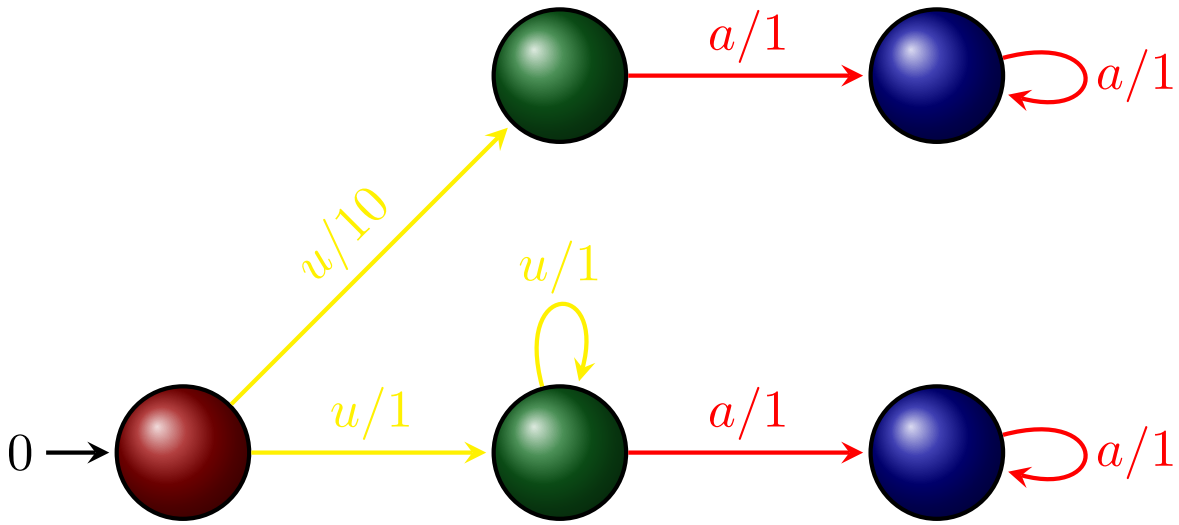
Technical University of Berlin

Finite-State Dynamical Systems



Kuize Zhang

Winter 2021/22

Contents

List of Figures

List of Tables

0 Acronyms

$\varnothing$	Empty set
$\mathbb{N}$	Set of natural numbers ($0 \in \mathbb{N}$)
$\mathbb{Z}$	Set of integers
$\mathbb{Z}_+$	Set of positive integers
$\mathbb{Q}$	Set of rational numbers
$\mathbb{Q}_+$	Set of positive rational numbers
$\mathbb{R}$	Set of real numbers
$\mathbb{R}_{\geqslant 0}$	Set of nonnegative real numbers
$\mathbb{R}^n$	Set of n-dimensional real column vectors
$\mathbb{R}^{m\times n}$	Set of $m \times n$ real matrices
$\mathcal{L}_{m\times n}$	Set of $m \times n$ logical matrices
$\lvert E\rvert$	Cardinality of set E
2^E	Power set of set E, i.e., $\{F \mid F \subset E\}$
A^B	Set of mappings from set B to set A
Id	Identity map
$\overline{A}$	Complement of set A
$A \cap B$	Intersection of sets A and B
$A \cup B$	Union of sets A and B
$A \cup B$	Disjoint union of sets A and B
$A \backslash B$	Difference of sets A and B
$A \subset B$ ($A \subseteq B$)	Set A is a subset of set B
$A \subsetneq B$	Set A is a strict subset of set B
$A \supset B$ ($A \supseteq B$)	Set A is a superset of set B
$A \supsetneq B$	Set A is a strict superset of set B
$[a, b]$	Closed interval with endpoints a and b ($a \leqslant b$)
$[\![a, b]\!]$	Set of integers no less than a and no greater than b ($a \leqslant b$)
(a, b)	Open interval with endpoints a and b ($a \leqslant b$)
$[a, b)$	$[a, b] \backslash \{b\}$
$(a, b]$	$[a, b] \backslash \{a\}$
S^*	Set of words over alphabet S
S^+	Set of words over alphabet S excluding the empty word ϵ
S^ω	Set of configurations over alphabet S

$\mathcal{D}$	$\{0,1\}$
$\operatorname{lcm}(p,q)$	Least common multiple of positive integers p and q
$\gcd(p,q)$	Greatest common divisor of positive integers p and q
I_n	Identity matrix of order n
δ_n^i	i-th column of the identity matrix I_n
$\mathbf{1}_n$	$\sum_{i=1}^{n} \delta_n^i$
Δ_n ($\Delta_2 =: \Delta$)	Set of the columns of the identity matrix I_n
A^{Tr}	Transpose of matrix A
$\operatorname{col}_i(A)$	i-th column of matrix A
$\operatorname{col}(A)$	Set of columns of matrix A
$\operatorname{row}_i(A)$	i-th row of matrix A
$\operatorname{row}(A)$	Set of rows of matrix A
$A = [\operatorname{Blk}_1(A), \ldots, \operatorname{Blk}_n(A)]$	All blocks $\operatorname{Blk}_i(A)$ have equally many columns
$\ltimes$	Semitensor product
$\otimes$	Kronecker product
$A_1 \oplus \cdots \oplus A_n$	$\begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_n \end{bmatrix}$

1 Introduction

The purpose of this course is to introduce three classes of dynamical systems with finitely many states as well as several basic topics among these classes that are studied mainly in theoretical computer science and control theory.

The first class of systems are finite automata. Finite automata are one fundamental model in theoretical computer science, the central topics in which are properties of regular languages — the formal languages recognized by finite automata. In this sense, transitions in finite automata are caused by reading letters. While in discrete-event systems (a branch of control theory), events-partially-observed (i.e., labeled) finite automata are studied in the sense that transitions are caused by occurrences of events (the events are the letters in finite automata). The material of this part includes basic knowledge on regular languages (in theoretical computer science) and inference-based properties (in control theory) in labeled finite automata.

The second class of systems are Boolean (control) networks. This class were initially proposed to model genetic regulatory networks, and during the recent two decades control properties therein have drawn wide attention. The material of this part is to introduce basic definitions on topological structures of Boolean networks and basic control properties of Boolean control networks.

Weighted automata are finite automata in which transitions carry weights, where the weights could have diverse practical meanings, e.g., time consumptions, position deviations, probabilities, etc. They have plenty of behaviors. The material of this part is to introduce only basic definitions.

2 Finite automata

2.1 Introduction

Theoretical computer science studies the mathematical foundation of computation. It provides a mathematical framework for the power and limitation of computing devices without going deep into their mechanical details. Roughly speaking, the framework characterizes whether problems are/are not solvable algorithmically, i.e., whether they can be solved concisely after a finite number of computations implemented on some computing device. In this framework, problems are formulated as *formal languages*, the tools — algorithms — that are used to solve problems, are formulated as all kinds of computing models such as *finite automata*, *pushdown automata*, and *Turing machines*. *A problem is solvable by an algorithm* is formulated as *a formal language is recognized by some computing model*. Turing machines are full-powered computing models, pushdown automata are intermediate-level models, finite automata are the most restricted models. We only introduce finite automata (and the formal languages recognized by them, which are called *regular languages*). For further reading on all the three models with their recognized formal languages, we refer to the reader to books [Sip96; HMU06; GJ90], etc.

2.2 Alphabets, words, and formal languages

An *alphabet* Σ is a nonempty finite set such that each finite sequence of elements of Σ has a unique decomposition of elements of Σ.

Example 1: The sets $\Sigma_1 = \{a, b\}$, $\Sigma_2 = \{0, 1\}$, $\Sigma_3 = \{0, 01\}$ are alphabets, but the set $\Sigma_4 = \{0, 00\}$ is not.

Choose a finite sequence *abaabaaab* of elements of Σ_1, it has a unique decomposition $a \cdot b \cdot a \cdot a \cdot b \cdot a \cdot a \cdot a \cdot b$.

Choose a finite sequence 001000100001 of elements of Σ_3, it has a unique decomposition $0 \cdot 01 \cdot 0 \cdot 0 \cdot 01 \cdot 0 \cdot 0 \cdot 0 \cdot 01$ of elements of Σ_3. For every sequence s of symbols 0's and 1's, one can check whether s is a sequence of elements of Σ_3 by repetitively performing the following procedure while reading s from left to right: at the current position (initially at the leftmost symbol of s), if a 1 is read, the answer is no; if a 0 is read and right after the 0 is also a 0, one puts a dot $\cdot$ between the two 0's and jumps to the latter 0; if a 0 is read and right after the 0 is a 1, one puts a dot $\cdot$ right after the 01 and jumps to the symbol right after the 01. One

then easily sees that for every sequence s of 0's and 1's, if s is a sequence of elements of Σ_3, then s has a unique decomposition of elements of Σ_3 which has been obtained after doing the above check. Hence Σ_3 is an alphabet.

Σ_4 is not alphabet because $000 = 0 \cdot 00 = 00 \cdot 0$, where $0, 00 \in \Sigma_4$. □

Let Σ be an alphabet. Elements of Σ are called *letters*, finite sequences of elements of Σ are called *words* or *strings* (*over* Σ). Particularly, ϵ is called the *empty word*.

Σ^* denotes the set of all words over the alphabet Σ. $\Sigma^+ := \Sigma^* \backslash \{\epsilon\}$. For example,

$$\{a, b\}^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\},$$
$$\{a\}^* = \{\epsilon, a, aa, aaa, \dots\}.$$

The *length* of a word $w \in \Sigma^*$, denoted by $|w|$, is defined by the number of its letters, where particularly $|\epsilon| =: 0$. For two words $u, v \in \Sigma^*$, their *concatenation* forms a new word uv. One sees for all $u, v, w \in \Sigma^*$, $(uv)w = u(vw) =: uvw$, $u\epsilon = \epsilon u = u$. For a word $w \in \Sigma^*$ and a natural number n, w^n denotes the concatenation of n copies of w. Particularly, $w^0 = \epsilon$. For alphabet Σ, Σ^n denotes the set of all n-length words over Σ, Then $\Sigma^0 = \{\epsilon\}$ (note that $\Sigma^0 \neq \varnothing$), $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$. For a word $w = w_1 w_2 \dots w_n \in \Sigma^*$, where $w_1, \dots, w_n \in \Sigma$, w^R denotes its *mirror image* $w_n \dots w_2 w_1$. A word $u \in \Sigma^*$ is called a *prefix* of the word $w \in \Sigma^*$, denoted by $u \sqsubset w$, if there is another word $v \in \Sigma^*$ such that $uv = w$, where v is called a *suffix* of w. A word $v \in \Sigma^*$ is called a *subword* of $z \in \Sigma^*$ if there exist $u, w \in \Sigma^*$ such that $uvw = z$.

Example 2: Recall Example 1. One has

over Σ_1,	$\lvert abaabaaab\rvert = 9$,	$(abaabaaab)^R = baaabaaba$;
over Σ_2,	$\lvert 001000100001\rvert = 12$,	$(001000100001)^R = 100001000100$;
over Σ_3,	$\lvert 001000100001\rvert = 9$,	$(001000100001)^R = 010000100010$.

One sees that the sequence 001000100001 of 0's and 1's has different lengths and mirror images if it is regarded as words over different alphabets Σ_2 and Σ_3. □

For an alphabet Σ, a *(formal) language* is a subset of Σ^*. The languages consisting of finitely many words are called *finite*, the languages consisting of infinitely many words are called *infinite*. Infinite languages are interesting.

Example 3: Examples of languages over the alphabet $\{a, b\}$ are shown as follows:

$$L_1 = \{a, ab, abb\},$$
$$L_2 = \{w | w \text{ ends with } aa\},$$
$$L_3 = \{w | w \text{ has an odd number of } a\text{'s}\},$$
$$L_4 = \{w | w \text{ contains } aa \text{ as a subword}\},$$
$$L_5 = \{a^n b^n | n \geqslant 0\},$$

$$L_6 = \{w | w = w^{\mathrm{R}}\} = \{w | w \text{ is a palindrome}\}.$$

A *palindrome* is a word such that it is the same as its mirror image, e.g., *radar*, *level*, *madam*, and *refer* are palindromes over the alphabet $\{a, b, \ldots, z\}$. □

2.3 Regular languages and deterministic finite automata

Assume we have a formal language L over an alphabet Σ, whether L can be represented finitely, a little more formally, whether L can be computed by some computing model M, where M can be finitely represented, is a fundamental question in theoretical computer science. If the answer is yes, then several properties of L can be analyzed by M algorithmically. Because L consists of words, if L can be computed by M, we can set a scenario in which M accepts every word of L but does not accept any word not in L. Regarding M as a dynamical system, which contains finitely many states for simplicity, the process of checking whether M accepts a given word w of L is intuitively described as: M starts from the initial state, and reads the letters of w one by one from left to right along with a state transition caused by reading each letter, then M accepts w if one of a special type of states is reached (after reading the whole w), and does not accept w if one of the other states is reached. Such a computing model M is usually called a *finite automaton* and such a language L is usually called a *regular language*. The above scenario is formulated as follows.

Formal definitions

Definition 1: A *deterministic finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of *states*,
2. Σ is an *alphabet*,
3. $\delta : Q \times \Sigma \to Q$ is the *transition function*,
4. $q_0 \in Q$ is the *initial state* (aka *start state*), and
5. $F \subset Q$ is the set of *final states* (aka *accept states*).

We use "DFA" to denote "deterministic finite automaton" for short. We extend the transition function $\delta : Q \times \Sigma \to Q$ to $\delta : Q \times \Sigma^* \to Q$ recursively as: for all $q \in Q$, $u \in \Sigma^*$, and $\sigma \in \Sigma$, one has $\delta(q, \epsilon) = q$, and $\delta(q, u\sigma) = \delta(\delta(q, u), \sigma)$.

With the formal definition of DFA, we give formal definitions for *a DFA accepts a word and recognizes a language*.

Definition 2: Let Σ be an alphabet and $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. A word $w \in \Sigma^*$ is called *accepted by DFA A* if $\delta(q_0, w) \in F$. The formal language *recognized by DFA A* is the set of all words accepted by A, i.e., $\{w \in \Sigma^* | \delta(q_0, w) \in F\}$. A formal language recognized by some DFA is called *regular*.

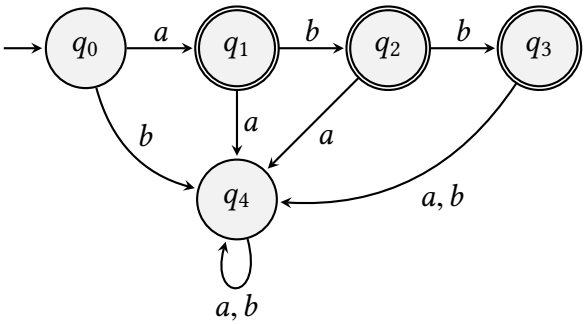
To check whether a DFA $A = (Q, \Sigma, \delta, q_0, F)$ accepts a word $w = w_1 \dots w_n \in \Sigma^*$, where $w_1, \dots, w_n \in \Sigma$, assume that A is in state q_0, and then A reads w_1 and moves to state $\delta(q_0, w_1) =: q_1$; and then A reads w_2 and moves to state $\delta(q_1, w_2) =: q_2, \dots$, finally A reads w_n and moves to state $\delta(q_{n-1}, w_n) =: q_n$. A accepts w if and only if q_n is a final state. Particularly, A accepts the empty word ϵ if and only if $q_0 \in F$.

