $Membership(G, A) = YES$ then
            Return $YES$
        else
            Return $NO$
        end
    end
    Increment i by 1
end
```

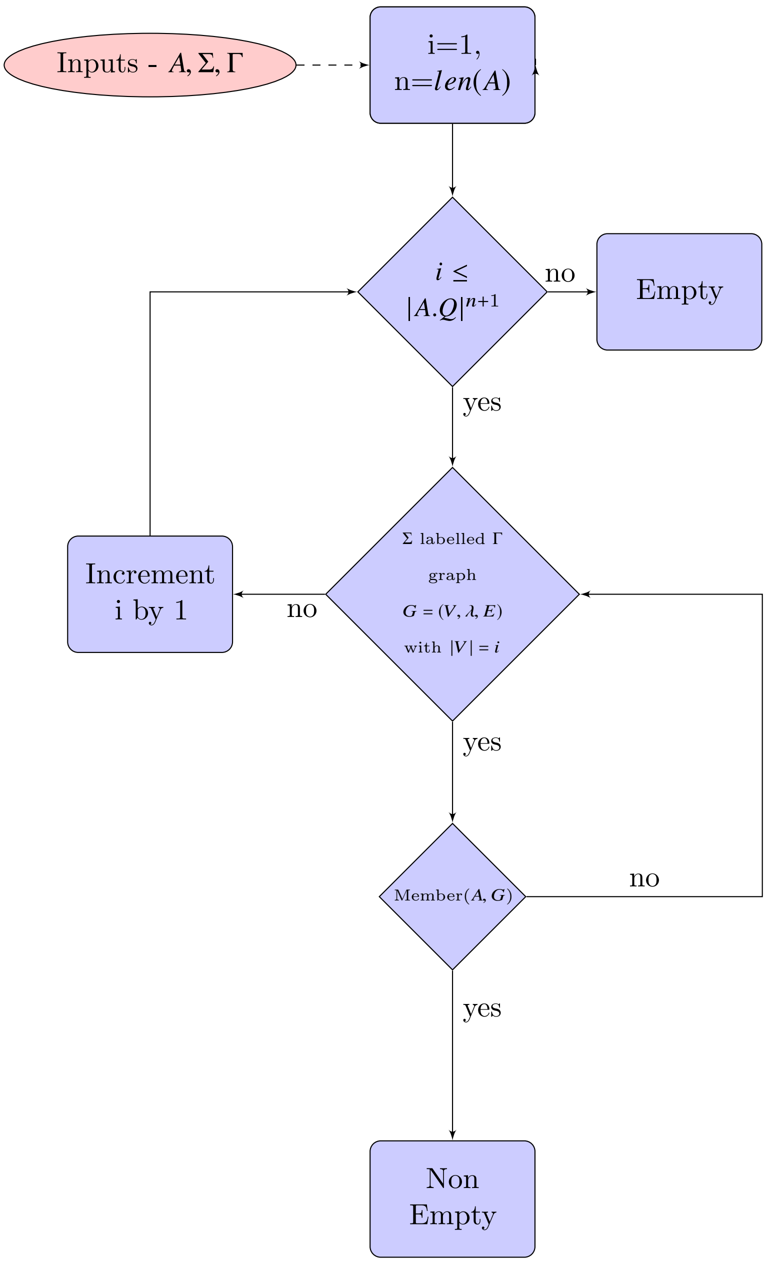
2.7.1 Complexity

$LabelledGraph(A, \Sigma, \Gamma, i)$ takes $O(|\Sigma|^i 2^{i^2 \Gamma})$ as the number of possible node labels for a graph with i nodes is $|\Sigma|^i$ and for each edge label γ in Γ, there exists an incidence matrix to indicate the presence of the edge γ between the nodes and there exists 2^{i^2} possible incidence matrices for each γ. For each graph with i nodes obtained from $LabelledGraph(A, \Sigma, \Gamma, i)$, we check the membership whose complexity has been computed earlier.

Time(NonEmptiness) $=$ Time(Nonemptiness check)

$= O(\sum_{i=1}^{i=|Q|^{n+1}} |\Sigma|^i 2^{i^2|\Gamma|} |Q|^{2i})(|\Sigma|^{|Q|^{n+1}} 2^{|Q|^{2n+2}|\Gamma|} |Q|^{2|Q|^{n+1}}$ is the upper bound for each of the term).

$$= O(|Q|^{n+1} |\Sigma|^{|Q|^{n+1}} 2^{|Q|^{2n+2}|\Gamma|} |Q|^{2|Q|^{n+1}})$$


Inputs - A, Σ, Γ
i=1, n=$len(A)$
$i \leq |A.Q|^{n+1}$
no
Empty
yes
Σ labelled Γ graph $G = (V, \lambda, E)$ with $|V| = i$
no
Increment i by 1
yes
Member(A, G)
no
yes
Non Empty

2.8 Proof of correctness

Claim:The algorithm defined checks for all graphs with number of nodes less than or equal to $|Q|^{len(A)+1}$.

Claim: If an NDGA A of length l is nonempty then there exists a graph G with its number of nodes less than or equal to $|Q|^{l+1}$ which has an accepting run ρ.
Proof: Consider a run $\rho = \rho_0\rho_1\rho_2\rho_3....\rho_l$ of the NDGA on an input graph $G = (V, \lambda, E)$. We define State sequence of a node $v \in V$ as $SS = q_0q_1q_2q_3....q_l$ where $\rho_i(v) = q_i$. The total number of distinct State sequences will be $|Q|^{l+1}$ as each state in SS belongs to Q and the maximum number of states in SS is bounded by $l + 1$.

Let the input graph G have more than $|Q|^{len(A)+1}$ nodes, by pigeon hole principle there must be two distinct nodes v, v' that follow same state sequence. We can construct a smaller graph G' by removing v' from G and adding directed edges from v to the outgoing neighbors of v'. ρ' will be the new run sequence which is same as ρ except that the domain of each ρ'_i in ρ' is $V \setminus v'$. All the nodes in the new graph G' will still follow the same sequence of states in ρ'.
If the new graph obtained still contains nodes following the same state sequence in ρ', then those nodes can be removed by the method stated above. The final graph obtained will contain only nodes with distinct state sequences and we know that the number of possible distinct state sequences is bounded by $|Q|^{len(A)+1}$.

Thus, If ρ is accepting for G, then there exists an accepting run ρ' for a graph G'(where G' has number of nodes less than or equal to $|Q|^{len(A)+1}$) as all the nodes in G' follow same state sequence in ρ' as G in ρ.

Summarizing the proof, we can classify any input NDGA into one of the following cases:

Case 1: When the NDGA is nonempty and there exists a graph $G(V, \lambda, E)$ with $|V| \leq |Q|^{len(A)+1}$ having an accepting run.

The algorithm defined checks for all graphs $G(V, \lambda, E)$ with $|V| \leq |Q|^{len(A)+1}$. Hence the algorithm will return Yes as G has an accepting run.

Case 2: When the NDGA is nonempty and there exists a graph $G = (V, \lambda, E)$ with $|V| > |Q|^{len(A)+1}$ having an accepting run.

Using our claim, we know that if a graph $G = (V, \lambda, E)$ with $|V| > |Q|^{len(A)+1}$ is accepted by the NDGA, then a new graph $G' = (V', \lambda', E)$ can be constructed(by removing nodes from G) with $|V'| \leq |Q|^{len(A)+1}$ which also has an accepting run. The algorithm defined will check for the membership of $G' = (V', \lambda', E)$ and 'Yes' will be returned.

Case 3: When the NDGA is empty.

No graph $G = (V, \lambda, E)$ will be found to have an accepting run ρ. Hence 'No' will be returned by the algorithm.