A DFA can be graphically represented as a *state diagram*, by using which one can see how a DFA works intuitively. In a state diagram, states are represented by circles with names inside themselves, where particularly the final states are represented by double circles, and the initial state is indicated by an input arrow from nowhere. To draw a transition $q \xrightarrow{\sigma} q'$ (that is, $q, q' \in Q$, $\sigma \in \Sigma$, $q' = \delta(q, \sigma)$), an arrow from q to q' labeled by σ is drawn.

Example 4: Consider the DFA $A_1 = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$, $F = \{q_1, q_2, q_3\}$, δ is described as

	a	b
q_0	q_1	q_4
q_1	q_4	q_2
q_2	q_4	q_3
q_3	q_4	q_4
q_4	q_4	q_4

The state diagram of A_1 is depicted as



By the state diagram, one sees that when A_1 reads a, A_1 moves to q_1, a final state, so A_1 accepts the word a. Similarly, A_1 accepts words ab and abb. One also sees A_1 does not accept any other word. Hence A_1 recognizes the language L_1 in Example 3. □

The above DFA A_1 recognizes a finite language. So A_1 is trivial. However, there exist DFAs that recognize all kinds of infinite languages and some of them are nontrivial and interesting. Hence although DFAs are the most restricted computing model, they have relatively strong computing power.

In the sequel, we do not distinguish between a DFA and its state diagram.

Example 5: Consider the DFA A_2 as in Figure 2.1.

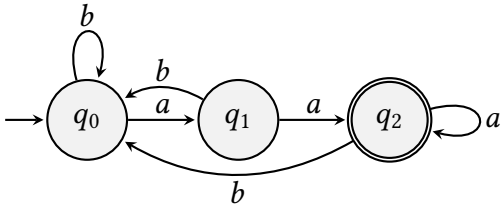


Figure 2.1: The DFA A_2 studied in Example 5.

In A_2, all transitions ending with the unique final state q_2 are $q_1 \xrightarrow{a} q_2$ and $q_2 \xrightarrow{a} q_2$, and the unique transition ending with q_1 is $q_0 \xrightarrow{a} q_1$. Hence all transition sequences of length 2 ending with q_2 are $q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2$, $q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_2$, and $q_2 \xrightarrow{a} q_2 \xrightarrow{a} q_2$. Hence all words accepted by A_2 end with aa. On the other hand, for every word waa, where $w \in \Sigma^*$, no matter $\delta(q_0, w)$ is equal to q_0, q_1, or q_2, A_2 accepts waa. Hence A_2 exactly accepts all words ending with aa. That is, A_2 recognizes the language L_2 in Example 3. □

Designing deterministic finite automata

Generally, for a given language L, it is difficult to construct a DFA A recognizing L, because one does not know whether L is regular, and even if L is regular, L may be very complicated. Despite the difficulty, for several particular languages, one can find methods to do this.

Consider the language L_3 in Example 3, which consists of all words over the alphabet $\{a, b\}$ that have an odd number of a's. We do not know whether L_3 is regular, and try to find some method to construct a DFA recognizing L_3. If we succeeded, then L_3 is regular; however, if we failed, we cannot make sure that L_3 is not regular.

Assume a DFA A_3 recognizing L_3. While A_3 is reading a given word of $\{a, b\}^*$, A_3 can only be in one of two states: (1) A_3 has read an even number of a's (denoted by q_{even}) and (2) A_3 has read an odd number of a's (denoted by q_{odd}). When A_3 is in state q_{even}, if an a is read then A_3 transitions to state q_{odd}, otherwise if a b is read then A_3 remains in state q_{even}. Similarly, when A_3 is in state q_{odd}, if an a is read then A_3 transitions to state q_{even}, otherwise if a b is read then A_3 remains in state q_{odd}. Initially, A_3 must be in state q_{even}, because A_3 has read no a's, hence q_{even} is the initial state. Apparently, q_{odd} is the final state because we

want A_3 to accept all words containing an odd number of a's. Then we obtain A_3 shown in Figure 2.2.

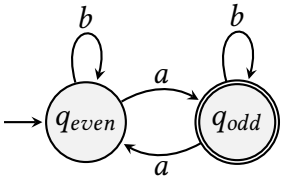


Figure 2.2: A DFA A_3 recognizing the language L_3 in Example 3.

Now we consider to construct a DFA A_2' that recognizes the more complicated language L_2 (in Example 3) over $\{a, b\}$ consisting of all words ending with aa, by imitating the procedure of constructing the above A_3. While A_2' is reading a given word over $\{a, b\}$, it can be in three states: (1) q_ϵ (it has not read any word, or has read any word ending with b), (2) q_a (it has just read an a, and before the a it might have not read any word or might have read any word ending with b), and q_{aa} (it has just read the subword aa, and before the aa, it might have not read any word or might have read any word ending with b). q_ϵ is the initial state, q_{aa} is the final state. The transitions are defined as follows: First, when A_2' is in state q_ϵ, if an a is read then A_2' transitions to q_a, otherwise if a b is read then A_2' remains in q_ϵ (in this case A_2' has just read a b). Second, when A_2' is in state q_a, if an a is read then A_2' transitions to q_{aa} (in the case, A_2' has just read the pattern aa), and if a b is read then A_2' moves back to q_ϵ. Third, when A_2' is in state q_{aa}, if an a is read then A_2' remains in q_{aa}, if a b is read then it moves back to q_ϵ. Then we obtain the A_2' as in Figure 2.3.

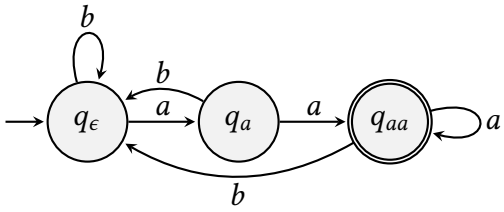


Figure 2.3: A DFA A_2' recognizing the language L_2 in Example 3.

One sees that the DFA A_2' is the same as the DFA A_2 in Figure 2.1 up to renaming the states.

Similarly, one can construct the following DFA A_4 in Figure 2.4 that recognizes the language L_4 (in Example 3) over $\{a, b\}$ consisting of all words containing aa as a subword.

Remark 1: From the above procedure of constructing DFA A_2' (resp., A_4) to recognize the language L_2 (resp., L_4), one sees that it is not very easy to do the construction. One way to simplify the construction process is to allow an automaton to have *nondeterminism*, that is, from a state q, there may exist multiple (zero, one, or more than one) transitions labeled by the same letter σ. Such an extension will make the process of constructing an automaton

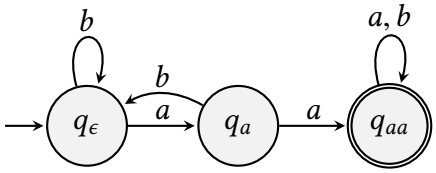


Figure 2.4: A DFA A_4 recognizing the language L_4 in Example 3.

easier and more flexible. Taking the language L_4 consisting of all words over $\{a, b\}$ containing aa as a subword for example, one can construct a "generalized automaton" recognizing L_4 in the following trivial way: (1) On the initial state q_ϵ, a self-loop is added labeled by a, b, meaning that when reading an arbitrary word over $\{a, b\}$, the automaton *can always* remain in state q_ϵ. (2) From q_ϵ, add a transition $q_\epsilon \xrightarrow{a} q_a$ and then add a transition $q_a \xrightarrow{a} q_{aa}$, meaning that when the pattern aa is read, the automaton can transition to q_{aa}. (Note that differently from a DFA, from state q_ϵ there are two transitions labeled by the same letter a.) (3) Finally, on q_{aa}, a self-loop is added labeled by a, b, and set q_{aa} to be the final state. See Figure 2.5. Intuitively, one sees that the "generalized automaton" accepts all words of the form $uaav$, where $u, v \in \{a, b\}^*$ are arbitrary. One also sees that the DFA A_4 and the "generalized automaton" A'_4 both recognize the language L_4, but it is much easier to construct A'_4 than to construct A_4.

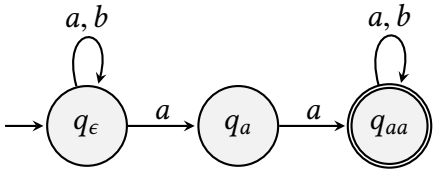


Figure 2.5: A "generalized automaton" A'_4 recognizing the language L_4 in Example 3.

Such an extension yields a more general notion of *nondeterministic finite automaton* (NFA). In the next subsection, we give formal definitions of NFA and an NFA accepts a word and characterize the languages recognized by NFAs. □

2.4 Nondeterministic finite automata

A nondeterministic machine generalizes a deterministic machine, because after reading a word, the former may generate several (one or more than one) transition sequences, hence a word may lead to several states (zero, one, or more than one, here "zero" means it is possible that the nondeterministic machine could not read the whole word, but halted after reading some of its intermediate letters.). In order to formulate whether a word is accepted, one usual way is to define a word to be accepted if it leads to some final state (it is not required that all states it leads to are final states).

Formal definitions

Definition 3: A *nondeterministic finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of *states*,
2. Σ is an *alphabet*,
3. $\delta : Q \times \Sigma \to 2^Q$ is the *transition function*,
4. $q_0 \in Q$ is the *initial state*, and
5. $F \subset Q$ is the set of *final states*.

There exists a transition $q \xrightarrow{a} q'$ if and only if $q' \in \delta(q, a)$. When an NFA A is in a state q and reads a letter a, then A can transition to any state of $\delta(q, a)$. We extend the transition function $\delta : Q \times \Sigma \to 2^Q$ to $\delta : Q \times \Sigma^* \to 2^Q$ recursively as: for all $q \in Q$, $u \in \Sigma^*$, and $a \in \Sigma$, one has $\delta(q, \epsilon) = \{q\}$, and $\delta(q, ua) = \bigcup_{p \in \delta(q,u)} \delta(p, a)$.

With the formal definition of NFA, we give formal definitions for *an NFA accepts a word and recognizes a language*.

Definition 4: Let Σ be an alphabet and $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA. A word $w \in \Sigma^*$ is called *accepted by* A if $\delta(q_0, w) \cap F \neq \varnothing$. The formal language *recognized by* A is the set of all words accepted by A, i.e., $\{w \in \Sigma^* | \delta(q_0, w) \cap F \neq \varnothing\}$.

In other words, for an NFA $A = (Q, \Sigma, \delta, q_0, F)$ over an alphabet Σ and a word $w = w_1 \dots w_n \in \Sigma^*$, where $w_1, \dots, w_n \in \Sigma$, A accepts w if and only if there exist states $q_1, \dots, q_n$ such that $q_i \in \delta(q_{i-1}, w_i)$ for all $1 \leqslant i \leqslant n$ and $q_n \in F$.

Example 6: By definition, the "generalized automaton" A_4' in Figure 2.5 is an NFA and does recognize the language L_4. Every word accepted by A_4' must contain aa, because once the final state q_{aa} is reached, transition sequence $q_\epsilon \xrightarrow{a} q_a \xrightarrow{a} q_{aa}$ must have been generated. For every word $uaav$, where $u = u_1 \dots u_n \in \{a, b\}^*$ and $v = v_1 \dots v_m \in \{a, b\}^*$ are arbitrary, $uaav$ can lead to the final state q_{aa} via transition sequence

$$q_\epsilon \xrightarrow{u_1} \cdots \xrightarrow{u_n} q_\epsilon \xrightarrow{a} q_a \xrightarrow{a} q_{aa} \xrightarrow{v_1} \cdots \xrightarrow{v_m} q_{aa}.$$

Note that $uaav$ may also lead to state q_ϵ that is not the final state, but this does not mean that $uaav$ is not accepted by A_4'. Taking $aabba$ for example, one computes as

$$\begin{aligned}
\delta(q_\epsilon, a) &= \{q_\epsilon, q_a\}, \\
\delta(q_\epsilon, aa) &= \delta(q_\epsilon, a) \cup \delta(q_a, a) = \{q_\epsilon, q_a, q_{aa}\}, \\
\delta(q_\epsilon, aab) &= \delta(q_\epsilon, b) \cup \delta(q_a, b) \cup \delta(q_{aa}, b) = \{q_\epsilon, q_{aa}\},
\end{aligned}$$

$$\delta(q_\epsilon, aabb) = \delta(q_\epsilon, b) \cup \delta(q_{aa}, b) = \{q_\epsilon, q_{aa}\},$$
$$\delta(q_\epsilon, aabba) = \delta(q_\epsilon, a) \cup \delta(q_{aa}, a) = \{q_\epsilon, q_a, q_{q_{aa}}\}.$$

Hence the word *aabba* leads A_4' to three states q_ϵ, q_a, and q_{aa}. Because the final state q_{aa} belongs to $\delta(q_\epsilon, aabba)$, A_4' accepts *aabba*. □

Example 7: Reconsider the language L_2 (in Example 3) over $\{a, b\}$ consisting of all words ending with *aa*. It is easy to construct the following NFA A_2'' recognizing L_2 (in Figure 2.6), which is similar to the DFA A_2' in Figure 2.3.

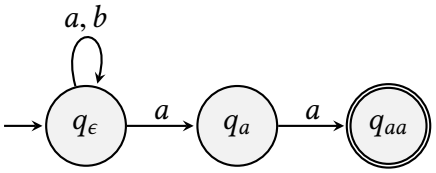


Figure 2.6: An NFA A_2'' recognizing the language L_2 in Example 3.

□

Equivalence of NFAs and DFAs

Although NFAs are more general than DFAs, NFAs do not recognize more languages than DFAs. Actually NFAs exactly recognize the regular languages. To see this, given an NFA, we can compute an *equivalent* DFA which recognizes the same language as the NFA. This property is useful, because when one wants to find a DFA that recognizes a given regular language, he/she can first construct an NFA that recognizes the language (it is easier to do so than to directly construct a DFA, see Remark 1), and then transform the NFA to its equivalent DFA.

Theorem 2.1: *Every nondeterministic finite automaton has an equivalent deterministic finite automaton.*

Proof. Consider an NFA $A_N = (Q, \Sigma, \delta, q_0, F)$ that recognizes a language L. We next construct a DFA A_D from A_N that also recognizes L. The idea of constructing A_D is keeping track of every word, meanwhile, collecting all states the word can lead to. That is, states of A_D are subsets of Q. A state of A_D is a final state if and only if it contains a final state of A_N. This construction results in that the state set of A_D is the powerset of Q, so the construction is called *powerset construction* (sometimes also called *subset construction*).

Formally, $A_D = (Q', \Sigma, \delta', q_0', F')$, where

1. $Q' = 2^Q$,

2. for all $x \in Q'$ and $a \in \Sigma$, $\delta'(x, a) = \bigcup_{q \in x} \delta(q, a)$,
3. $q_0' = \{q_0\}$,
4. $F' = \{x \in Q' | x \cap F \neq \varnothing\}$.

By the construction, one sees for every word $w \in \Sigma^*$, $\delta(q_0, w) \cap F \neq \varnothing$ if and only if $\delta'(q_0', w) \in F'$. Hence A_N and A_D recognize the same language. One also sees $\varnothing \in Q'$ is never *reachable* in A_D, e.g., there exists no $w \in \Sigma^*$ such that $\delta'(q_0', w) = \varnothing$. □

Example 8: Recall the NFA A_4' in Figure 2.5. By the proof of Theorem 2.1, we compute its equivalent DFA A_{4D}' shown in Figure 2.7.

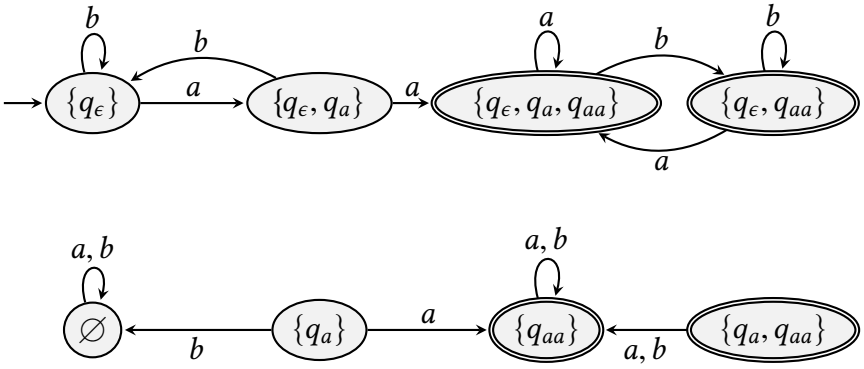


Figure 2.7: A DFA A_{4D}' that is equivalent to the NFA A_4' in Figure 2.5.

□

Reducing the size of a DFA equivalently

We have shown that the DFA A_4 in Figure 2.4 and the NFA A_4' in Figure 2.5 both recognize the language L_4 in Example 3. Then by Theorem 2.1, the DFA A_{4D}' in Figure 2.7 is equivalent to A_4. Although we used an easier way to construct A_{4D}' compared with directly constructing A_4, the size of A_{4D}' is larger than that of A_4. Is it possible to reduce the size of A_{4D}' equivalently? The answer is yes. Next we show how to do so.

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a given DFA. A state $q \in Q$ is called *reachable* if there exists $w \in \Sigma^*$ such that $q = \delta(q_0, w)$. Particularly, q_0 is reachable because $\delta(q_0, \epsilon) = q_0$.

In the first step, we remove all unreachable states of A, the remaining part is trivially equivalent to A.

To proceed, some additional notions are needed.

Definition 5: Let X be a set. A *partition* of X is a set of pairwise disjoint nonempty subsets of X whose union is equal to X, where these subsets are called the *parts* (or *cells*) of the partition. A partition of X is *finite* if it consists of finitely many subsets of X. For two partitions α, β of X, we say α *refines* β, or α *is finer than* β, or α *is a refinement of* β, if for every $M \in \alpha$, there exists $N \in \beta$ such that $M \subset N$.

In the second step, the target is to find a partition E_Q of Q such that (1) each of its cells contains only final states or only non-final states, (2) with reading each letter, all states in each cell go to the same cell, i.e., for every $M \in E_Q$ and $a \in \Sigma$, there exists $N \in E_Q$ such that $\delta(q, a) \in N$ for all $q \in M$. Then regarding each cell as a new state, and replacing the transitions of A by transitions between the cells of E_Q correspondingly, we obtain a new DFA E_A that is no greater than but equivalent to A.

The procedure of computing a partition E_Q and a reduced DFA E_A is as follows:

Procedure 1: Input a DFA $A = (Q, \Sigma, \delta, q_0, F)$ with no unreachable states.

(1) If $F = \varnothing$ or $F = Q$ then set $E_Q = \{Q\}$, go to (4); otherwise construct the initial partition $\{F, Q\backslash F\}$ of Q, go to (2).

(2) If there exists a cell M of the current partition α and a letter $a \in \Sigma$ such that with reading a, some states in M go to different cells, then partition M into several new cells $M_1, \ldots, M_n$ such that for every M_i, with reading a, all states of M_i go to the same cell of α, and for all different i, j, the states of M_i and M_j go to different cells of α. A new refined partition $(\alpha\backslash\{M\}) \cup \{M_1, \ldots, M_n\}$ is obtained.

(3) Repeat (2) until for the current partition there exist no such M and a as above, then the current partition is denoted by E_Q.

(4) By using E_Q, construct a reduced DFA

$$E_A = (E_Q, \Sigma, \delta', q_0', F'), \tag{2.1}$$

where

a) for every $M \in E_Q$ and $a \in \Sigma$, $\delta'(M, a) = N \in E_Q$, where for every $q \in M$, $\delta(q, a) \in N$,

b) $q_0 \in q_0' \in E_Q$,

c) $F' = \{M \in E_Q | M \subset F\}$.

Proposition 2.2: *A DFA $A = (Q, \Sigma, \delta, q_0, F)$ with no unreachable states is equivalent to any reduced DFA $E_A = (E_Q, \Sigma, \delta', q_0', F')$ returned by Procedure 1.*

Proof. Consider a word w in Σ^*. We check that $\delta(q_0, w) \in F$ if and only if $\delta'(q_0', w) \in F'$. This holds for $w = \epsilon$ because $\delta(q_0, \epsilon) \in F \iff q_0 \in F \iff q_0' \in F' \iff \delta'(q_0', \epsilon) \in F'$. Next we assume $w = w_1 \dots w_n \in \Sigma^+$, where $w_1, \dots, w_n \in \Sigma$.

"only if": Assume $\delta(q_0, w) \in F$, then there exist states $q_1, \dots, q_n \in Q$ such that $q_i = \delta(q_{i-1}, w_i)$ for all $1 \leqslant i \leqslant n$ and $q_n \in F$. For each $1 \leqslant i \leqslant n$, choose the $M_i \in E_Q$ such that $q_i \in M_i$, then $\delta'(q_0', w_1) = M_1$, $\delta'(M_1, w_2) = M_2$, …, $\delta'(M_{n-1}, w_n) = M_n \in F'$. That is, $\delta'(q_0', w) \in F'$.

"if": Assume $\delta'(q_0', w) \in F'$, then there exist states $M_1, \dots, M_n \in E_Q$ such that $\delta'(q_0', w_1) = M_1$, $\delta'(M_1, w_2) = M_2$, …, $\delta'(M_{n-1}, w_n) = M_n \in F'$. Then one has $\delta(q_0, w_1) =: q_1 \in M_1$, $\delta(q_1, w_2) =: q_2 \in M_2$, …, $\delta(q_{n-1}, w_n) =: q_n \in M_n \subset F$. That is, $\delta(q_0, w) \in F$. □

Example 9: Reconsider the DFA A'_{4D} in Figure 2.7. We use Procedure 1 to obtain a reduced equivalent DFA. First, we remove the unreachable states $\varnothing$, $\{q_a\}$, $\{q_{aa}\}$, and $\{q_a, q_{aa}\}$. Second, we partition the remaining states as two cells $\{\{q_\epsilon\}, \{q_\epsilon, q_a\}\} =: M_1$ consisting of all non-final states and $\{\{q_\epsilon, q_a, q_{aa}\}, \{q_\epsilon, q_{aa}\}\} = M_2$ consisting of all final states. Choose M_1 and letter a, one sees with reading a, $\{q_\epsilon\}$ remains in M_1, but $\{q_\epsilon, q_a\}$ goes into M_2. So, we further partition M_1 to $M_1^1 = \{\{q_\epsilon\}\}$ and $M_1^2 = \{\{q_\epsilon, q_a\}\}$. For M_2, with reading either a or b, all states of M_2 remain in M_2, so we will not partition M_2. Thus, we obtain $E_Q = \{M_1^1, M_1^2, M_2\}$, and a reduced equivalent DFA $E_{A'_{4D}}$ which is shown in Figure 2.8. The reduced DFA $E_{A'_{4D}}$ is the same as the DFA A_4 in Figure 2.4 up to renaming states.

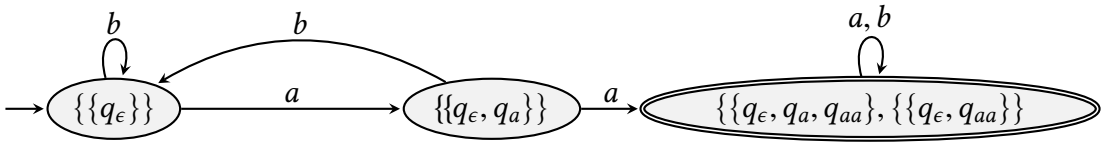


Figure 2.8: A reduced DFA $E_{A'_{4D}}$ that is equivalent to the DFA A'_{4D} in Figure 2.7.

□

Let us recall the regular language L_4 in Example 3 that consists of all words over $\{a, b\}$ containing aa as a subword. In order to find a DFA that recognizes L_4, we adopted two methods:

(1) By the feature of L_4, directly construct the DFA A_4 shown in Figure 2.4.

(This method directly yields the result, but needs indepth thinking of the feature of L_4, so it is not easy to use this method, particularly when a regular language is complicated.)

(2) a) Construct an NFA A'_4 in Figure 2.5 that recognizes L_4.

(This is a straightforward process, and does not need indepth thinking of L_4.)

b) Convert A'_4 to its equivalent DFA A'_{4D} in Figure 2.7.

(By the proof of Theorem 2.1, this process is also straightforward, but the obtained DFA is larger than A_4.)

c) Reduce the size of A'_{4D} equivalently and obtain a smaller equivalent DFA $E_{A'_{4D}}$ in Figure 2.8.

(This process is also straightforward, and $E_{A'_{4D}}$ is the same as A_4 up to renaming states.)

By the above two methods, in order to construct a DFA that recognizes a given regular language, if the language itself is not very complicated then one could choose the first method; otherwise, one would better choose the second one, because although the second method is complicated (containing three steps), every step is easier to be implemented.

Procedure 1 returns a *unique* DFA up to renaming states. To see this, we show the *Myhill-Nerode theorem*. This theorem provides an equivalent condition for a language to be regular, and also characterizes the DFA with minimal number of states that recognizes a regular language.

2.5 The Myhill-Nerode theorem

Before showing the Myhill-Nerode theorem, additional notions are needed.

On a given set X, a *relation* $\sim$ is a subset of $X \times X$. We say *two states* $x, x' \in X$ *have relation* $\sim$ (or x *has relation* $\sim$ *with* x') if $(x, x') \in \sim$, which is denoted by $x \sim x'$. A relation $\sim \subset X \times X$ is an *equivalence relation* if it is reflexive, symmetric, and transitive. That is, for all $x, y, z \in X$,

1. $x \sim x$ (reflexivity),
2. $x \sim y$ if and only if $y \sim x$ (symmetry), and
3. if $x \sim y$ and $y \sim z$ then $x \sim z$ (transitivity).

For $x \in X$, the *equivalence class* generated by x is defined by $[x]_\sim = \{y \in X | y \sim x\}$, i.e., the set of all elements of X that have the relation $\sim$ with x. One will see for all $x, y \in X$, either $[x]_\sim = [y]_\sim$ or $[x]_\sim \cap [y]_\sim = \emptyset$; $[x]_\sim = [y]_\sim$ if and only if $x \sim y$. Then the set $\{[x]_\sim\}_{x \in X}$ of all equivalence classes forms a partition of X, which is called the partition induced by relation $\sim$. The *index* of an equivalence relation is the number of its equivalence classes. The index of an equivalence relation could be finite or infinite.

For two equivalence relations α, β on a set X, we say α *refines* β if the partition of X induced by α refines the partition of X induced by β, equivalently, for all x, y in X, if $x \alpha y$ then $x \beta y$. Apparently if α refines β then the index of α is no less than the index of β.

On the set $\mathbb{Z}$ of integers, "=" is an equivalence relation and has infinite index. For each $z \in \mathbb{Z}$, the equivalence class generated by z is the singleton $\{z\}$.

On $\mathbb{Z}$, the equivalence relation $a \equiv b \pmod 2$ has index 2. Its two equivalence classes are the set of odd numbers and the set of even numbers.

Let Σ be an alphabet, and let $L \subset \Sigma^*$ be a formal language. On the set Σ^*, we use L to define an equivalence relation $\sim_L$: for all words $u, v \in \Sigma^*$, $u \sim_L v$ if and only if for all $x \in \Sigma^*$, $ux \in L \iff vx \in L$. Words u and v have the relation $\sim_L$ if and only if exactly the same extensions take them into L.

For a DFA $A = (Q, \Sigma, \delta, q_0, F)$, we also define an equivalence relation $\sim_A$ on Σ^*: for all words $u, v \in \Sigma^*$, $u \sim_A v$ if and only if $\delta(q_0, u) = \delta(q_0, v)$. The equivalence relation $\sim_A$ has finite index no greater than $|Q|$. Particularly when all states are reachable, $\sim_A$ has index $|Q|$. Relation $\sim_A$ refines the equivalence relation $\sim_L$, where L is the regular language recognized by A, because for all $u, v \in \Sigma^*$, if $u \sim_A v$ then $u \sim_L v$.

For a regular language, call a DFA with the minimal number of states recognizing the language a *minimum-state* DFA. Next we state and prove the Myhill-Nerode theorem.

Theorem 2.3 (Myhill-Nerode)**:** *Let Σ be an alphabet and $L \subset \Sigma^*$ be a formal language.*

(1) L is regular if and only if the equivalence relation $\sim_L$ has finite index.

(2) If $\sim_L$ has finite index n, then a minimum-state DFA recognizing L has exactly n states.

Proof. (1): "only if": Assume L is regular. Then there exists a DFA $A = (Q, \Sigma, \delta, q_0, F)$ recognizing L. One sees that for all words $u, v \in \Sigma^*$, if $\delta(q_0, u) = \delta(q_0, v)$, then u and v have the equivalence relation $\sim_L$. Hence equivalence relation $\sim_A$ refines relation $\sim_L$. Then the index of $\sim_L$ is no greater than the index of relation $\sim_A$, hence finite.

"if": Assume the relation $\sim_L$ has finite index n. We next construct a DFA A that recognizes L. The partition $\{[u]_{\sim_L} | u \in \Sigma^*\}$ induced by $\sim_L$ has cardinality n by definition, and is set to be the state set of A. We use $[u]$ to denote $[u]_{\sim_L}$ for short. The alphabet is Σ. The initial state is $[\epsilon]$. The transition function δ is describe as follows: for all $u \in \Sigma^*$ and $a \in \Sigma$, $\delta([u], a) = [ua]$. δ is well defined because if $[u] = [v]$ then $[ua] = [va]$, i.e., the definition does not depend the word we chose from the equivalence class $[u]$. (If $[ua] \neq [va]$, then for some $x \in \Sigma^*$, either $uax \in L$ and $vax \notin L$, or $uax \notin L$ and $vax \in L$, then $[u] \neq [v]$.) The set of final states is $\{[u] | u \in \Sigma^*, u \in L\}$. The set of final states is also well defined because if $[u] = [v]$ and $u \in L$ then $v \in L$. (If $u \in L$ and $v \notin L$, then $u\epsilon \in L$ and $v\epsilon \notin L$, $[u] \neq [v]$.)

(2): Assume $\sim_L$ has finite index n. By (1) we know that L is regular. By the proof of the "if" part of (1), there exists a DFA that recognizes L and has exactly n states. Then a minimum-state DFA recognizing L has at most n states. By the proof of the "only if" part of (1), if a DFA recognizes L, then it has at least n states. Then a minimum-state DFA recognizing L has at least n states. Hence a minimum-state DFA recognizing L has exactly n states. □

Furthermore, by Theorem 2.3, we can prove the uniqueness of minimum-state DFA rec-

ognizing a given regular language.