2.9 Alternating Distributed Graph Automata

An ADGA A takes an input graph $G = (V, \lambda, E)$, initializes each node of the graph with the result of initialization function. One of the sets of the result of transition function is chosen for each node. All the possible combinations of states from the sets chosen are evaluated and a tree is constructed depicting a run. The leaf node in each of the branches should be accepting for the ADGA to accept the input graph G.

Let $A = (Q, \delta, \sigma, F)$ be an ADGA on node labels Σ and edge labels Γ where

- Q is a finite set of states.
- σ is the initialization function defined as $\sigma : \Sigma \longrightarrow Q$.
- $F \subseteq 2^Q$, F is a finite set containing subsets of state set Q which can be accepted by the ADGA A.
- δ is the transition function defined as $\delta : Q \times 2^{\Gamma \times Q} \longrightarrow 2^{2^Q}$.

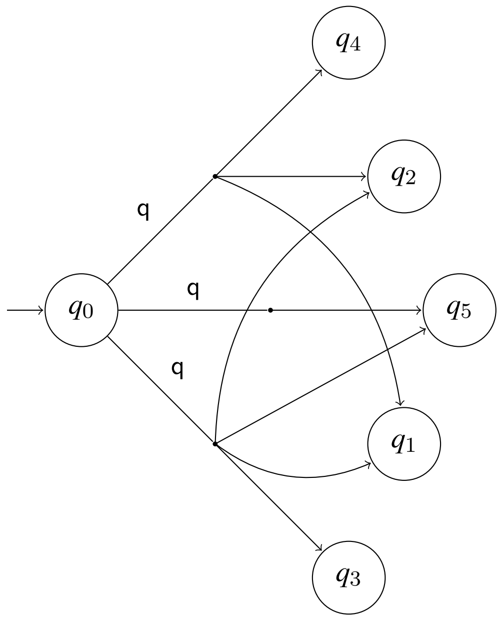


FIGURE 2.4: transition $\delta(q_0, \{q\}) = \{\{q_4, q_2, q_1\}, \{q_3, q_1, q_2, q_5\}, \{q_5\}\}$ of the ADGA

2.9.1 Run

To define a run of an ADGA $A = (Q, \delta, \sigma, F)$, we first define the terms configuration ρ, successor set function X and successor $succ(\rho)$.

Configuration ρ is a mapping from V to Q. $\rho : V \longrightarrow Q$
Successor set function X is a mapping from V to 2^Q. $X : V \longrightarrow 2^Q$

$X \models next(\rho)$ if for all $v \in V, X(v) \in \delta(\rho(v), S)$ where $S = \{(\gamma, q) \mid \exists u \in V, (u, v, \gamma) \in E, \rho(u) = q\}$
$Succ(\rho) = \{\rho' \mid \forall v \in V\ \rho'(v) \in X(v)\}$

Run of an ADGA $A = (Q, \delta, \sigma, F)$ is a tree with each node ρ_i having $succ(\rho_i)$ as its children. Root of the tree is the composition of initialization function and λ for an input graph $G = (V, \lambda, E)$, $\sigma \circ \lambda$. For all i, there exists $X : V \rightarrow 2^Q$ such that $X \models next(\rho_i)$ and for all $\rho' \in Succ(\rho_i)$, there exists j such that $\rho_j = \rho'$ where ρ_j is the child node to ρ_i and if ρ_j is equal to any of its ancestor nodes, then ρ_j is a leaf node.

A configuration ρ is accepting if it maps all the nodes of a graph to a set which is exactly equal to one of the sets in F.

Accepting Run:A run is said to be accepting if each of the leaf node configurations are accepting.

2.9.2 Algorithm for membership problem

The algorithm takes as inputs graph $G = (V, \lambda, E)$ and ADGA $A = (Q, \delta, \sigma, F)$. A pruned tree T is constructed by enumerating all possible mappings from V to 2^Q and validating the function X with the the transition function δ over each node $v \in V$ and configuration ρ. Node X is added as a child node to ρ and all possible combinations of mappings v to Q from X are added as its child nodes. Each of those mappings are also added to the set containing the configurations R. Each node before addition are checked for their presence among its ancestors. If the same node is already present, then the node is not added.

After constructing the pruned tree, all the leaf nodes of the tree are checked and if a leaf node maps the nodes V to a set in F, then the node in the tree is replaced with 1, else it is replaced with 0. Each node in the tree T at even height is replaced with $\vee$ and the nodes at odd height with $\wedge$. If any node in the tree contains only one child node, then repeat the same child node again so that the node now contains two children. The AND-OR tree constructed is evaluated and and the YES is returned if the tree evaluated to 1, else NO is returned.

The algorithm is defined in Algorithm 3 and function $enumerate(V, Q)$ defined in the algorithm returns all possible mappings from V to 2^Q and $Combination(X, V)$ returns all possible mappings from V to Q in X.
$Combination(\{1 \rightarrow \{q_1, q_2\}, 2 \rightarrow \{q_0, q_2\}\}, \{1, 2\})$ returns $\{1 \rightarrow q_1, 2 \rightarrow q_0\}, \{1 \rightarrow q_1, 2 \rightarrow q_2\}, \{1 \rightarrow q_2, 2 \rightarrow q_0\}$ and $\{1 \rightarrow q_2, 2 \rightarrow q_2\}$.

Algorithm 3 Algorithm to check membership

```
Data: Graph $G = (V, E, \lambda)$, ADGA $A = (Q, \delta, \sigma, F)$
Initialize root of Tree $T$ to $\sigma$ and set $R$ to $\{\sigma\}$
while $R \neq \emptyset$ do
    Initialize $\rho$ to any element in $R$
    Assign result of $enumerate(V, Q)$ to set $E$
    foreach $X$ in $E$ do
        Initialize $flag$ to 1
        foreach $v \in V$ do
            Initialize set $S$ to $\emptyset$
            foreach $(u_1, u_2, \gamma) \in E$ do
                if $u_2 == v$ then
                    Add $\rho(u_1)$ to $S$
                end
            end
            if $X(v) \notin \delta(\rho(v), S)$ then
                Initialize $flag$ to 0
                break
            end
        end
        if $flag \neq 0$ then
            initialize subtree $ST$ root to $X$
            Assign the result of $Combination(X, V)$ to $C$
            foreach $c \in C$ do
                Add child node $c$ to root of the Subtree $ST$
                if $\rho$ and its ancestor nodes are not equal to $c$ then
                    Add $c$ to the set $R$
                end
            end
            Add root of the Subtree $ST$ as a child node of node $\rho$ in the tree $T$
        end
    end
    Remove $\rho$ from set $R$
end
foreach node $t$ of $T$ do
    if $t$ is the leaf node then
        Initialize set $E$ to $\emptyset$
        foreach $v$ in $V$ do
            Add $t(v)$ to $E$
        end
        if $E \in F$ then
            Replace node $t$ with 1
        else
            Replace node $t$ with 0
        end
    else
        if height of node $t$ in the tree $T$ is even then
            Replace $t$ with $\vee$
        else
            Replace $t$ with $\wedge$
        end
    end
end
if AND-OR tree $T$ evaluates to 1 then
    return $YES$
else
    return $NO$
end
```

2.9.3 Complexity

Constructing the pruned tree involves adding the X node and adding all possible combinations of the mappings of X as its children. Therefore, the complexity of construction will be $O(1 + 2^{|Q||V|} + (2^{|Q||V|})^2 + ... + (2^{|Q||V|})^{2Q^{|V|}})$ as the maximum number of child nodes for a node is $2^{|Q||V|}$ and the height of the tree is bounded by $2Q^{|V|}$. Evaluating the tree involves reading each node of the tree. Therefore it takes $O(V + E)$ which here is $O(1 + 2^{|Q||V|} + (2^{|Q||V|})^2 + ... + (2^{|Q||V|})^{2Q^{|V|}} + 1 + 2^{|Q||V|} + (2^{|Q||V|})^2 + ... + (2^{|Q||V|})^{2Q^{|V|}})$.

Time(Membership)=Time(Constructing Pruned tree) + Time(evaluating the tree)
$= O(1 + 2^{|Q||V|} + (2^{|Q||V|})^2 + ... + (2^{|Q||V|})^{2Q^{|V|}}) + O(1 + 2^{|Q||V|} + (2^{|Q||V|})^2 + ... + (2^{|Q||V|})^{2Q^{|V|}})$
$= O(1+2^{|Q||V|}+(2^{|Q||V|})^2+....+(2^{|Q||V|})^{2Q^{|V|}})$(compute geometric progression sum)
$= O((2^{|Q||V|})^{2|Q|^{|V|}+1})$
$= O(2^{2|V||Q|^{|V|+1}+|Q||V|})$

Chapter 3

Nondeterministic Distributed Word Automata

3.1 Linear Bounded Automata

A linear bounded automaton(as defined in [9],[5]) is a turing machine in which the tape length is a finite fixed number. The tape head is not allowed to move off the portion of the tape containing the input, thereby restricting the computation to a finite bounded area. If the machine transition tries to move the tape head off the input, then the tape head stays where it is. The input alphabets contains two special characters which serve as the left end marker and right end marker.

3.2 Equivalence of LBA and NDGA on words

$Graph(w)$ is a graph constructed from the word w with each symbol in w represented as a node in $Graph(w)$ and receiving an edge labelled $\rightharpoonup$ from its left neighbor and edge labelled $\leftharpoonup$ from its right neighbor. Figure 3.1 shows a construction of $Graph(w)$ from a word $w = aabbaa$.

$Word(G)$ represents each node of the graph G as a symbol and each symbol takes a position relative to its neighboring node in G.

Any LBA B with an input word w can be converted to an NDGA A with the input graph $Graph(w)$. Similarly, any NDGA A with input graph G can be converted to an LBA B with input word being $Graph^{-1}(G)$ or $Word(G)$. We call any NDGA on words as Nondeterministic distributed word automata.

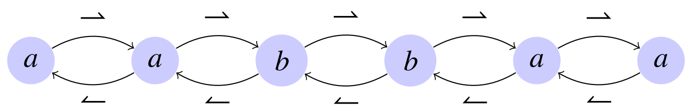


FIGURE 3.1: *Graph*(w) when $w = aabbaa$

3.2.1 LBA to NDGA

Consider an LBA $B = (Q, \Sigma, \Gamma, \delta, s, t, r)$ where

- Q is the finite set of states
- Σ is the finite set of symbols which can be present in the input word
- Γ is a finite set of symbols which can appear on the tape
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L/R\}$ is the transition function which takes the current state and tape symbol as input and returns the next state, tape symbol to replace the current symbol with and the direction in which the tape head should move.
- s is the initial state of the LBA B
- t is the accepting state
- r is the reject state

3.2.1.1 Construction of NDGA

Consider an LBA $B = (Q_B, \Sigma_B, \Gamma_B, \delta_B, s, t, r)$. An NDGA A can be constructed such that $L(A) = graph(L(B))$. The input to the NDGA will be a word graph with the nodes same as the input tape to the LBA. Each node in the input word graph receives a directed edge labelled as $\rightharpoonup$ from it's left neighbor and directed edge labelled as $\leftharpoondown$ from it's right neighbor.

NDGA $A = (Q, \delta, \sigma, F)$ is defined as the following

- $Q = \Gamma_B \cup (Q_B \times \Gamma_B)$
- $\sigma(\vdash) = (s, \vdash)$, $\sigma(\dashv) = \dashv$, $\sigma(x) = x \; \forall \; x \in \Sigma_B$
- $F = 2^{\Gamma_B \setminus \{r,t\}} \cup t$
- $\delta(x, \rightharpoonup y) = q', x$ if $y = q, z$ and $\delta_B(q, z) = q', w, R$ where $x, z, w \in \Gamma_B$ and $q, q' \in Q_b$.
  $\delta(x, \leftharpoondown y) = q', x$ if $y = q, z$ and $\delta_B(q, z) = q', w, L$ where $x, z, w \in \Gamma_B$ and $q, q' \in Q_B$.

  $\delta(x, \{\rightharpoonup y, \leftharpoondown y'\}) = x$ if $x, y, y' \in \Gamma_B$
  $\delta(x, \{\rightharpoonup y, \leftharpoondown y'\}) = z$ if $x = (q, w)$, $\delta_B(q, w) = q', z, L$ and $y, y', z, w \in \Gamma_B$

3.2.2 Example

Figure 3.3 is an NDGA equivalent for a turing machine accepting $a^n b^n$.

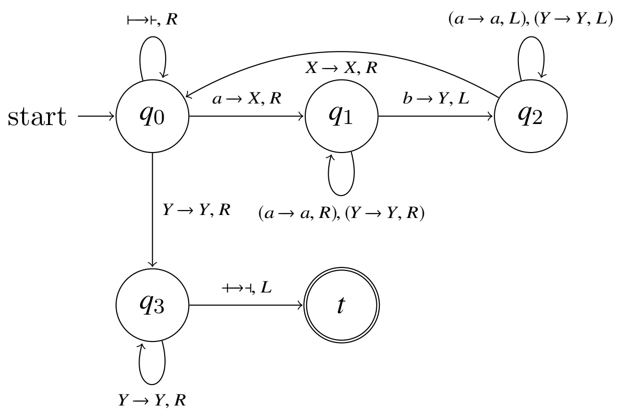


FIGURE 3.2: Turing machine to accept $a^n b^n$

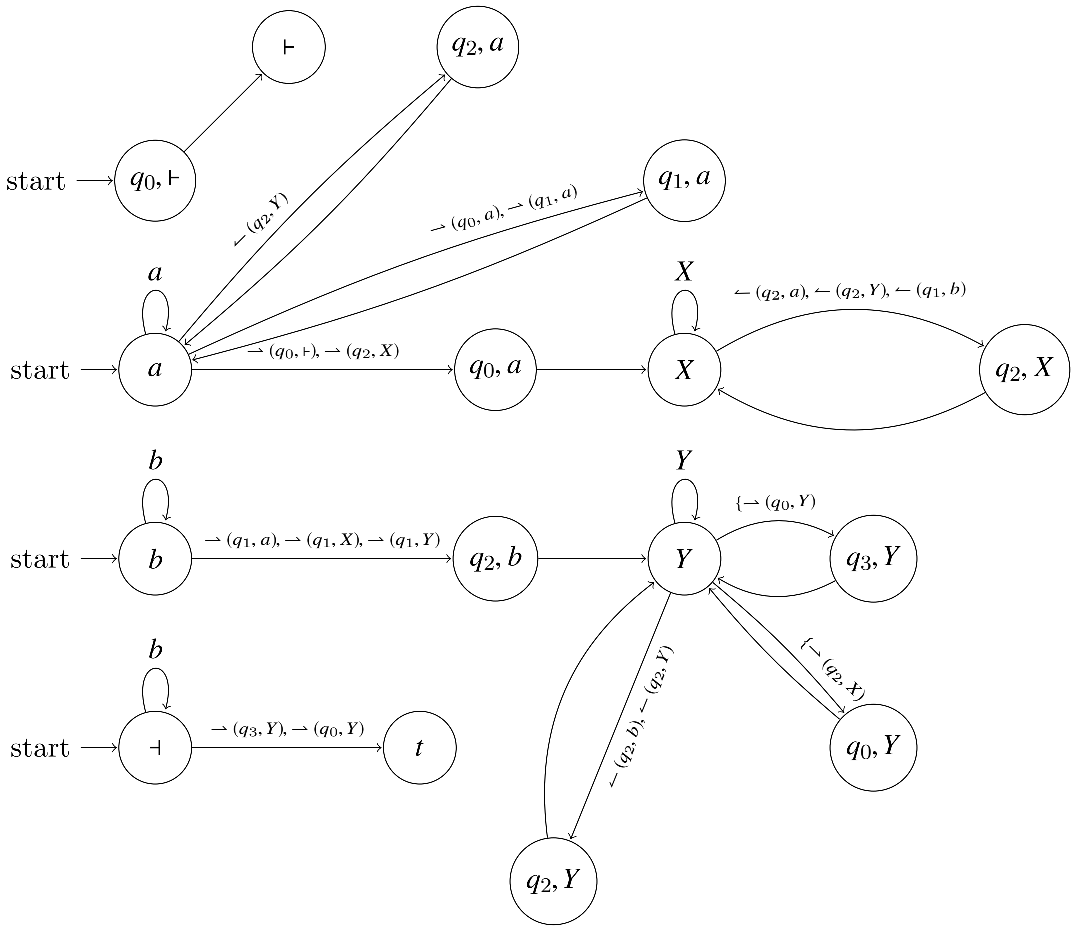

⊢
q_2, a
start
q_0, ⊢
q_1, a
↼ (q_2, Y)
⇀ (q_0, a), ⇀ (q_1, a)
a
X
↼ (q_2, a), ↼ (q_2, Y), ↼ (q_1, b)
start
a
⇀ $(q_0$, ⊢$)$, ⇀ (q_2, X)
q_0, a
X
q_2, X
b
Y
{⇀ (q_0, Y)
start
b
⇀ (q_1, a), ⇀ (q_1, X), ⇀ (q_1, Y)
q_2, b
Y
q_3, Y
{⇀ (q_2, X)
b
↼ (q_2, b), ↼ (q_2, Y)
start
⊣
⇀ (q_3, Y), ⇀ (q_0, Y)
t
q_0, Y
q_2, Y

FIGURE 3.3: NDGA to accept $a^n b^n$

3.2.3 proof of correctness

The input tape of an LBA with directed labelled edges between the nodes is the input for the NDGA constructed.