Theorem 2.4: *A regular language has a unique minimum-state DFA (up to renaming states).*

Proof. Assume a regular language L over a given alphabet Σ, and two minimum-state DFAs A_1 and A_2 also over Σ recognizing L. By Theorem 2.3, the equivalence relation $\sim_L$ on Σ^* has finite index, denoted by n, and both A_1 and A_2 have exactly n states. As mentioned before, the equivalence relations $\sim_{A_1}$ and $\sim_{A_2}$ (also on Σ^*) both refine $\sim_L$. Note that the partitions induced by $\sim_{A_1}$ and $\sim_{A_2}$ both have cardinality n. Hence $\sim_{A_1}=\sim_{A_2}=\sim_L$.

Denote $A_i = (Q_i, \Sigma, \delta_i, q_0^i, F_i)$, $i = 1, 2$. Then $|Q_1| = |Q_2|$. Define a function

$$f : Q_1 \to Q_2, \quad q_1 \mapsto \delta_2(q_0^2, w),$$

where $q_1 \in Q_1$, $w \in \Sigma^*$ is such that $\delta_1(q_0^1, w) = q_1$. f is well defined and injective: for all $u, v \in \Sigma^*$, $\delta_1(q_0^1, u) = \delta_1(q_0^1, v)$ if and only if $\delta_2(q_0^2, u) = \delta_2(q_0^2, v)$ ("only if" implies f is well defined, "if" implies f is injective), because $\sim_{A_1}=\sim_{A_2}$. Particularly, $f(q_0^1) = q_0^2$. Then f is also surjective, because Q_1, Q_2 are finite sets and have the same cardinality. By the construction of f, if we replace each state q_1 of A_1 by $f(q_1)$, then A_1 coincides with A_2. □

Next we show for a given regular language L and a DFA A recognizing L, how to construct the minimum-state DFA recognizing L. Actually, this had been done in Procedure 1!

Theorem 2.5: *After Procedure 1 receives a DFA $A = (Q, \Sigma, \delta, q_0, F)$ with no unreachable states, it returns the unique minimum-state DFA $E_A = (E_Q, \Sigma, \delta', q_0', F')$ (shown in (2.1)) recognizing the language recognized by A up to renaming states.*

Proof. By Proposition 2.2, we have known that A and E_A recognize the same language L. Hence the equivalence relation $\sim_{E_A}$ refines the equivalence relation $\sim_L$ by the proof of Theorem 2.3. Next we show E_A is the minimum-state DFA recognizing L (up to renaming states).

We first claim that for every two different states $M, N \in E_Q$, there exists $w \in \Sigma^*$ such that $\delta'(M, w)$ and $\delta'(N, w)$ must have a final state and a non-final state, which we call w *distinguishes M and N*. We prove this claim via the process of generating E_Q in Procedure 1. In the first division, Q is divided into F and $Q\backslash F$. With reading ϵ, elements of F remain in F, elements of $Q\backslash F$ remain in $Q\backslash F$ (see Figure 2.9). Hence ϵ distinguishes F and $Q\backslash F$. The current partition is $\{F, Q\backslash F\}$. In the second division, without loss of generality, we assume that $Q\backslash F$ is divided into two cells Q_1 and Q_2, and according to the working principle of Procedure 1, there exists $a \in \Sigma$ such that with reading a, all elements of Q_1 go to the one of F and $Q\backslash F$, and all elements of Q_2 go to the other of them (see Figure 2.10). Hence a distinguishes Q_1 and Q_2. The current partition is $\{F, Q_1, Q_2\}$. In the third division, without loss of generality, we assume that Q_1 is divided into two cells Q_1^1 and Q_1^2, then similarly there exists $a' \in \Sigma$ such

that with reading a', all elements of Q_1^1 go to one of F, Q_1, and Q_2, and all elements of Q_1^2 go to another of them (see Figure 2.11). If elements of Q_1^1 or Q_1^2 go to F, then a' distinguishes Q_1^1 and Q_1^2; if elements of Q_1^1 and Q_1^2 go to Q_1 and Q_2, respectively, then $a'a$ distinguishes Q_1^1 and Q_1^2. The current partition is $\{F, Q_1^1, Q_1^2, Q_2\}$. Repeating this argument until the last division when we obtain the partition E_Q, we have for every two different cells M, N of E_Q, there exist $w \in \Sigma^*$ such that with reading w, elements of M and N go to the one of F and $Q \backslash F$ and the other of them. Hence w distinguishes M and N in DFA E_A.

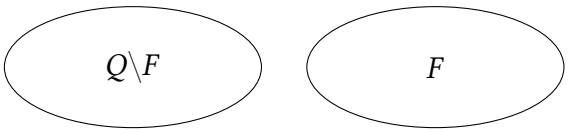


Figure 2.9: The first division in the proof of Theorem 2.5.

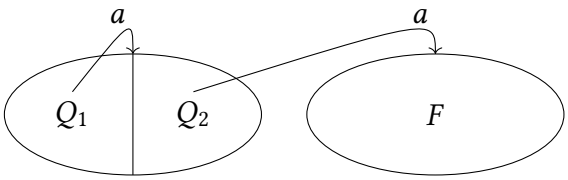


Figure 2.10: The second division in the proof of Theorem 2.5.

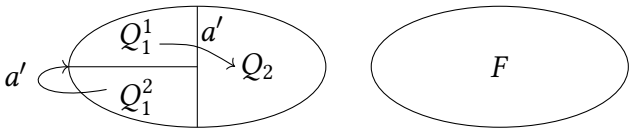


Figure 2.11: The third division in the proof of Theorem 2.5.

We second show that relation $\sim_L$ refines relation $\sim_{E_A}$. For every $u, v \in \Sigma^*$ that do not have the relation $\sim_{E_A}$. That is, $\delta'(q_0', u) =: M \neq \delta'(q_0', v) =: N$. By the above claim, there exists $w \in \Sigma^*$ such that $\delta'(M, w) \in F'$ and $\delta'(N, w) \notin F'$, or, $\delta'(M, w) \notin F'$ and $\delta'(N, w) \in F'$. That is, u and v do not have relation $\sim_L$. Hence $\sim_L$ refines $\sim_{E_A}$. Also because $\sim_{E_A}$ refines $\sim_L$, we have $\sim_{E_A} = \sim_L$. That is, E_A is the minimum-state DFA recognizing L. By the uniqueness of minimum-state DFA (Theorem 2.4), Procedure 1 returns the unique minimum-state DFA up to renaming states. □

Example 10: Reconsider language L_4 in Example 3 over alphabet $\{a, b\}$ consisting of all words containing aa as a subword. We had constructed an NFA A_4' in Figure 2.5 recognizing L_4. By the proof of Theorem 2.1, we had computed a DFA A_{4D}' shown in Figure 2.7 that is equivalent to A_4'. Finally, we had used Procedure 1 to convert A_4' to the minimum-state DFA A_{4D}' recognizing L_4 shown in Figure 2.8. □

2.6 Nondeterministic finite automata with ε-transitions

Previously, we showed that in order to construct a finite automaton to recognize a given regular language, it is easier to construct an NFA than to construct a DFA. In this subsection, we show that it is even easier to construct an NFA with *spontaneous* transitions, where a spontaneous transition is a transition between two states without reading any letter. In this case, a new symbol ε is added to such transitions as their label, and then spontaneous transitions are also called *ε-transitions*. Note that we adopt the symbol ε instead of the empty word ϵ for ease of subsequent representation. Following the previous notation, one has $\varepsilon^0 = \epsilon$. Analogously, NFAs with ε-transitions are called ε-NFAs. We will prove that although ε-NFAs are more general than NFAs, they still recognize exactly the regular languages.

Definition 6: An *ε-nondeterministic finite automaton* is a 5-tuple $(Q, \Sigma_\varepsilon, \delta, q_0, F)$, where

1. Q is a finite set of *states*,
2. Σ is an *alphabet*, $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, $\varepsilon \notin \Sigma$,
3. $\delta : Q \times \Sigma_\varepsilon \to 2^Q$ is the *transition function*,
4. $q_0 \in Q$ is the *initial state*, and
5. $F \subset Q$ is the set of *final states*.

The same as an NFA, there exists a transition $q \xrightarrow{a} q'$ in an ε-NFA A if and only if $q' \in \delta(q, a)$. Additionally, there exists a spontaneous transition $q \xrightarrow{\varepsilon} q'$ in A if and only if $q' \in \delta(q, \varepsilon)$. In order to define what words are accepted by A, we adopt two steps. First, we extend $\delta : Q \times \Sigma_\varepsilon \to 2^Q$ to $\delta : Q \times (\Sigma_\varepsilon)^* \to 2^Q$ in the usual recursive way: for all $q \in Q$, $u \in (\Sigma_\varepsilon)^*$, and $a \in \Sigma_\varepsilon$, one has $\delta(q, \epsilon) = \{q\}$, $\delta(q, ua) = \bigcup_{p \in \delta(q,u)} \delta(p, a)$. Second, we use the extended δ to define a new transition function $\hat{\delta} : Q \times \Sigma^* \to 2^Q$. To this end, we need a notion of *ε-closure*. For a subset $x \subset Q$, we define its *ε-closure* by

$$\varepsilon\text{-CLO}(x) = \bigcup_{n=0}^{|Q|-1} \bigcup_{q \in x} \delta(q, \varepsilon^n), \tag{2.2}$$

i.e., the set of all states of x and all states of A reachable from states of x without reading any letter. By using $\delta : Q \times (\Sigma_\varepsilon)^* \to 2^Q$ and ε-closure, we define

$$\hat{\delta} : Q \times \Sigma^* \to 2^Q \tag{2.3}$$

as follows: for all $q \in Q$, $u \in \Sigma^*$, and $a \in \Sigma$, one has $\hat{\delta}(q, \epsilon) = \varepsilon\text{-CLO}(\{q\})$, $\hat{\delta}(q, ua) = \bigcup_{p \in \hat{\delta}(q,u)} \varepsilon\text{-CLO}\left(\delta(p, a)\right)$. Here, $\hat{\delta}(q, \epsilon)$ is the set of q and all states reachable from q through only ε-transitions, which sometimes differs from $\delta(q, \epsilon)$ that consists of only q. For a word $w_1 \dots w_n \in \Sigma^+$, where $w_1, \dots, w_n \in \Sigma$, $\hat{\delta}(q, w)$ denotes the set of states reachable from q by

reading letters $w_1, \ldots, w_n$ successively, and before w_1, between w_i and w_{i+1}, $1 \leqslant i \leqslant n-1$, and after w_n, any number of possible ε-transitions are allowed to be passed.

Definition 7: Let Σ be an alphabet and $A = (Q, \Sigma_\varepsilon, \delta, q_0, F)$ be an ε-NFA. A word $w \in \Sigma^*$ is called *accepted by* A if $\hat{\delta}(q_0, w) \cap F \neq \varnothing$. The formal language *recognized by* A is the set of all words accepted by A, i.e., $\{w \in \Sigma^* | \hat{\delta}(q_0, w) \in F\}$.

By Definition 7, the empty word ϵ is accepted by A if and only if $q_0 \in F$ or some final state is reachable from q_0 through only ε-transitions. For a word $w = w_1 \ldots w_n \in \Sigma^+$, where $w_1, \ldots, w_n \in \Sigma$, w is accepted if and only if there exist states $q'_0, q_1, q'_1, \ldots, q_n, q'_n$ such that $q_i \in \delta(q'_{i-1}, w_i)$, $1 \leqslant i \leqslant n$, q'_j is the same as q_j or q'_j is reachable from q_j through only ε-transitions, $1 \leqslant j \leqslant n$, and $q'_n \in F$.

Example 11: Consider an ε-NFA A_5 shown in Figure 2.12.

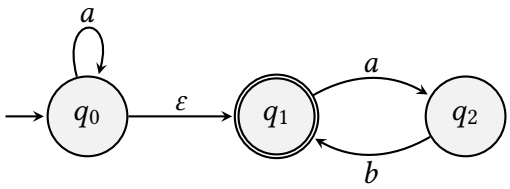


Figure 2.12: An ε-NFA A_5.

By definition, A_5 accepts exactly the words consisting of any number of a's followed by any number of repetitions of ab's. Its transition function δ is as follows:

$$\begin{aligned}
&\delta(q_0, \varepsilon) = \{q_1\}, && \delta(q_0, a) = \{q_0\}, && \delta(q_0, b) = \varnothing,\\
&\delta(q_1, \varepsilon) = \varnothing, && \delta(q_1, a) = \{q_2\}, && \delta(q_1, b) = \varnothing,\\
&\delta(q_2, \varepsilon) = \varnothing, && \delta(q_2, a) = \varnothing, && \delta(q_2, b) = \{q_1\}.
\end{aligned}$$

One can be compute as follows:

$$\begin{aligned}
\hat{\delta}(q_0, \epsilon) &= \varepsilon\text{-CLO}(\{q_0\}) = \{q_0, q_1\},\\
\hat{\delta}(q_0, a) &= \varepsilon\text{-CLO}\left(\delta(q_0, a) \cup \delta(q_1, a)\right) = \varepsilon\text{-CLO}(\{q_0, q_2\}) = \{q_0, q_1, q_2\},\\
\hat{\delta}(q_0, ab) &= \varepsilon\text{-CLO}(\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)) = \varepsilon\text{-CLO}(\{q_1\}) = \{q_1\},\\
\hat{\delta}(q_0, aba) &= \varepsilon\text{-CLO}(\delta(q_1, a)) = \varepsilon\text{-CLO}(\{q_2\}) = \{q_2\}.
\end{aligned}$$

$\hat{\delta}(q_0, ab) \cap F = \{q_1\} \neq \varnothing$, so A_5 accepts ab. On the other hand, $\hat{\delta}(q_0, aba) \cap F = \varnothing$, so A_5 does not accept aba.

It is not so easy to directly construct an NFA recognizing the same language. □

Equivalence of ε-NFAs and DFAs

Next we prove that the same as NFAs, every ε-NFA also has an equivalent DFA. Hence ε-NFAs also recognize exactly the regular languages. The process of constructing an equivalent DFA for an ε-NFA is similar to but a little more involved than that of constructing an equivalent DFA for an NFA as in the proof of Theorem 2.1.

Theorem 2.6: *Every ε-nondeterministic finite automaton has an equivalent deterministic finite automaton.*

Proof. Consider an ε-NFA $A_N^\varepsilon = (Q, \Sigma_\varepsilon, \delta, q_0, F)$ that recognizes a given language L. We next construct a DFA A_D^ε from A_N^ε that also recognizes L. The idea of constructing A_D^ε is keeping track of every word, meanwhile, collecting all states the word can lead to, by additionally considering all possible ε-transitions.

Formally, $A_D^\varepsilon = (Q', \Sigma, \delta', q_0', F')$, where

1. $Q' = 2^Q$,
2. for all $x \in Q'$ and $a \in \Sigma$, $\delta'(x, a) = \varepsilon\text{-CLO}\left(\bigcup_{q \in x} \delta(q, a)\right)$,
3. $q_0' = \varepsilon\text{-CLO}(\{q_0\})$,
4. $F' = \{x \in Q' | x \cap F \neq \varnothing\}$.

By the construction, one directly sees for every word $w \in \Sigma^*$, $\hat{\delta}(q_0, w) \cap F \neq \varnothing$ if and only if $\delta'(q_0', w) \in F'$, where $\hat{\delta}$ is as in (2.3). Hence A_N^ε and A_D^ε recognize the same language. □

Example 12: Reconsider the ε-NFA A_5 in Figure 2.12. We construct its equivalent DFA A_{5D} as in Figure 2.13 by the proof of Theorem 2.6.