Claim:Computation tree of the LBA and the computation tree of the NDGA constructed are isomorphic.

Consider an LBA $B = (Q_B, \Sigma_B, \Gamma_B, \delta_B, s, t, r)$, NDGA $A = (Q, \delta, \sigma, F)$ and an input word X. LBA B begins at $(s, \vdash X \dashv, 0)$ where s is the initial state of the LBA, 0 is the position of the state. Let G be the graph constructed from the word $\vdash X \dashv$ in which each tape alphabet has incoming directed edge labelled $\rightharpoonup$ from its left neighbor and incoming directed edge labelled $\leftharpoondown$ from its right neighbor. ρ_0 maps $\vdash$ of the input graph G to $s, \vdash$ and every other node to itself.

The LBA then checks for $\delta_B(s, \vdash)$ where $\vdash$ is the left end marker and moves to the right and replaces the alphabet $\vdash$ with $\vdash$.

In the NDGA, the second node receives state $s, \vdash$ from the first node through an edge labelled $\rightharpoonup$. The node now goes to state named as the tuple obtained from $\delta(x, (s, \vdash))$ that is if $\delta_B(s, \vdash) = (q, \vdash, R)$ then $\delta(x, (s, \vdash)) = (q, x)$ and the first node is replaced with $\vdash$.

Assuming n transitions have taken place in the LBA and NDGA, we prove that both the DGA and LBA undergo the same $(n+1)^{th}$ transition. Let LBA be in (q', Y, i) where $q' \in Q_b$, Y is the tape after n transitions and $i \in \mathbb{N}$ is the position of the tape head. The LBA reads the current alphabet of the tape, checks for $\delta_b(q', y)$ where $y \in \Gamma$ is the alphabet in the i^{th} position of Y, replaces the alphabet y with w and tape head moves one step to the right if $\delta_B(q', y) = (q", w, R)$. Consider run ρ, on applying the function ρ_n on the input graph, each node of the input graph will be in the same state as that of the tape alphabets after n transitions except the $(i)^{th}$ node which will be in q', y where y is the $(i)^t h$ alphabet in Y. ρ_{n+1} maps the i^{th} node of the graph to w, $(i+1)^{th}$ node to $q", x'$ if $\delta_B(q', y) = (q", w, R)$ and x' if the $(i+1)^{th}$ node of the graph G. Therefore, LBA and NDGA both undergo the same $(n+1)^{th}$ transition.

A tree can be constructed for the LBA's computation with the nodes being the tape after each transition for the LBA. Similarly, a tree can be constructed for the NDGA with i^{th} node being ρ_i for all $i \in \mathbb{N}$. Each node in both the trees will have exactly one child node except the leaf node. The height of both the trees are the same as both the LBA on an input word and NDGA on an input graph undergo equivalent transitions. The leaf node will either be an accepting configuration or a rejecting configuration. Thus both the trees are isomorphic.

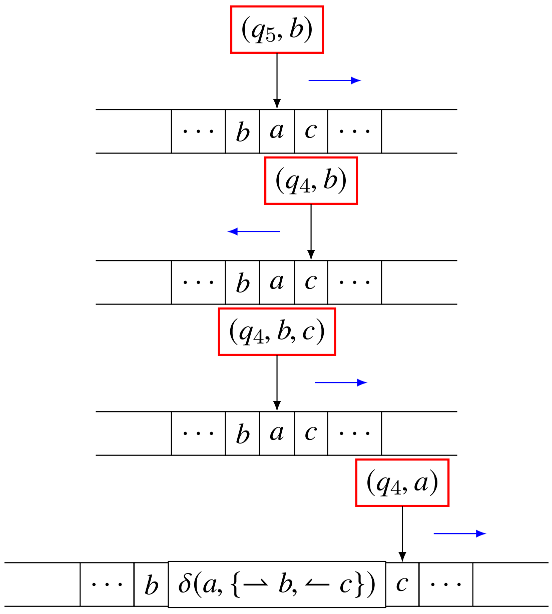
Claim: Language accepted by the NDGA is same as the language accepted by the LBA.
Proof:Using our previous claim, we know that the computation tree for the LBA and the NDGA are isomorphic. Also, we know that for any input, if the leaf node of the tree constructed for

the LBA is accepted, then the leaf node of the tree constructed for the NDGA is also accepted. Similarly, if the leaf node of the tree constructed for the LBA is rejected, then the leaf node of the tree constructed for the NDGA is also rejected. Therefore, the language accepted by the NDGA is same as the language accepted by the LBA.

3.2.4 NDGA to LBA

Consider an NDGA $A = (Q_A, \delta_A, \sigma_A, F)$ over node labels Σ and edge labels Γ. LBA $B = (Q_B, \Sigma_B, \Gamma_B, \delta_B, s, t, r)$ can be constructed from the NDGA A where

- $\Sigma_B = \Sigma_A$
- $\Gamma_B = \Sigma_A \cup Q_A$
- $Q_B = (\{q_0, q_1, q_3, q_4, q_5, t\} \times (\Gamma_B \cup 2^{\Gamma_B})) \cup \{q_0, q_1, q_3, q_4, q_5, t\}$
- $s = q_0$, s is the start state
- if x is the first node of the input graph
-
  - $\delta_B(q_0, \vdash) = (q_0, \vdash, R)$ where $\vdash$ is the left end marker
  - $\delta_B(q_0, x) = (q_0, \sigma(x), R)$ for all $x \in X$
  - $\delta_B(q_0, \dashv) = (q_1, \dashv, L)$ where $\dashv$ is the right end marker
  - $\delta_B(q_1, x) = (q_1, x, L)$ for all $x \in \Gamma_B$
  - $\delta_B(q_1, \vdash) = ((q_2, \emptyset), \vdash, R)$
  - $\delta_B((q_2, S), x) = ((q_2, S \cup \{x\}), x, R)$ if $S \cup \{x\}$ is a subset of any set in F and and $S \subseteq 2^{\Gamma_B}$
  - $\delta_B((q_2, S), x) = (q_3, x, L)$ if $S \cup \{x\}$ is a not a subset of any set in F
  - $\delta_B((q_2, S), \dashv) = (t, \dashv, L)$ if $S \in F$,
  - $\delta_B((q_2, S), \dashv) = (q_3, \dashv, L)$ if $S \notin F$
  - $\delta_B(q_3, x) = (q_3, x, L)$ for all $x \in \Gamma_B$
  - $\delta_B(q_3, \vdash) = ((q_4, \vdash), \vdash, R)$
  - $\delta_B((q_4, y), x) = ((q_5, y), x, R)$ for all $x \in \Gamma_B$
  - $\delta_B((q_5, y), z) = ((q_5, y, z), y, L)$ for all $y \in \Gamma_B$
  - $\delta_B((q_5, y, z), x) = ((q_4, x), \delta(x, \{\rightharpoonup y, \leftharpoondown z\}), R)$ for all $x, y, z \in \Gamma_B$
  - $\delta_B((q_5, y, z), x) = ((q_4, x), \delta(x, \{\leftharpoondown z\}), R)$ for all $x, z \in \Gamma_B$ and $y =\vdash$
  - $\delta_B((q_5, y, z), x) = ((q_4, x), \delta(x, \{\rightharpoonup y\}), R)$ for all $x, y \in \Gamma_B$ and $z =\dashv$
  - $\delta_B((q_4, x), \dashv) = (q_1, \dashv, L)$ for all $x \in \Gamma_B$



(q_5, b)
$\cdots$ b a c $\cdots$
(q_4, b)
$\cdots$ b a c $\cdots$
(q_4, b, c)
$\cdots$ b a c $\cdots$
(q_4, a)
$\cdots$ b $\delta(a, \{\rightharpoonup b, \leftharpoondown c\})$ c $\cdots$

FIGURE 3.4

The LBA starts at state q_0 and for each tape alphabet $x \in \Sigma_B$ scanned, tape head moves to the right and stays in the state q_0. As the tape head scans the right end marker $\dashv$, the LBA goes to state q_1 and the tape head moves to the left. The tape head keeps moving left till it points to the left end marker $\vdash$. On reading the left end marker, LBA goes to state q_2 and moves to the right. The tape header then scans the first alphabet of the tape x, if $\{x\}$ is a subset of any set in F, then the LBA goes to state (q_2, X) where $X = \{x\}$ and the tape head moves to the right. If $\{x\}$ is not a subset of any set in F, then LBA goes to state q_3 and tape head moves to the left. The LBA when in state (q_2, X) scans the tape alphabet y and goes to state $(q_2, X \cup y)$ if $X \cup y$ is a subset of any set in F and the tape head moves to the right, else it goes to q_3 and the tape head moves to the left. If tape head scans the right end marker when in state (q_2, X) and $X \subseteq F$, then LBA goes to accepting state t and tape head moves to the left, else LBA goes to state q_3 and tape head goes to the left. When in state q_3, the tape head keeps moving left without making any changes to the tape until it reaches the left end marker. On reaching the left end marker, LBA goes to state q_4 and tape head moves to the right. For each tape alphabet, the tape head scans both its neighbors and replaces the alphabet with the state as determined by the transition function of the NDGA. When the right end marker is reached, the LBA goes back to state q_1 and the tape head moves to the left.

3.3 Undecidability of Nonemptiness problem for NDGA

Nonemptiness problem for an NDGA is undecidable. The proof is by reduction from nonemptiness of an LBA. We show that if nonemptiness problem for an NDGA is decidable,then nonemptiness problem of LBA will also be decidable. For this reduction, we use the construction of NDGA from LBA as defined before.

Suppose a turing machine R decides Nonemptiness of an NDGA. We construct a turing machine S to decide Nonemptiness of an LBA as follows.

S = "On input $\langle B\rangle$ where B is an LBA:
1. Construct an equivalent NDGA A from B using the construction defined earlier.
2. Run R on input $\langle A\rangle$.
3. If R accepts, accept; if R rejects, reject."

If R accepts $\langle A\rangle$, then $L(A) = \emptyset$. Thus $L(B) = \emptyset$ as A is an equivalent conversion of B to NDGA.

Chapter 4

Closure properties

4.1 Closure Properties

4.1.1 ADGA

ADGA are closed under union,intersection and complement

Consider two ADGAs A_1 and A_2. We can construct an ADGA $A_\cup$ to recognize $L(A_1) \cup L(A_2)$. Initialize every node to a state named by the node name itself. Nondeterministically allocate the nodes to the initialization function mappings of A_1 or A_2 in step 2 and evaluate the chosen ADGA. Accepting set will be the union of the accepting sets of the two ADGAs $F_\cup = F_1 \cup F_2$. Suppose a node of the input graph chooses A_1 and receives a state in A_2 from its neighbor, then it goes to state q_{no} which is not contained in $F_\cup$.

Suppose there exists an ADGA $A = (Q, \delta, \sigma, F)$. We construct its complement ADGA $\overline{A} = \overline{Q}, \overline{\delta}, \overline{\sigma}, \overline{F})$ as follows:

- $\overline{Q} = Q \cup \{(q_S) \mid q \in Q, S \in 2^Q\}$
- $\overline{\sigma} = \sigma$
- $\overline{F} = 2^Q - F$
- For all $q \in Q$ if $\delta(q, S) = \{S_1, S_2, S_3, ..., S_n\}$ where S is the set of input states and $S_1, S_2, S_3 .... S_n$ are sets of states. $\overline{\delta}(q, S) = \{\{q_{S_1}, q_{S_2}, q_{S_3}, ..., q_{S_n}\}\}$ $\overline{\delta}(q_{S_i}, X) = \{\{{q_i}^1\}, \{{q_i}^2\}, \{{q_i}^3\}, ..., \{{q_i}^m\}\}$ where X denotes for any incoming set of states.