Next we construct the minimum-state DFA equivalent to A_{5D}. First, we remove all unreachable states of A_{5D}, and then obtain a DFA A'_{5D} in Figure 2.14. Second, we use A'_{5D} to construct the minimum-state equivalent DFA. According to Procedure 1, we do this in the following steps:

1. Partition the state set Q of A'_{5D} as $\{Q_1, Q_2\}$, where $Q_1 = \{\{q_0, q_1\}, \{q_0, q_1, q_2\}, \{q_1\}\}$ consists of all final states, $Q_2 = \{\{q_2\}, \varnothing\}$ consists of all non-final states.
2. Choose Q_2, with reading b, $\{q_2\}$ goes to Q_1 but $\varnothing$ remains in Q_2, so partition Q_2 to $Q_2^1 = \{\{q_2\}\}$ and $Q_2^2 = \{\varnothing\}$. The current partition of Q is $\{Q_1, Q_2^1, Q_2^2\}$.
3. Choose Q_1, with reading a, $\{q_1\}$ goes to Q_2^1, but $\{q_0, q_1\}$ and $\{q_0, q_1, q_2\}$ remain in Q_1, so partition Q_1 to $Q_1^1 = \{\{q_0, q_1\}, \{q_0, q_1, q_2\}\}$ and $Q_1^2 = \{\{q_1\}\}$. The current partition of Q is $\{Q_1^1, Q_1^2, Q_2^1, Q_2^2\}$.

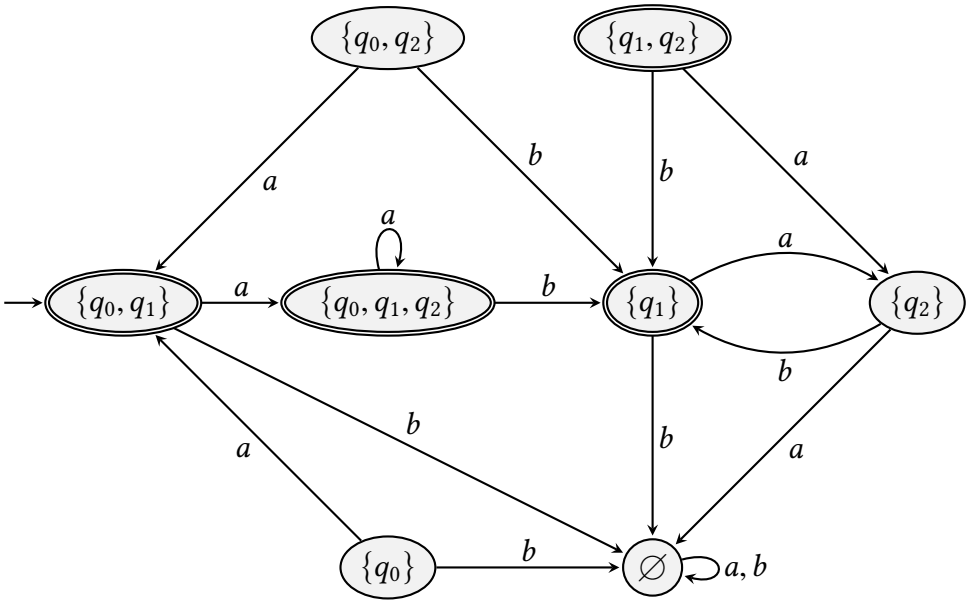


Figure 2.13: An equivalent DFA A_{5D} of the ε-NFA A_5 in Figure 2.12.

4. Choose Q_1^1. With reading b, $\{q_0, q_1\}$ goes to Q_2^2 but $\{q_0, q_1, q_2\}$ goes to Q_1^2, so partition Q_1^1 to $\{\{q_0, q_1\}\}$ and $\{\{q_0, q_1, q_2\}\}$.

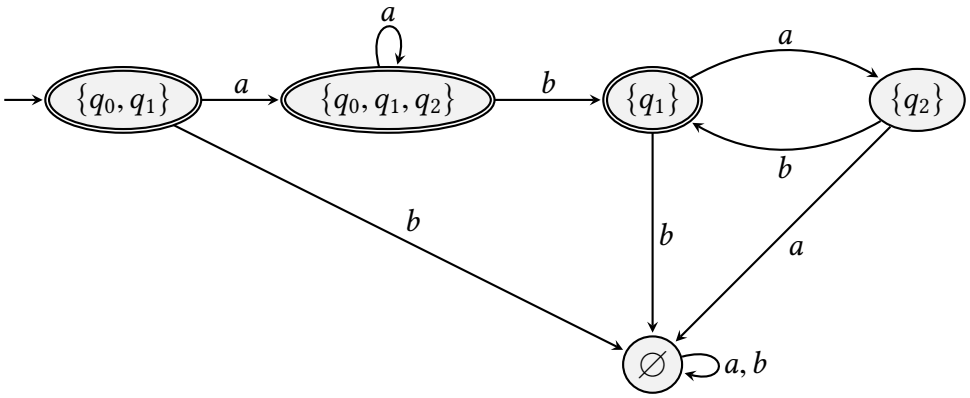


Figure 2.14: The minimum-state equivalent DFA A'_{5D} of the ε-NFA A_5 in Figure 2.12.

Hence A'_{5D} is the minimum-state equivalent DFA. Now look at A'_{5D}. Before the first b was read, any number of a's might have been read, so the words of any number of a's are accepted. Before the first ab was read, any number of a's might have been read; after that, any number of ab's could be read. So, A'_{5D} accepts exactly all words of any number of a's followed by any number of repetitions of ab's. □

Equivalence of ε-NFAs and NFAs

By Theorem 2.6, we know that ε-NFAs are equivalent to DFAs, hence they are also equivalent to NFAs (by Theorem 2.1). Next we show that given an ε-NFA A_N^ε, one can directly construct an equivalent NFA A_N. The construction is easier than constructing an equivalent DFA A_D, but the proof of the equivalence of A_N^ε and A_N is more complicated than proving the equivalence of A_N^ε and A_D.

Theorem 2.7: *Every ε-nondeterministic finite automaton has an equivalent nondeterministic finite automaton.*

Proof. Consider an ε-NFA $A_N^\varepsilon = (Q, \Sigma_\varepsilon, \delta, q_0, F)$. We construct an NFA $A_N = (Q, \Sigma, \delta', q_0, F')$ equivalent to A_N^ε. For all $q, q' \in Q$, and $a \in \Sigma$, $q' \in \delta'(q,a)$ if and only if in A_N^ε, there exist states $q'', q''' \in Q$ such that (1) $q''' \in \delta(q'', a)$, (2) either q'' is equal to q or q'' is reachable from q through only ε-transitions, (3) either q' is equal to q''' or q' is reachable from q''' through only ε-transitions. Extend $\delta' : Q \times \Sigma \to 2^Q$ recursively to $\delta' : Q \times \Sigma^* \to 2^Q$ as before. We claim that for all $q \in Q$ and $w \in \Sigma^+$, $\delta'(q,w) = \hat{\delta}(q,w)$, where $\hat{\delta}$ is as in (2.3). Note that $\delta'(q,\epsilon)$ and $\hat{\delta}(q,\epsilon)$ may not be the equal, because $\delta'(q,\epsilon) = \{q\}$, but $\hat{\delta}(q,\epsilon)$ may also contain other states reachable from q through only ε-transitions. We prove the claim by mathematical induction.

1. Assume $|w| = 1$. One has

$$\begin{aligned}\hat{\delta}(q,w) &= \bigcup_{p \in \varepsilon\text{-CLO}(\{q\})} \varepsilon\text{-CLO}(\delta(p,w)) \\ &= \delta'(q,w)\end{aligned}$$

   by definition.

2. Assume that the claim has been proved for all $(n-1)$-length words. We next consider $w = ua \in \Sigma^+$, where $|u| = n-1 > 0$, $a \in \Sigma$. Then

$$\begin{aligned}\delta'(q,ua) &= \bigcup_{p \in \delta'(q,u)} \delta'(p,a) \\ &= \bigcup_{p \in \hat{\delta}(q,u)} \hat{\delta}(p,a) \\ &= \bigcup_{p \in \hat{\delta}(q,u)} \bigcup_{r \in \varepsilon\text{-CLO}(\{p\})} \varepsilon\text{-CLO}(\delta(r,a)) \\ &= \bigcup_{p \in \hat{\delta}(q,u)} \varepsilon\text{-CLO}(\delta(p,a)) \left(\text{because } \bigcup_{p \in \hat{\delta}(q,u)} \varepsilon\text{-CLO}(\{p\}) = \hat{\delta}(q,u) \right) \\ &= \hat{\delta}(q,ua).\end{aligned}$$

Here $\bigcup_{p \in \hat{\delta}(q,u)} \varepsilon\text{-CLO}(\{p\}) = \hat{\delta}(q,u)$ because $\varepsilon\text{-CLO}(\{p\}) \subset \hat{\delta}(q,u)$ for all $p \in \hat{\delta}(q,u)$ (see Figure 2.15).

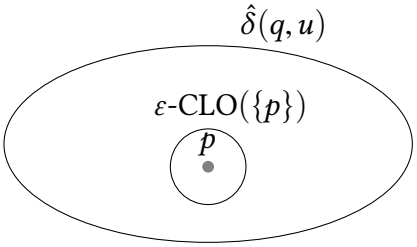


Figure 2.15: Illustration in the proof of Theorem 2.7 (by $q \xrightarrow{\varepsilon^*} \xrightarrow{u} \xrightarrow{\varepsilon^*} p \xrightarrow{\varepsilon^*} r$).

The rest is to specify F'. We divide it into two cases: (i) ϵ is not accepted by A_N^ε and (ii) ϵ is accepted by A_N^ε.

Case (i): Because ϵ is not accepted by A_N^ε, ε-CLO$(\{q_0\}) \cap F = \varnothing$. Choose $F' = F$. Then q_0 is not a final state of A_N, so ϵ is not accepted by A_N either. For all $w \in \Sigma^+$, $\hat{\delta}(q_0, w) \cap F \neq \varnothing \iff \delta'(q_0, w) \cap F' \neq \varnothing$ (by the previous claim), then w is accepted by A_N^ε if and only if it is accepted by A_N.

Case (ii): Because ϵ is accepted by A_N^ε, ε-CLO$(\{q_0\}) \cap F \neq \varnothing$. Choose $q_0' \in \varepsilon$-CLO$(\{q_0\}) \cap F$ which is a final state of A_N^ε. Choose $F' = F \cup \{q_0\}$. Then ϵ is accepted by A_N. Let w be in Σ^+. We next check w is accepted by A_N^ε if and only if it is accepted by A_N. Assume w is accepted by A_N^ε, i.e., $\hat{\delta}(q_0, w) \cap F \neq \varnothing$. Then $\delta'(q_0, w) \cap F' \neq \varnothing$, w is also accepted by A_N. Assume w is accepted by A_N, then $\delta'(q_0, w) \cap F' \neq \varnothing$. If $\delta'(q_0, w) \cap F \neq \varnothing$ then $\hat{\delta}(q_0, w) \cap F \neq \varnothing$, and w is accepted by A_N^ε; otherwise one has $q_0 \in \delta'(q_0, w)$, then $q_0 \in \hat{\delta}(q_0, w)$ and $q_0' \in \hat{\delta}(q_0, w)$, w is also accepted by A_N^ε. □

Example 13: Reconsider the ε-NFA A_5 in Example 11.

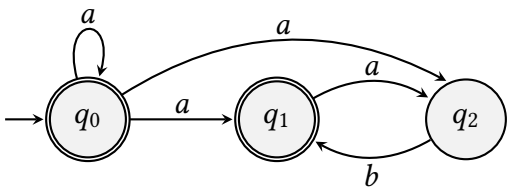


Figure 2.16: An NFA A_5' equivalent to the ε-NFA A_5 in Figure 2.12.

By the proof of Theorem 2.7, we construct an equivalent NFA A_5' as in Figure 2.16. The transitions $q_0 \xrightarrow{a} q_1$ and $q_0 \xrightarrow{a} q_2$ in A_5' come from the transition sequences $q_0 \xrightarrow{a} q_0 \xrightarrow{\varepsilon} q_1$ and $q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{a} q_2$ in A_5. Because ϵ is accepted by A_5 ($q_1 \in \varepsilon$-CLO$(\{q_0\})$ is a final state), q_0 is a final state of A_5'. □

2.7 Regular expressions and the Kleene theorem

In this subsection, we introduce *regular expressions* which provide another definition for a regular language that looks completely different from the definition based on finite automata. Regular expressions are recursively defined by combining atomic expressions based on three operations: *union*, *concatenation*, and *Kleene star*. Usually regular expressions do not provide concise representation for regular languages. However, just because of the redundancy, it is relatively easier to use regular expressions to represent complicated regular languages, which is similar to the case of using ε-NFAs.

Union and concatenation are binary operations, Kleene star is a unary operation. Let Σ be an alphabet and $L_1, L_2, L \subset \Sigma^*$ be formal languages. The *union* $L_1 + L_2$ of L_1 and L_2 is defined by $\{w \in \Sigma^* | w \in L_1 \text{ or } L_2\}$. The *concatenation* $L_1 \circ L_2$ (in most cases "$\circ$" was omitted) of L_1 and L_2 is defined by $\{uv | u \in L_1, v \in L_2\}$, the set of all concatenations of a word of L_1 and a word of L_2. The *Kleene star* L^* of L is defined by $\{w_1 \dots w_n | n \geqslant 0, w_i \in L, 1 \leqslant i \leqslant n\} = \bigcup_{n=0}^{\infty} L^n$, the set of all concatenations of any number of words of L, where L^i denotes the set of all concatenations of any i words of L. $L^+ = \bigcup_{n=1}^{\infty} L^n$. Particularly, $L^0 = \{\epsilon\}$, $(L)^1 = L$, $L^* = L^+ \cup \{\epsilon\}$, $L^+ = LL^*$, $L^* = L^+$ if and only if $\epsilon \in L$, $\varnothing^* = \{\epsilon\}$, $\varnothing^+ = \varnothing$.

For example,

$$\begin{aligned}
\{a, b\}\{ab, ba\} &= \{aab, aba, bab, bba\},\\
\{a, ab\}^0 &= \{\epsilon\},\\
\{a, ab\}^1 &= \{a, ab\},\\
\{a, ab\}^2 &= \{aa, aab, aba, abab\},\\
\{a, ab\}^* &= \{\epsilon, a, ab, aa, aab, aba, abab, aaa, aaab, aaba, aabab, \dots\}.
\end{aligned}$$

Definition 8: Let Σ be an alphabet. r is called a *regular expression* (over Σ) if r is

1. $\varnothing$,
2. ϵ,
3. a, where $a \in \Sigma$,
4. $r_1 + r_2$, where r_1 and r_2 are regular expressions,
5. $r_1 r_2$, where r_1 and r_2 are regular expressions,
6. $(r_1)^*$, where r_1 is a regular expression.

Here $\varnothing$ represents language $\varnothing$, ϵ represents language $\{\epsilon\}$, a represents language $\{a\}$, $r_1 + r_2$ (resp., $r_1 r_2$) means the union (resp., concatenation) of the languages represented by r_1 and r_2, and $(r_1)^*$ means the Kleene star of the language represented by r_1.