For example, consider an ADGA A with $\sigma(a) = q_1, \sigma(b) = q_2$, $\delta(q_1, \{q_2\}) = \{\{q_3, q_4\}, \{q_5, q_3\}\}$, $\delta(q_2, \{q_1\}) = \{\{q_6, q_4\}\}$ and $F = \{\{q_3, q_6\}, \{q_5, q_6\}\}$. Its complement $\overline{A}$ will have $\sigma(a) = q_1, \sigma(b) =$

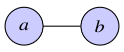


FIGURE 4.1: Input Graph

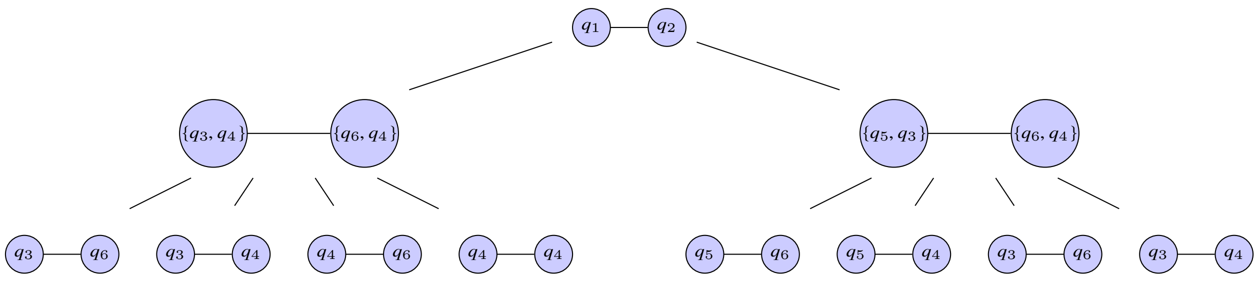


FIGURE 4.2: Pruned tree for ADGA with $\sigma(a) = q_1, \sigma(b) = q_2$, $\delta(q_1, \{q_2\}) = \{\{q_3, q_4\}, \{q_5, q_3\}\}$, $\delta(q_2, \{q_1\}) = \{\{q_6, q_4\}\}$, $F = \{\{q_3, q_6\}, \{q_5, q_6\}\}$

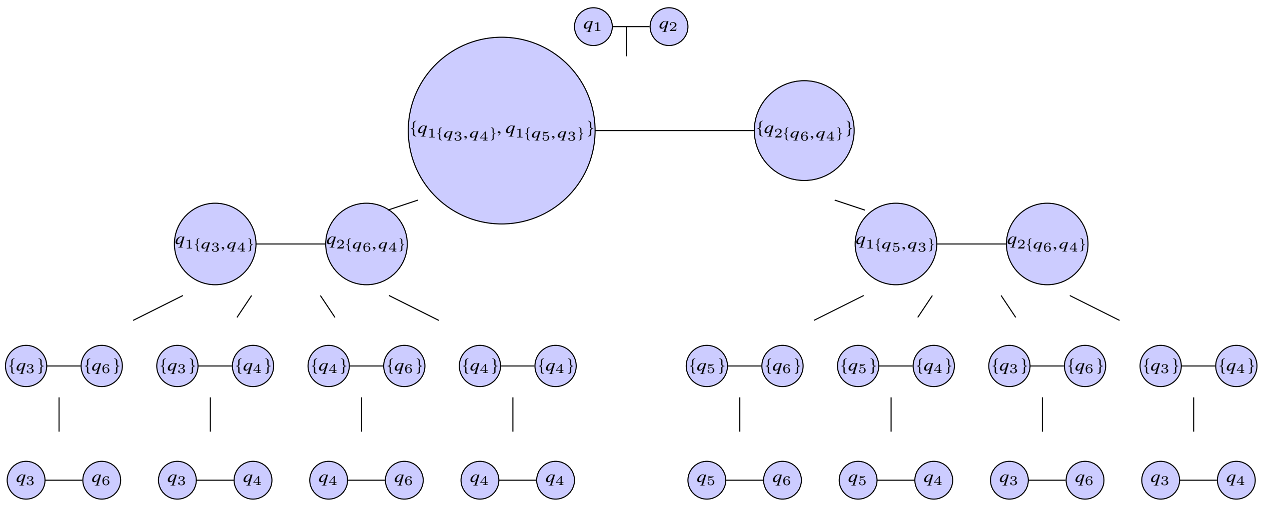


FIGURE 4.3: Pruned tree for the complement of the ADGA in 4.2

q_2, $\overline{\delta}(q_1, \{q_2\}) = \{\{q_{1\{q_3,q_4\}}, q_{1\{q_5,q_3\}}\}\}$, $\overline{\delta}(q_2, \{q_1\}) = \{q_{2\{q_6,q_4\}}\}$, $\overline{\delta}(q_{1\{q_3,q_4\}}, X) = \{\{q_3\}, \{q_4\}\}$, $\overline{\delta}(q_{1\{q_5,q_3\}}, X) = \{\{q_5\}, \{q_3\}\}$, $\overline{\delta}(q_{2\{q_6,q_4\}}, X) = \{\{q_6\}, \{q_4\}\}$ and $F = 2^{\{q_1,q_2,q_3,q_4,q_5,q_6\}} - \{\{q_3, q_6\}, \{q_5, q_6\}\}$. The pruned tree constructed for A on the input graph 4.1 is shown in 4.2 and the pruned tree for the complement ADGA $\overline{A}$ is shown in 4.3.

Proof:

If possible, let there exist input graph G which is accepted by both A and A'. Suppose G has k nodes and each node i is assigned with q_i by initialization function and $\delta(q_i, S) = \{S_1{}^i, S_2{}^i, S_3{}^i, ..., S_n{}^i\}$. If G is accepted, then we know there exists state sets $S_{j1}{}^1, S_{j2}{}^2, S_{j3}{}^3, ..., S_{jk}{}^k$ chosen by the nodes which will be accepted. Therefore, each branch of the set choice will be accepted. In the automata A', all the branches for the choice of states $q_{S_{j1}{}^1}, q_{S_{j2}{}^2}, q_{S_{j3}{}^3}, ..., q_{S_{jk}{}^k}$ will be rejected as all the transitions in $\delta(q_{S_{ji}{}^i})$ will lead to rejecting states. Therefore, A' will reject G and we obtain a contradiction.

If possible, let there exist input graph G which is rejected by both A and A'. For A to reject G, there must exist a rejecting branch in every choice of state sets where $\delta(q_i, S) = \{{S_1}^i, {S_2}^i, {S_3}^i, ..., {S_n}^i\}$ and each of ${S_j}^i$ are state sets. Suppose there exists one rejecting branch in each choice, then that branch is accepted in A' and since each choice will now contain atleast one accepting branch, A' will accept G.

4.1.2 NDGA

NDGA are closed under union and intersection. The construction of the NDGA for union is same as that for ADGA.

For intersection, we construct an automata in which each state is the Cartesian product of a state in A_1 and a state in A_2. Transition function $\delta((q_1, q_2), (S_1 \times S_2)) = (\delta_{A_1}(q_1, S), \delta_{A_2}(q_2, S)$ where S_1 is the incoming set of states in A_1 and S_2 is the incoming set of states in A_2. The input graph is accepted if the projection of the first symbols(first symbol in the Cartesian product) of each node in a configuration is exactly equal to any set in F_1 and the projection of the second symbols of each node are exactly equal to any set in F_2.

BNDGA are proved to be not closed under complement in [8]. However, checking closure of NDGA under complement requires further research.

4.1.3 DDGA

DDGA are closed under union, intersection and complement. The construction for intersection is same as that of NDGA. The construction for union is same as that of construction for intersection except the accepting set. The input graph is accepted if the projection of the first symbols(first symbol in the Cartesian product) of each node in a configuration is exactly equal to any set in F_1 or the projection of the second symbols of each node are exactly equal to any set in F_2(F_1 and F_2 are the accepting sets of the two automata). For complementing, the accepting set becomes $2^Q \setminus F$.

DDWA:Deterministic Distributed Word Automata is a DDGA whose input $Graph(w)$ where w is a word. DDWA is closed under union, intersection and complement and the constructions are same as that for DDGA.

4.1.4 NDWA

Nondeterministic Distributed Word Automata are NDGA whose input is $Graph(w)$ where w is a word. NDWA are closed under union, intersection and complement. We know that NDWA

	ADGA	NDGA	ADWA	NDWA	DDWA	BADGA	BNDGA	BDDGA
Nonemptiness	undecidable	undecidable	undecidable	undecidable	undecidable	undecidable	decidable	decidable
Membership	decidable	decidable	decidable	decidable	decidable	decidable	decidable	decidable

TABLE 4.1: Decidability of Nonemptiness and Membership problems

are equivalent to LBA. A language recognized by an LBA is always context sensitive and context sensitive language is closed under set operations. Therefore, NDWA is closed under set operations.

The same construction as that of ADGA can be provided for union and intersection. The construction of complement is believed to follow Immerman and Szelepcsényi theorem(from [4],[10]).

	ADGA	NDGA	ADWA	NDWA	DDWA	BADGA	BNDGA	BDDGA
Union	✓	✓	✓	✓	✓	✓	✓	✓
Intersection	✓	✓	✓	✓	✓	✓	✓	✓
Complement	✓	?	✓	✓	✓	✓	✗	✓

TABLE 4.2: closure properties

4.2 Summary

The definition of NDGA as mentioned in [8] as been extended to include loops and also generalized the definition of ADGA. We provided membership algorithm for the new definition of NDGA and ADGA. We then proved the equivalence of NDGA on words(termed as NDWA) and LBA by providing constructions from LBA to NDWA and vice versa. Therefore, proving that NDWA can recognize only context sensitive language.

We analyzed the closure properties over set operations and the results are shown in table 4.2. The decidability of nonemptiness problem and membership problem are summarized in table 4.1.

There exists further scope of research in various directions. Some of them being:

- What is the Expressive power of ADGA?
- What is the Expressive power of NDGA?
- Can a construction of complement for NDGA be provided?
- Study the similarities between ADGA and cellular automata(defined in [2]).
- Can NDWA be converted to DDWA? (long standing open problem for LBA)

Appendix A

Monadic Second Order Logic for a graph language

A.1 Monadic Second Order Logic for a graph language

Let $G_\lambda \in \Sigma^\Gamma$ and $\lambda : V_G \rightarrow \Sigma$ where Σ is the set of node labels and Γ is the set of edge labels. Σ^Γ is the set of all Σ labelled Γ graphs and V_G is a finite nonempty set of nodes.
The set MSO$\langle\Sigma, \Gamma\rangle$ on any G_λ is built with the following atomic formulas.

- $\diamond_a x$ (x has label a).
- $x \overset{\gamma}{\rightharpoonup} y$ (x has a γ edge to y where $\gamma \in \Gamma$).
- $x = y$ (x is equal to y).
- $x \in X$ (x is an element of X).

where $x, y \in V_{node}$, $X \in V_{set}$, $a \in \Sigma$, $\gamma \in \Gamma$. V_{node} is the set of node variables and V_{set} is the set of set variables. If ϕ and ψ are MSO(Σ, Γ) formulas then so are $\neg\phi$,$\phi \wedge \psi$,$\phi \vee \psi$,$\exists x(\phi)$,$\forall x(\phi)$,$\exists X(\phi)$ and $\forall X(\phi)$.

A.1.1 MSO Semantics

Formulas(ϕ) are interpreted on pairs (G_λ, α), where $G_\lambda \in \Sigma^\Gamma$ and
$\alpha : free(\phi) \rightarrow V_G \cup 2^{V_G}$. The truth conditions of the atomic formulas are

- $(G_\lambda, \alpha) \vDash \diamond_a x$ iff $\lambda(\alpha(x)) = a$

- $(G_\lambda, \alpha) \vDash x \overset{\gamma}{\rightharpoonup} y$ iff $\alpha(x) \overset{\gamma}{\rightharpoonup} \alpha(y)$ and $\gamma \in \Gamma$
- $(G_\lambda, \alpha) \vDash x = y$ iff $\alpha(x) = \alpha(y)$
- $(G_\lambda, \alpha) \vDash x \in X$ iff $\alpha(x) \in \alpha(X)$

where x,y$\in V_{node}$ and X$\in V_{set}$. If ϕ and ψ are MSO(Σ, Γ) formulas, then

- $(G_\lambda, \alpha) \vDash \neg\phi$ iff $(G_\lambda, \alpha) \nvDash \phi$
- $(G_\lambda, \alpha) \vDash (\phi \vee \psi)$ iff $(G_\lambda, \alpha) \vDash \phi$ or $(G_\lambda, \alpha) \vDash \psi$
- $(G_\lambda, \alpha) \vDash (\phi \wedge \psi)$ iff $(G_\lambda, \alpha) \vDash \phi$ and $(G_\lambda, \alpha) \vDash \psi$
- $(G_\lambda, \alpha) \vDash \forall x(\phi)$ iff for all $k \in V_G$, $(G_\lambda, \alpha') \vDash \phi$ where $\alpha' : free(\phi) \to V_G \cup 2^{V_G}$, $\alpha'(y) = \alpha(y)$ for all $y \neq x$ and $\alpha'(y) = k$ when $y = x$.
- $(G_\lambda, \alpha) \vDash \exists x(\phi)$ iff for some $k \in V_G$, $(G_\lambda, \alpha') \vDash \phi$ where $\alpha' : free(\phi) \to V_G \cup 2^{V_G}$, $\alpha'(y) = \alpha(y)$ for all $y \neq x$ and $\alpha'(y) = k$ when $y = x$.
- $(G_\lambda, \alpha) \vDash \forall X(\phi)$ iff for all $K \in 2^{V_G}$, $(G_\lambda, \alpha') \vDash \phi$ where $\alpha' : free(\phi) \to V_G \cup 2^{V_G}$, $\alpha'(Y) = \alpha(Y)$ for all $Y \neq X$ and $\alpha'(Y) = k$ when $Y = X$.
- $(G_\lambda, \alpha) \vDash \exists X(\phi)$ for some $K \in 2^{V_G}$, $(G_\lambda, \alpha') \vDash \phi$ where $\alpha' : free(\phi) \to V_G \cup 2^{V_G}$, $\alpha'(Y) = \alpha(Y)$ for all $Y \neq X$ and $\alpha'(Y) = k$ when $Y = X$.

Appendix B

Alternating Distributed Graph Automata in [8]

B.1 Alternating Distributed Graph Automata in [8]

Here, we discuss the definition of ADGA in [8]. In an NDGA, when a node in a state has more than one options for the same incoming states, the node nondeterministically chooses to go to one of the states. An ADGA contains universal states represented as rectangle as shown in fig B.1, any node of an input graph when in universal state and has more than one options, checks for an accepting run in each of the branches([3]).

Bounded ADGA: An ADGA $A = (Q, \delta, \sigma, F)$ is said to be bounded if there exists an $n \in \mathbb{N}$ which is computable from A such that for all graphs G and for all runs ρ of G on A, $length(\rho) \leq n$. The length of the ADGA A is defined as the minimum value of n such that $length(\rho) \leq n$ denoted by $len(A)$=n.

Nonemptiness problem: Bounded ADGA is proved to be equivalent to MSO logic in [8]. Since nonemptiness problem is undecidable for MSO, it is also undecidable for bounded ADGA.

B.1.1 Mixed and Pure configuration

If ρ_i maps the nodes to states containing both universal and nonuniversal states, then the configuration obtained is called mixed configuration. On the other hand, if ρ_i maps the nodes

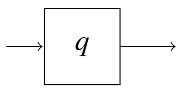


FIGURE B.1: Representation of universal state in an ADGA

to states containing only nonuniversal states, then the configuration obtained is called as pure configuration.

Claim: Any ADGA whose computation tree contains mixed configuration can be converted to an ADGA whose computation tree contains only pure configurations.