The above definition is in a recursive form. For a regular expression r, we also use r to denote the language represented by r. In addition, rr^* will be denoted by r^+. In addition, one has $r^0 = \epsilon$. Denote $\underbrace{r \dots r}_{n} =: r^n$ for all natural numbers n.

The three operations have the following precedence:

$$\text{Kleene star} > \text{concatenation} > \text{union},$$

which is similar to the precedence order of power, product, and addition in real numbers.

Proposition 2.8: *The three operations have the following properties: Let r, r_1, r_2, r_3 be arbitrary regular expressions over a given alphabet, (1) union has commutativity: $r_1 + r_2 = r_2 + r_1$; (2) union and concatenation both have associativity, i.e., $(r_1 + r_2) + r_3 = r_1 + (r_2 + r_3) =: r_1 + r_2 + r_3$, $(r_1 r_2) r_3 = r_1 (r_2 r_3) =: r_1 r_2 r_3$; (3) concatenation distributes over union on both sides: $r_1(r_2 + r_3) = r_1 r_2 + r_1 r_3$, $(r_1 + r_2) r_3 = r_1 r_3 + r_2 r_3$; (4) $\varnothing + r = r + \varnothing = r$, $\epsilon r = r\epsilon = r$; (5) $\varnothing$ is absorbing: $\varnothing r = r\varnothing = \varnothing$; (6) union is idempotent: $r + r = r$.*

Hence the structure $(R, +, \circ, \varnothing, \epsilon)$ forms an *idempotent semiring* (see Section 4.2), where R denotes the set of all regular expressions over a given alphabet.

For example,

$$(((ab)^*) + a(ba)b)((a^*)b) = ((ab)^* + abab)a^*b = (ab)^*a^*b + ababa^*b.$$

Example 14: Construct regular expressions for the following languages over the alphabet $\{a, b\}$:

1. $\{a, ab, abb\}$.
2. All words of Σ.
3. All words containing aa as a subword.
4. All words ending with aa.
5. All words beginning with aa and ending with bb.
6. All words containing two b's separated by an even number of a's.
7. All words that are concatenations of any number of a's followed by concatenations of any number of repetitions of ab's followed by concatenations of any number of b's.

The answer is as follows:

1. $a + ab + abb$.
2. $(a + b)^*$.
3. $(a + b)^* aa (a + b)^*$.

4. $(a+b)^*aa$.
5. $aa(a+b)^*bb$.
6. $(a+b)^*b(aa)^*b(a+b)^*$.
7. $a^*(ab)^*b^*$.

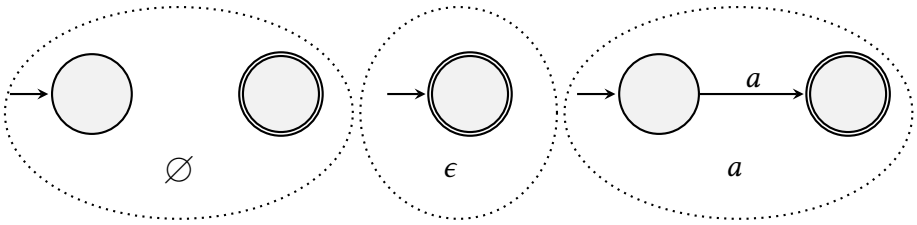
Actually, the above regular expressions were obtained by just writing down atomic expressions along with reading the English expressions of above languages. Is this procedure similar to constructing an ε-NFA to recognize a given regular language? □

Next we show the Kleene theorem: regular expressions exactly represent the regular languages.

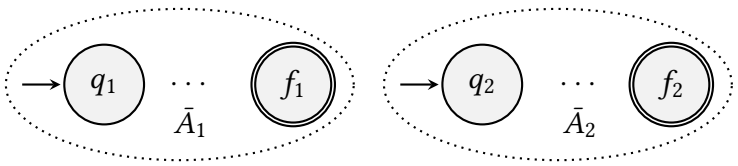
Theorem 2.9 (Kleene)**:** *Let Σ be an alphabet. A language L over Σ is regular if and only if there exists a regular expression r over Σ representing it.*

Proof. Every regular language can be recognized by an ε-NFA with a single final state. To see this, for a given ε-NFA A, if A contains no final states, then we add a final state to A, thus we obtain a new ε-NFA A'; if A contains more than one final state, then we add a new state $\diamond$ into A, and then for every final state q_f of A, we add an ε-transition from q_f to $\diamond$, set q_f to be not final any more, and set $\diamond$ to be final, thus we also obtain a new ε-NFA A'. In both cases, A' accepts the same language as A accepts. So, next we only consider ε-NFAs with a single final state without loss of generality.

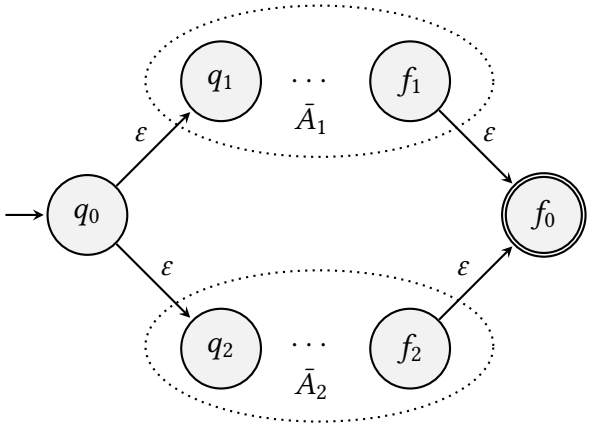
"only if": We need to prove for every regular expression r over Σ, there exists an ε-NFA recognizing r. We prove this recursively just as defining regular expressions. The ε-NFAs recognizing $\varnothing$, ϵ, and a are shown as follows:



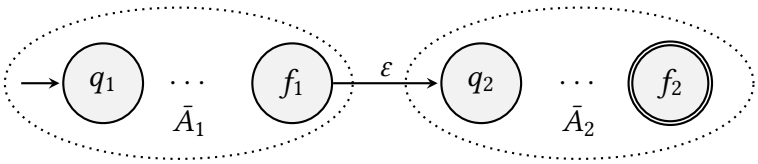
Consider two regular expressions r_1 and r_2 over Σ. Assume the following $\bar{A}_1$ and $\bar{A}_2$ to be two ε-NFAs recognizing them, respectively, where $\bar{A}_i$ has a single final state f_i, the initial state of $\bar{A}_i$ is q_i, $i = 1, 2$.



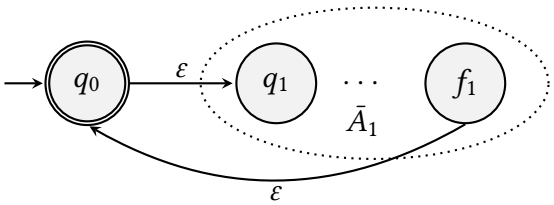
Next we construct ε-NFAs from $\bar{A}_1$ and $\bar{A}_2$ recognizing regular expressions $r_1 + r_2$, $r_1 r_2$, and $(r_1)^*$, respectively. An ε-NFA $\bar{A}_3$ recognizing $r_1 + r_2$ is shown as follows:



An ε-NFA $\bar{A}_4$ recognizing $r_1 r_2$ is shown as follows:



An ε-NFA $\bar{A}_5$ recognizing $(r_1)^*$ is shown as follows:



Until now, we have constructed three ε-NFAs $\bar{A}_3$, $\bar{A}_4$, $\bar{A}_5$ recognizing $r_1 + r_2$, $r_1 r_2$, and $(r_1)^*$, respectively, and all of them have a single final state. Hence we can recursively construct ε-NFAs with a single final state to recognize every regular expression. This completes the "only if" part.

"if": We next design Procedure 2 to convert an ε-NFA A_N^ε to a regular expression r such that A_N^ε recognizes r. The overall idea is that, first, add two new states s and a into A_N^ε to make A_N^ε have a single final state a and have s as the initial state but the recognized language does not change; second, repetitively choose a state q that is not s or a, remove q but meanwhile merging every pair of transitions going in and leaving q and adding new regular expressions into the labels of the new transitions so that the recognized language still does not change, until only s and a are left. Then the regular expression from s to a is recognized by A_N^ε. Note

that in the process, the labels of transitions may not be elements of Σ_ε but regular expressions over Σ. In this sense, the automata generated in Procedure 2 are called *generalized ε-NFAs*, and the language recognized by a generalized ε-NFA is the union of all regular expressions accepted by the generalized ε-NFA.

Procedure 2: Input an ε-NFA $A_N^\varepsilon = (Q, \Sigma_\varepsilon, \delta, q_0, F)$. Denote by L the language recognized by A_N^ε.

(1) For every transition $q \xrightarrow{\varepsilon} q'$, change ε to ϵ. Add two new states s and a into A_N^ε. Add a transition $s \xrightarrow{\epsilon} q_0$, set s to be initial and q_0 not initial. For every final state $q_f \in F$, add a transition $q_f \xrightarrow{\epsilon} a$, set all states of F not to be final any more, and set a to be the unique final state. For every two state $q, q' \in Q \cup \{s, a\}$, if there exists a transition from q to q', we set $r_{qq'}$ to be the label of the transition, otherwise we set $r_{qq'} = \varnothing$. (Now $r_{qq'}$ are always regular expressions for all states q, q'. A generalized ε-NFA has been obtained.)

(2) Choose a state q differs from s and a. For every two states q_1, q_2 different from q such that $q_1 \neq a$ and $q_2 \neq s$, $r_{q_1q_2} := r_{q_1q_2} + r_{q_1q}(r_{qq})^* r_{qq_2}$. Delete q.

(3) Repeat (2) until only s and a are left. Return r_{sa}.

Apparently, after step (1), the obtained generalized ε-NFA recognizes language L. We next prove that during the running of Procedure 2, in all subsequent steps, the obtained generalized ε-NFAs always recognize L by mathematical induction. Hence the final regular expression r_{sa} is exactly L.

Assume an intermediate generalized ε-NFA $\bar{A}_N^\varepsilon$ recognizes L. Denote by $\tilde{A}_N^\varepsilon$ the generalized ε-NFA that is obtained by executing step (2) once on $\bar{A}_N^\varepsilon$ and deleting one state denoted by $\bar{q}$. Hence $\tilde{A}_N^\varepsilon$ has one state fewer than $\bar{A}_N^\varepsilon$. For two states q, q' of $\bar{A}_N^\varepsilon$ (resp., $\tilde{A}_N^\varepsilon$), we use $\bar{r}_{qq'}$ (resp., $\tilde{r}_{qq'}$) to denote the label of the transition from q to q' in $\bar{A}_N^\varepsilon$ (resp., $\tilde{A}_N^\varepsilon$), and let $\bar{r}_{qq'} = \varnothing$ (resp., $\tilde{r}_{qq'} = \varnothing$) if there is no such a transition.

Consider a transition sequence

$$s \xrightarrow{w_1} p_1 \xrightarrow{w_2} \cdots \xrightarrow{w_{n-1}} p_{n-1} \xrightarrow{w_n} a \tag{2.4}$$

of $\bar{A}_N^\varepsilon$, where $s =: p_0$, $a =: p_n$, $w_i = \bar{r}_{p_{i-1}p_i}$, $1 \leqslant i \leqslant n$. That is, $w_1 \ldots w_n$ is accepted by $\bar{A}_N^\varepsilon$. Choose an arbitrary $0 \leqslant j \leqslant n$ such that $p_j \neq \bar{q}$. Choose $p_j \xrightarrow{w_{j+1}} \cdots \xrightarrow{w_k} p_k \xrightarrow{w_{k+1}} p_{k+1}$ such that $p_{j+1} = \cdots = p_k = \bar{q}$ but $p_{k+1} \neq \bar{q}$. Such a p_{k+1} must exist because $a \neq \bar{q}$. If $j = k$, then $p_j, p_{k+1} \neq \bar{q}$, by $\tilde{r}_{p_jp_{k+1}} = w_{j+1} + \bar{r}_{p_j\bar{q}}(\bar{r}_{\bar{q}\bar{q}})^* \bar{r}_{\bar{q}q_{k+1}}$, one has $w_{j+1} \subset \tilde{r}_{p_jp_{k+1}}$; otherwise if $j < k$, then $w_{j+2} = \cdots = w_k = \bar{r}_{\bar{q}\bar{q}}$, by $\tilde{r}_{p_jp_{k+1}} = \bar{r}_{p_jp_{k+1}} + \bar{r}_{p_j\bar{q}}(\bar{r}_{\bar{q}\bar{q}})^* \bar{r}_{\bar{q}q_{k+1}}$, one has $w_{j+1} \ldots w_{k+1} \subset \bar{r}_{p_j\bar{q}}(\bar{r}_{\bar{q}\bar{q}})^* \bar{r}_{\bar{q}q_{k+1}} \subset \tilde{r}_{p_jp_{k+1}}$. Hence $w_1 \ldots w_n$ is also accepted by $\tilde{A}_N^\varepsilon$.

Consider a transition sequence

$$s \xrightarrow{u_1} p_1 \xrightarrow{u_2} \cdots \xrightarrow{u_{n-1}} p_{n-1} \xrightarrow{u_n} a \tag{2.5}$$

of $\tilde{A}^{\varepsilon}_N$, where $s =: p_0$, $a =: p_n$, $u_i = \tilde{r}_{p_{i-1}p_i}$, $1 \leqslant i \leqslant n$. That is, $u_1 \dots u_n$ is accepted by $\tilde{A}^{\varepsilon}_N$. Here none of p_i's is equal to $\bar{q}$, because $\bar{q}$ is not a state of $\tilde{A}^{\varepsilon}_N$. For all $1 \leqslant i \leqslant n$, $u_i = \bar{r}_{p_{i-1}p_i} + \bar{r}_{p_{i-1}\bar{q}}(\bar{r}_{\bar{q}\bar{q}})^*\bar{r}_{\bar{q}q_i}$, hence $u_1 \dots u_n$ is also accepted by $\bar{A}^{\varepsilon}_N$. This completes the "if" part. □

Example 15: Reconsider the DFA A_4 in Figure 2.4 recognizing the language L_4 in Example 3 over alphabet $\{a, b\}$ consisting of all words containing aa as a subword. The regular expression for L_4 is $(a + b)^*aa(a + b)^*$ (see Example 14).

Next, we use Procedure 2 to convert A_4 to an equivalent regular expression, and see whether it is the same as $(a+b)^*aa(a+b)^*$. The process is shown in Figure 2.17, and the obtained regular expression is $b^*a(b^+a)^*a(a+b)^*$. By the proof of the "if" part of Theorem 2.9, one has $b^*a(b^+a)^*a(a + b)^* = (a + b)^*aa(a + b)^*$. However, they look quite different, so we still want to check whether they are equal. It is not easy to directly do the check, but we can construct the minimum-state DFA recognizing $b^*a(b^+a)^*a(a+b)^*$, and see whether this DFA is the same as A_4.

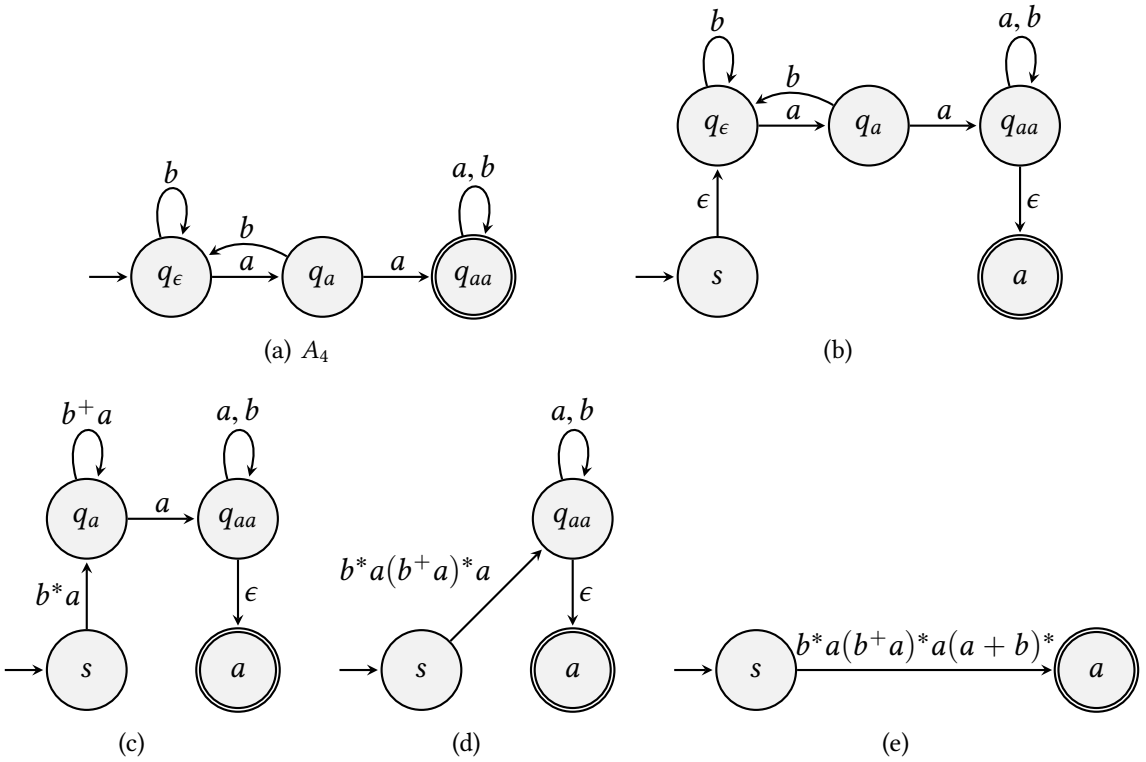
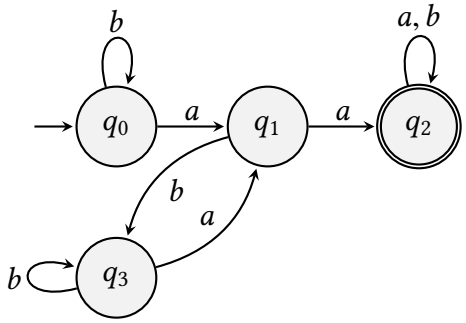
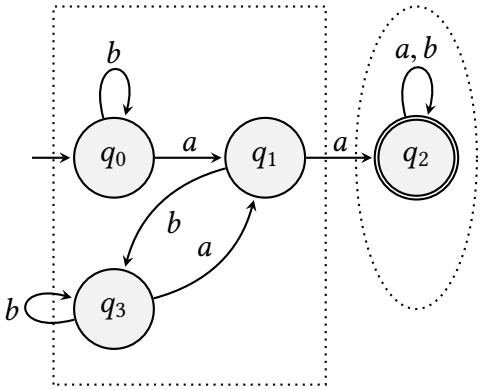


Figure 2.17: The process of using Procedure 2 to convert the DFA A_4 in Figure 2.4 to an equivalent regular expression.

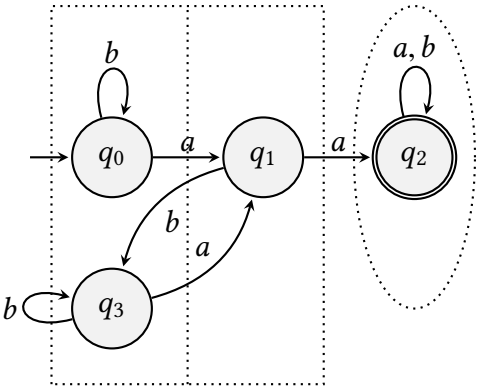
We first construct an ε-NFA recognizing $b^*a(b^+a)^*a(a + b)^*$ which is shown in Figure 2.18(a), which is coincidentally a DFA, second we use Procedure 1 to convert the obtained DFA to the minimum-state DFA (in Figure 2.18(d)) recognizing $b^*a(b^+a)^*a(a + b)^*$, where the process is shown in Figure 2.18 and the minimum-state DFA is the same as A_4 up to



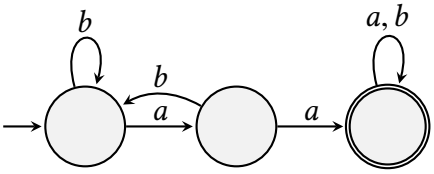
(a) An ε-NFA recognizing regular expression $b^*a(b^+a)^*a(a+b)^*$.



(b)



(c)



(d) The minimum-state DFA recognizing regular expression $b^*a(b^+a)^*a(a+b)^*$.

Figure 2.18: The process of using Procedure 1 to construct the minimum-state DFA recognizing regular expression $b^*a(b^+a)^*a(a+b)^*$.

renaming states. Hence $b^*a(b^+a)^*a(a+b)^*$ is indeed equal to $(a+b)^*aa(a+b)^*$. □

2.8 Closure properties

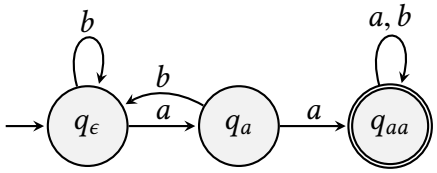
Let $L_1, L_2 \subset \Sigma^*$ be two regular languages over alphabet Σ. It follows from the proof of Theorem 2.9 that the union $L_1 \cup L_2$ of L_1 and L_2 is still regular. That is, let r_1 and r_2 be regular expressions for L_1 and L_2, respectively, then $r_1 + r_2$ is a regular expression for $L_1 \cup L_2$. In the same way, the concatenation L_1L_2 of L_1 and L_2 is also regular, because r_1r_2 is a regular expression for L_1L_2; similarly $(L_1)^*$ is also regular, because $(r_1)^*$ is a regular expression for $(L_1)^*$. To sum up, the family of regular languages is *closed* under union, concatenation, and Kleene star.

Union and concatenation are binary operations on the set of formal languages over a

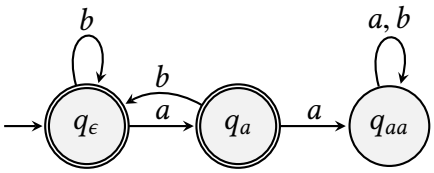
given alphabet. Kleene star is a unary operation on the set. We say a subfamily of formal languages is *closed under an operation* if when the operation acts on the subfamily, no language outside the subfamily is generated.

We next show the family of regular languages is closed under two other operations: intersection and complement. The intersection of L_1 and L_2 is $L_1 \cap L_2$, the complement of L_1 is $\Sigma^* \backslash L_1 =: \overline{L_1}$.

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing L_1. Construct $\overline{A} = (Q, \Sigma, \delta, q_0, Q \backslash F)$. A word $w \in \Sigma^*$ is accepted by A if and only if it is not accepted by $\overline{A}$. Hence $\overline{A}$ recognizes language $\overline{L_1}$. That is, the family of regular languages is closed under complement. For example, the following DFA A_4 accepts exactly the words that are not accepted by $\overline{A_4}$.



(a) A_4 in Figure 2.4.



(b) $\overline{A_4}$.

By De Morgan's laws, one has $L_1 \cap L_2 = \overline{\overline{A_1} \cup \overline{A_2}}$, hence the family of regular languages is also closed under intersection.

From the above discussions, let A_1 and A_2 be two ε-NFAs recognizing L_1 and L_2, respectively, one can design mechanical procedures (similar to Procedure 1) to construct another ε-NFA in finite steps from A_1 and A_2 recognizing any of $L_1 \cap L_2$, $L_1 \cup L_2$, $\overline{L_1}$, $L_1 L_2$, and $(L_1)^*$. We say such constructions are *effective*. These results are then summarized as follows.

Theorem 2.10: *The family of regular languages is effectively closed under intersection, union, complement, concatenation, and Kleene star.*

Particularly, for DFA $A_i = (Q_i, \Sigma, \delta_i, q_0^i, F_i)$ recognizing L_i, $i = 1, 2$, one can effectively construct a product of A_1 and A_2 that recognizes $L_1 \cup L_2$ and also effectively construct another product of A_1 and A_2 that recognizes $L_1 \cap L_2$. In order to recognize $L_1 \cup L_2$, one constructs $A = (Q, \Sigma, \delta, q_0, F)$, where $Q = Q_1 \times Q_2$, $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ for all $(q_1, q_2) \in Q$ and $a \in \Sigma$, $q_0 = (q_0^1, q_0^2)$, $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. In order to recognizes $L_1 \cap L_2$, one just needs to change F to $F_1 \times F_2$.

2.9 Nonregular languages and the pumping lemma

To understand the power of finite automata, one must also understand their limitations. In this section, we show the *pumping lemma* for regular languages and use the lemma to

prove several languages cannot be recognized by any finite automaton.

Intuitively, the pumping lemma shows that in a regular language, in a word, some positive-length subword can be repeated arbitrarily often in the same place but the modified words still belong to the language. In this case, we say the word can be pumped, or pumped down (in case of removing the subword). This holds because a finite automaton has finitely many states, and sufficiently long words must lead an automaton to visit some state at least twice.

Theorem 2.11 (Pumping lemma)**:** *Let L be a regular language. There is a positive integer p (the* pumping length*) such that if u is a word in L of length at least p, then u can be written as $u = xyz$, where*

1. *for each $i \geqslant 0$, $xy^iz \in L$,*
2. *$|y| > 0$, and*
3. *$|xy| \leqslant p$.*

Proof. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing L and have exactly p states. Let $u \in L$ be of length at least p. By the *pigeonhole principle*, the prefix $u_1 \dots u_p$ of u leads A to be in a same state at least twice, where $u = u_1 \dots u_{|u|}$, $u_i \in \Sigma$, $1 \leqslant i \leqslant |u|$. In detail, there exist $1 \leqslant k < l \leqslant p$ such that $\delta(q_0, u_1 \dots u_k) = \delta(q_0, u_1 \dots u_l)$. Choose $x = u_1 \dots u_k$, $y = u_{k+1} \dots u_l$, $z = u_{l+1} \dots u_{|u|}$, then xyz satisfy conditions 1, 2, and 3. Note that both x and z could possibly be ϵ. □

One can use Theorem 2.11 to prove some languages are not regular: given a language L, if one can find a sufficiently long word (whose length must be no less than some pumping length of L) in L that does not satisfy all conditions 1, 2, and 3, then L is not regular. Sometimes it is not easy to find such a word (it may require a bit of creative thinking).

Example 16: Consider language $L_5 = \{a^n b^n | n \geqslant 0\}$ in Example 3. We use the pumping lemma to prove L_5 is not regular (by contradiction).

Assume to the contrary that L_5 is regular. By Theorem 2.11, choose a pumping length p of L_5 and $a^p b^p \in L_5$. Then $a^p b^p$ can be written as $a^p b^p = xyz$ such that (1) for all $i \geqslant 0$, $xy^iz \in L_5$, (2) $|y| > 0$, and (3) $|xy| \leqslant p$. Hence xy contains no b, and $x = a^s$, $y = a^t$ for some $s \geqslant 0$ and $t > 0$ such that $s + t \leqslant p$. For the final step, we can either pump $a^p b^p$ or pump down $a^p b^p$. In the former case we have $a^s a^{2t} a^{p-s-t} b^p = a^{p+t} b^p \in L_5$, but this is impossible. In the latter case we have $a^s a^{p-s-t} b^p = a^{p-t} b^p \in L_5$, this is also impossible. Hence in both cases, we conclude L_5 is not regular. □

Example 17: Consider language $L_6 = \{a^{n^2} | n \geqslant 0\}$. Assume L_6 is regular, then by Theorem 2.11, choose a pumping length p of L_6 and $a^{p^2} \in L_6$. Then a^{p^2} can be written as $a^{p^2} = xyz$ such that (1) for all $i \geqslant 0$, $xy^iz \in L_6$, (2) $|y| > 0$, and (3) $|xy| \leqslant p$. We pump a^{p^2} as $xy^2z \in L_6$,

which satisfies $p^2 < |xy^2z| \leqslant p^2 + p < (p+1)^2$, which is impossible. Hence L_6 is not regular. □

2.10 Labeled finite-state automata

In the control science field, a research direction called *discrete-event systems* (DESs) was initiated in [RW87] under the framework of *partially-observed* (aka *labeled*) *finite-state automata* (LFSAs). A DES usually consists of discrete states and transitions between states caused by spontaneous occurrences of events, where the events are partially observed (represented by a labeling function). A DES can be regarded as an abstraction of the logic layer of a *cyber-physical system* (CPS) ubiquitous in control engineering, computer technology, communication engineering, etc. Usually, a CPS consists of a cyber layer and a physical layer, where the former is a decision process that is usually a discrete system, and the latter usually comprises several physical processes modeled by differential equations. The two layers are connected by networks, where the cyber layer should be able to monitor the working status of the physical layer in real time, and also allocate commands to the physical layer, both through networks. In such a way, DESs play a central role in governing global behavior of CPSs.

An LFSA is obtained from an NFA (see Definition 3) by removing all accepting states, replacing a unique initial state by a set of initial states, and adding a labeling function.

Definition 9: A *labeled finite-state automaton* is a 6-tuple $\mathcal{S} = (Q, E, \delta, Q_0, \Sigma, \ell)$, where

1. Q is a finite set of *states*,
2. E is a finite set of *events*,
3. $\delta : Q \times E \to 2^Q$ is the *transition function* (equivalently represented by the *transition relation* $\delta \subset Q \times E \times Q$),
4. $Q_0 \subset Q$ is the set of *initial states*,
5. Σ is a finite set of *outputs*/*labels*, and
6. $\ell : E \to \Sigma \cup \{\epsilon\}$ is the *labeling function*.

The event set E can been rewritten as disjoint union of *observable* event set $E_o = \{e \in E | \ell(e) \in \Sigma\}$ and *unobservable* event set $E_{uo} = \{e \in E | \ell(e) = \epsilon\}$. When an observable event occurs, its label can be observed; when an unobservable event occurs, nothing can be observed. Transition function $\delta : Q \times E \to 2^Q$ is recursively extended to $\delta : Q \times E^* \to 2^Q$ as usual; equivalently, transition relation $\delta \subset Q \times E \times Q$ is recursively extended to $\delta \subset Q \times E^* \times Q$ as follows: (1) for all $q, q' \in Q$, $(q, \epsilon, q') \in \delta$ if and only if $q = q'$; (2) for all $q, q' \in Q$, $s \in E^*$, and $e \in E$, one has $(q, se, q') \in \delta$, also denoted by $q \xrightarrow{se} q'$, called a *transition*

sequence or *run*, if and only if $(q, s, q''), (q'', e, q') \in \delta$ for some $q'' \in Q$.