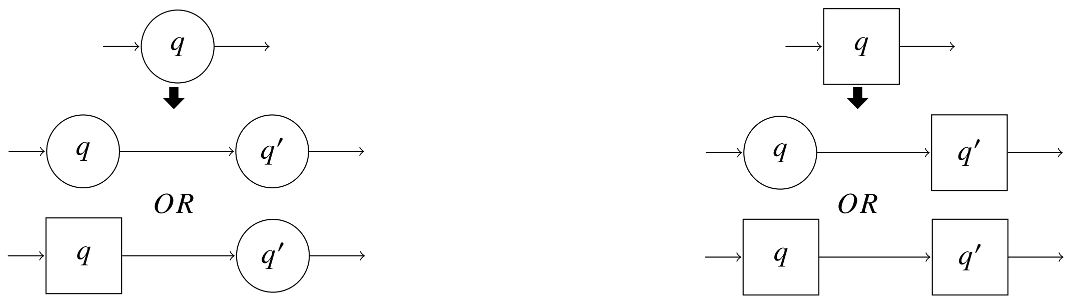


Figure B.2: Conversions used to construct DGA which generates only pure configuration

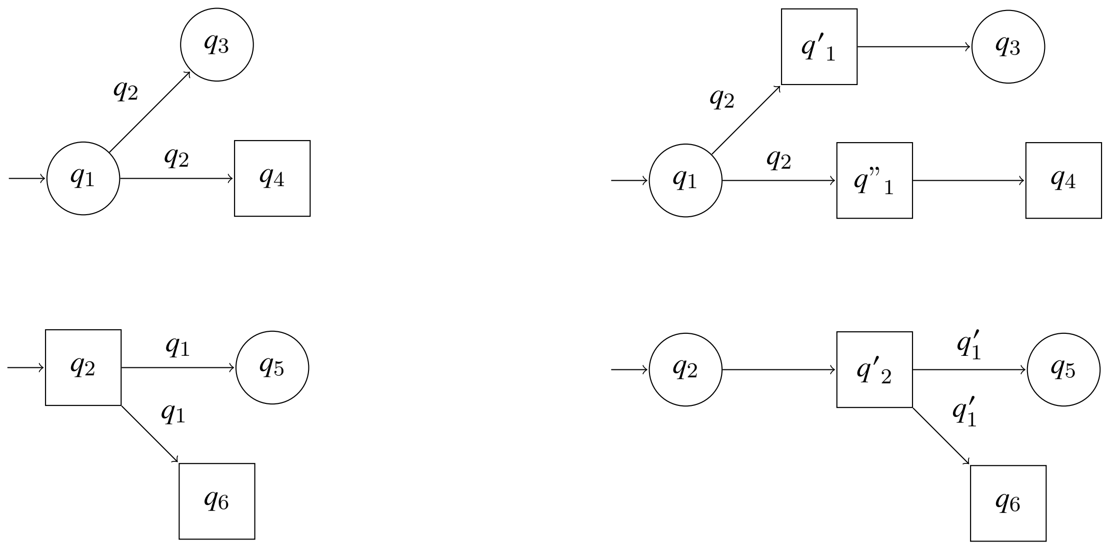


Figure B.3: ADGA which generates mixed configuration in the first step and it's equivalent ADGA which generates pure configuration for the first and second step

B.1.2 Converting Mixed configuration to pure configuration

Any ADGA whose computation tree contains mixed configurations can be converted to an ADGA whose computation tree contains only pure configuration using the conversion mentioned in B.3 We construct our new automata such that it contains alternate existential and universal states.

Claim: The language accepted by the original ADGA and the ADGA after conversion is the same.

An existential state, is chaged to existential state with the same state name and

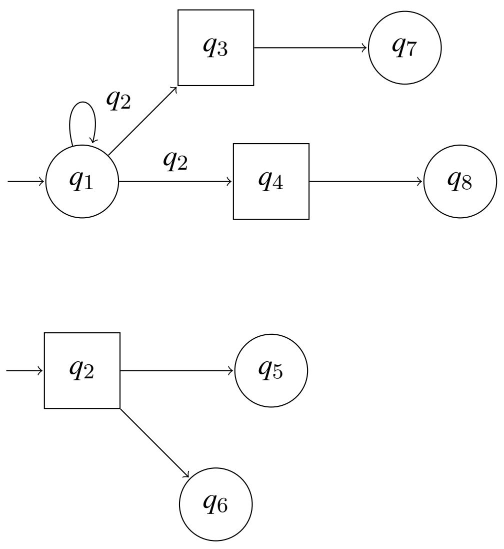


Figure B.4

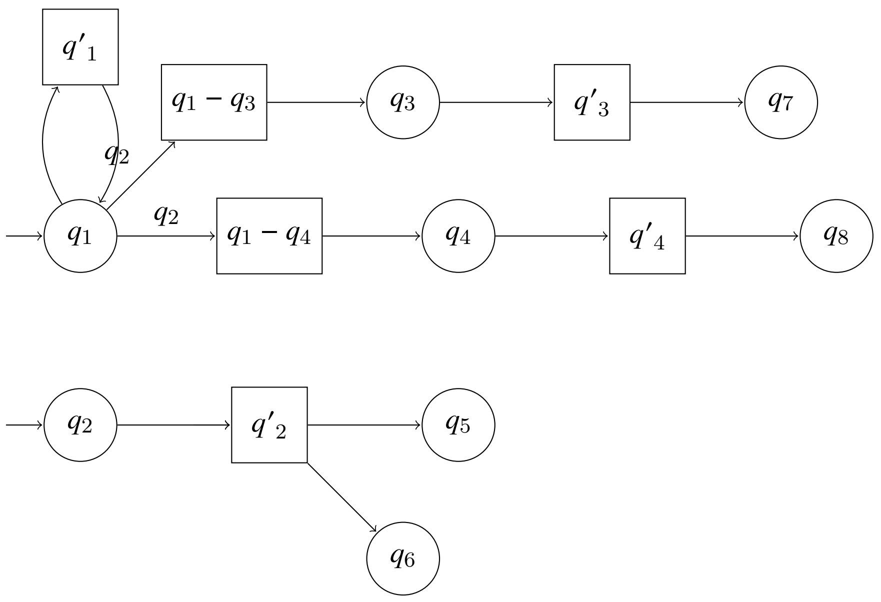


Figure B.5

B.1.3 Algorithm for membership Problem

For an input graph $G = (V, \lambda, E)$, the total number of distinct mappings ρ_i in a run ρ on an ADGA $A = (Q, \delta, \sigma, F)$ is $|Q|^{|V|}$. A tree can be constructed depicting all possible runs of a Graph on an ADGA A with ρ_0 being the root. The height of this tree is bounded by $|Q|^{|V|}$.

Step 1: Start with input Graph G and ADGA A.

step 2: Construct a pruned tree depicting all possible runs as follows:

step 2 a: ρ_0 is the root of the graph

step 2 b: each nondeterministic branch ρ_j of ρ_i is the child node of ρ_i where $i, j \in \mathbb{N} \cup \{0\}$

step 2 c: if for any ρ_j of the tree, ρ_i is the ancestor of ρ_j and $\rho_i = \rho_j$ and there exists no

ρ_k, ρ_l which are ancestors of ρ_j for which $\rho_k = \rho_l$, then ρ_j is the leaf node of that branch in the tree

Step 3: perform inorder traversal on the pruned tree and replace every universal state with $\wedge$ and every nonuniversal state with $\vee$ except the leaf nodes. For the leaf node ρ_j, check if ρ maps the input grpah to a set in F, if true then replace the node in the traversal result with 1, else replace with 0.

step 4: evaluate the traversal result as a boolean expression and result the Yes if the expression evaluates to 1, else return 0.

Bibliography

[1] Fabian Reiter Antti Kuusisto. "Emptiness Problems for Distributed Automata". In: *GandALF 2017: Rome* (2017).

[2] Howard Gutowitz. *Cellular Automata: Theory and Experiment.* MIT Press, 1991. ISBN: 9780262570862.

[3] Remi Gilleron Florent Jacquemard Denis Lugiez Christof Loding Sophie Tison Marc Tommasi Hubert Comon Max Dauchet. *Tree Automata Techniques and Applications.* 2008. URL: `http://tata.gforge.inria.fr/`.

[4] N. Immerman. "Nondeterministic space is closed under complementation". In: *SIAM Journal on Computing 17.* 1988.

[5] Dexter C. Kozen. *Automata and Computability.* Springer Science+Business Media,New York, 1999. ISBN: 9783642857089.

[6] Fabian Reiter. "Distributed Automata and Logic." In: *Computer Research Repository* (2018).

[7] Fabian Reiter. "Distributed Graph Automata." In: *LICS 2015:Kyoto,Japan* (2015).

[8] Fabian Reiter. "Distributed Graph Automata and Verification of Distributed Algorithms". In: *Computer Research Repository* (2014).

[9] Michael Sipser. *Introduction to the theory of computation.* third edition. Cengage Learning, 2012. ISBN: 9781133187790.

[10] R. Szelepcsényi. "The method of forcing for nondeterministic automata". In: *Bulletin of the EATCS 33.* 1987.