Labeling function $\ell : E \to \Sigma \cup \{\epsilon\}$ is recursively extended to $\ell : E^* \cup E^\omega \to \Sigma^* \cup \Sigma^\omega$ as $\ell(e_1 e_2 \dots) = \ell(e_1)\ell(e_2)\dots$ and $\ell(\epsilon) = \epsilon$, where E^ω (resp., Σ^ω) denotes the set of *configurations* over E (resp., Σ), where a configuration is an infinite-length sequence of elements of E (resp., Σ). Transitions $x \xrightarrow{e} x'$ with $\ell(e) = \epsilon$ (resp., $\ell(e) \neq \epsilon$) are called *unobservable* (resp., *observable*). For $q \in Q$ and $s \in E^+$, (q, s, q) is called a *transition cycle* if $(q, s, q) \in \delta$. An *observable transition cycle* is defined by a transition cycle with at least one observable transition. Analogously an *unobservable transition cycle* is defined by a transition cycle with no observable transition. An LFSA is called *deterministic* if $|Q_0| = 1$ and for all $q, q', q'' \in Q$ and $e \in E$, it holds that $(q, e, q'), (q, e, q'') \in \delta \implies q' = q''$.

A state $q \in Q$ is called *live* if $(q, e, q') \in \delta$ for some $e \in E$ and $q' \in Q$. $\mathcal{S}$ is called *live* if each of its reachable states is live. We say *a state $q' \in Q$ is reachable from a state $q \in Q$* if there exists $s \in E^+$ such that $q \xrightarrow{s} q'$. We say *a subset Q' of Q is reachable from a state $q \in Q$* if some state of Q' is reachable from q. Similarly *a state $q \in Q$ is reachable from a subset Q' of Q* if q is reachable from some state of Q'. We call a state $q \in Q$ *reachable* if either $q \in Q_0$ or it is reachable from some initial state. For a transition $(q, e, q') \in \delta$, we call a transition $(q'', e', q''') \in \delta$ a *predecessor* of (q, e, q') if either $q = q'''$ or q is reachable from q'''; call (q'', e', q''') a *successor* of (q, e, q') if either $q'' = q'$ or q'' is reachable from q'.

We use $L(\mathcal{S}) = \{s \in E^* | (\exists q_0 \in Q_0)(\exists q \in Q)[q_0 \xrightarrow{s} q]\}$ to denote the set of finite-length event sequences generated by $\mathcal{S}$, we also use $L^\omega(\mathcal{S}) = \{e_1 e_2 \dots \in E^\omega | (\exists q_0 \in Q_0)(\exists q_1, q_2, \dots \in Q)[q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \cdots]\}$ to denote the set of infinite-length event sequences generated by $\mathcal{S}$. For each $\sigma \in \Sigma^*$, we denote by $\mathcal{M}(\mathcal{S}, \sigma)$ the *current-state estimate*, i.e., the set of states that the system can be in when σ has just been generated, i.e., $\mathcal{M}(\mathcal{S}, \sigma) := \{q \in Q | (\exists q_0 \in Q_0)(\exists s \in E^*)[(\ell(s) = \sigma) \wedge (q_0 \xrightarrow{s} q)]\}$. $\mathcal{L}(\mathcal{S})$ denotes the *language generated* by $\mathcal{S}$, i.e., $\mathcal{L}(\mathcal{S}) := \{\sigma \in \Sigma^* | \mathcal{M}(\mathcal{S}, \sigma) \neq \varnothing\}$. We use $\mathcal{L}^\omega(\mathcal{S})$ to denote the *ω-language generated* by $\mathcal{S}$, i.e., $\mathcal{L}^\omega(\mathcal{S}) := \{\sigma \in \Sigma^\omega | (\exists s \in L^\omega(\mathcal{S}))[\ell(s) = \sigma]\}$.

Example 18: Consider the following LFSA $\mathcal{S}_1$. It is deterministic but not live (q_1 is not live). One sees $L(\mathcal{S}_1) = \{(e_1)^n, (e_1)^n e_2 | n \geqslant 0\}$, $L^\omega(\mathcal{S}_1) = \{(e_1)^\omega\}$, $\mathcal{L}(\mathcal{S}_1) = \{a^n, a^n b | n \geqslant 0\}$, $\mathcal{L}^\omega(\mathcal{S}_1) = \{a^\omega\}$, and $\mathcal{M}(\mathcal{S}_1, b) = \{q_1\}$.

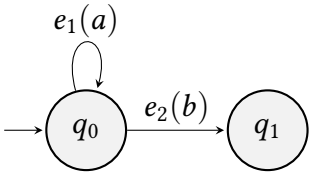


Figure 2.19: An LFSA $\mathcal{S}_1$, where $\ell(e_1) = a$, $\ell(e_2) = b$.

□

2.11 Detectability, diagnosability, and predictability

In this section, we introduce three fundamental properties — detectability [SLY07; SL11; Zha17; Mas18], diagnosability [Sam+95; Jia+01; YL02; CT08], and predictability [GL09], in LFSAs. They are all inference-based properties, where the first refers to inferring states, the latter two refer to inferring a special class of events called faulty events. Detectability describes whether one can use observed output sequences to determine the current and subsequent states, diagnosability (resp., predictability) describes whether one can use observed output sequences to make sure whether some faulty event has occurred (resp., will be certain to occur). A unified mathematical framework for the three properties could be found in [Zha21].

Detectability

We formulate different notions of detectability and study their verification problems. Two fundamental notions are *strong detectability* and *weak detectability*, where the former implies that there is a delay k such that for *each* event sequence generated by an LFSA, each prefix of its output sequence of length greater than k allows reconstructing the current state. The latter relaxes the former by changing *each* to *some*. When long-term behavior is considered, we let the above conditions apply to all infinite-length generated event sequences (in this case we call the notions ω-detectability); when short-term behavior is considered, we let them apply to all finite-length generated event sequences (in this case we call the notions $*$-detectability).

Definition 10: An LFSA $\mathcal{S}$ is called *ω-strongly detectable* if there exists a positive integer k such that for each infinite-length event sequence $s \in L^\omega(\mathcal{S})$, $|\mathcal{M}(\mathcal{S}, \sigma)| = 1$ for every prefix σ of $\ell(s)$ satisfying $|\sigma| > k$.

Definition 11: An LFSA $\mathcal{S}$ is called *ω-weakly detectable* if $L^\omega(\mathcal{S}) \neq \varnothing$ implies there exists an infinite-length event sequence $s \in L^\omega(\mathcal{S})$ such that for some positive integer k, $|\mathcal{M}(\mathcal{S}, \sigma)| = 1$ for every prefix σ of $\ell(s)$ satisfying $|\sigma| > k$.

Definition 12: An LFSA $\mathcal{S}$ is called *$*$-strongly detectable* if there exists a positive integer k such that for each finite-length event sequence $s \in L(\mathcal{S})$, $|\mathcal{M}(\mathcal{S}, \sigma)| = 1$ for every prefix σ of $\ell(s)$ satisfying $|\sigma| > k$.

Definition 13: An LFSA $\mathcal{S}$ is called $*$-*weakly detectable* if there exists a finite-length event sequence $s \in L(\mathcal{S})$ such that for some positive integer k, $|\mathcal{M}(\mathcal{S}, \sigma)| = 1$ for every prefix σ of $\ell(s)$ satisfying $|\sigma| > k$.

Verification of strong detectability: a concurrent-composition method

In order to verify strong detectability, we choose to characterize its negation (one can think about whether it is possible to directly verify strong detectability). By definition, the following proposition holds.

Proposition 2.12: *An LFSA $\mathcal{S}$ is not ω-strongly detectable (resp., $*$-strongly detectable) if and only if for every positive integer k there exists an infinite-length (resp., finite-length) event sequence $s \in L^{\omega}(\mathcal{S})$ (resp., $s \in L(\mathcal{S})$) such that $|\mathcal{M}(\mathcal{S}, \sigma)| > 1$ for some prefix σ of $\ell(s)$ satisfying $|\sigma| > k$.*

In order to characterize negation of strong detectability, we define a notion of *concurrent composition* of two LFSAs.

Definition 14: Consider two LFSAs $\mathcal{S}^i = (Q_i, E, \delta_i, Q_{0i}, \Sigma, \ell)$, $i = 1, 2$, we define the *concurrent composition* $\mathrm{CC_A}(\mathcal{S}^1, \mathcal{S}^2)$ of $\mathcal{S}^1$ and $\mathcal{S}^2$ by

$$\mathrm{CC_A}(\mathcal{S}^1, \mathcal{S}^2) = (Q', E', \delta', Q'_0, \Sigma, \ell'), \tag{2.6}$$

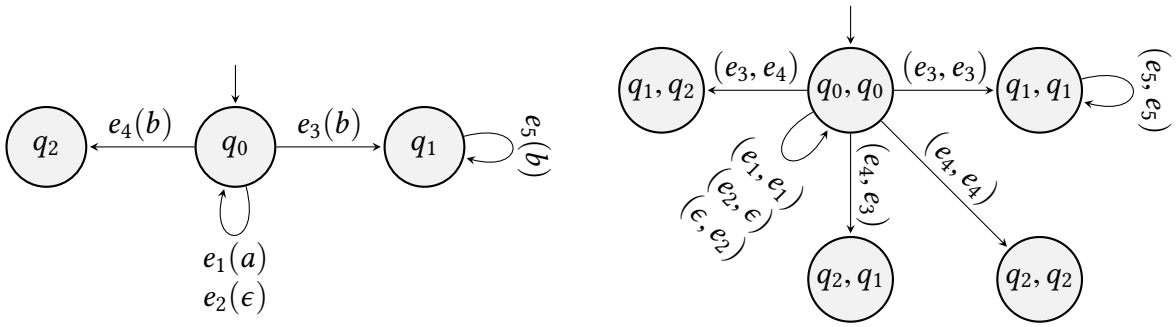
where

1. $Q' = Q_1 \times Q_2$;
2. $E' = E'_o \cup E'_{uo}$, where $E'_o = \{(\check{e}, \check{e}') | \check{e}, \check{e}' \in E_o, \ell(\check{e}) = \ell(\check{e}')\}$, $E'_{uo} = \{(\check{e}, \epsilon) | \check{e} \in E_{uo}\} \cup \{(\epsilon, \check{e}) | \check{e} \in E_{uo}\}$;
3. for all $(\check{q}_1, \check{q}'_1), (\check{q}_2, \check{q}'_2) \in Q'$, $(\check{e}, \check{e}') \in E'_o$, $(\check{e}'', \epsilon) \in E'_{uo}$, and $(\epsilon, \check{e}''') \in E'_{uo}$,
   - $((\check{q}_1, \check{q}'_1), (\check{e}, \check{e}'), (\check{q}_2, \check{q}'_2)) \in \delta'$ if and only if $(\check{q}_1, \check{e}, \check{q}_2) \in \delta_1$, $(\check{q}'_1, \check{e}', \check{q}'_2) \in \delta_2$,
   - $((\check{q}_1, \check{q}'_1), (\check{e}'', \epsilon), (\check{q}_2, \check{q}'_2)) \in \delta'$ if and only if $(\check{q}_1, \check{e}'', \check{q}_2) \in \delta_1$, $\check{q}'_1 = \check{q}'_2$,
   - $((\check{q}_1, \check{q}'_1), (\epsilon, \check{e}'''), (\check{q}_2, \check{q}'_2)) \in \delta'$ if and only if $\check{q}_1 = \check{q}_2$, $(\check{q}'_1, \check{e}''', \check{q}'_2) \in \delta_2$;
4. $Q'_0 = Q_{01} \times Q_{02}$;
5. for all $(\check{e}, \check{e}') \in E'$, $\ell'((\check{e}, \check{e}')) := \ell(\check{e}) = \ell(\check{e}')$.

Particularly, if $\mathcal{S}^1 = \mathcal{S}^2$, then $\mathrm{CC_A}(\mathcal{S}^1, \mathcal{S}^2) =: \mathrm{CC_A}(\mathcal{S}^1)$ is called the *self-composition* of $\mathcal{S}^1$.

For an event sequence $s' \in (E')^*$, we use $s'(L)$ and $s'(R)$ to denote its left and right components, respectively. Similar notation is applied to states of Q'. In addition, for every $s' \in (E')^*$, we use $\ell(s')$ to denote $\ell(s'(L))$ or $\ell(s'(R))$, since $\ell(s'(L)) = \ell(s'(R))$. In the above construction, $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}^1, \mathcal{S}^2)$ aggregates all pairs of runs of $\mathcal{S}_1$ and runs of $\mathcal{S}_2$ that produce the same label sequence.

Example 19: An LFSA $\mathcal{S}_2$ and its self-composition $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_2)$ are shown in Figure 2.20. □



(a) $\mathcal{S}_2$. (b) $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_2)$.

Figure 2.20: An LFSA $\mathcal{S}_2$ (left) and its self-composition (right, only reachable states illustrated).

With the notion of self-composition of an LFSA $\mathcal{S}$, we give sufficient and necessary conditions for negation of two versions of strong detectability.

Theorem 2.13: *An LFSA $\mathcal{S}$ is not $*$-strongly detectable if and only if in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S})$, there exists a run*

$$q'_0 \xrightarrow{s'_1} q'_1 \xrightarrow{s'_2} q'_1 \xrightarrow{s'_3} q'_2 \tag{2.7}$$

satisfying

$$q'_0 \in Q'_0;\ q'_1, q'_2 \in Q';\ s'_1, s'_2, s'_3 \in (E')^*;\ \ell(s'_2) \in \Sigma^+;\ q'_2(L) \neq q'_2(R). \tag{2.8}$$

An LFSA $\mathcal{S}$ is not ω-strongly detectable if and only if in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S})$, there exists a run (2.7) satisfying (2.8) and in $\mathcal{S}$, there exists a transition cycle reachable from $q'_2(L)$.

Proof. We use Proposition 2.12 to prove this theorem. We first consider $*$-strong detectability.

"only if": Assume $\mathcal{S}$ is not $*$-strongly detectable. Then by Proposition 2.12, choose $k = |Q|^2$, there exists $s_k \in L(\mathcal{S})$ and $\sigma \in \Sigma^+$ such that $\mathcal{M}(\mathcal{S}, \sigma) > 1$, $\sigma \sqsubset \ell(s)$, and $|\sigma| > k$. Then in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S})$, there exists a run $q'_0 \xrightarrow{s'} q'$ such that $q'_0 \in Q'_0$, $\ell(s') = \sigma$ and $q'(L) \neq q'(R)$.

Since $|\sigma| > |Q|^2$ and there exist at most $|Q|^2$ distinct states in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S})$, by the pigeonhole principle, the run $q'_0 \xrightarrow{s'} q'$ can be rewritten as $q'_0 \xrightarrow{s'_1} q'_1 \xrightarrow{s'_2} q'_1 \xrightarrow{s'_3} q'$, where $\ell(s'_2) \in \Sigma^+$.

"if": Assume in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S})$ there exists a run (2.7) satisfying (2.8). We choose event sequence $s := s'_1(L)(s'_2(L))^{k+1}s'_3(L) \in L(\mathcal{S})$, then $|\ell(s)| > k$ and $|\mathcal{M}(\mathcal{S}, \ell(s))| > 1$. By Proposition 2.12, $\mathcal{S}$ is not $*$-strongly detectable.

We second consider ω-strong detectability. Because a transition cycle reachable from $q'_2(L)$ can be repeated arbitrarily often, resulting in an infinite-length run starting from $q'_2(L)$. Then based on the above argument for $*$-strong detectability, the sufficient and necessary condition for ω-strong detectability also holds. □

Example 20: Reconsider the LFSA $\mathcal{S}_2$ in Figure 2.20 (left) and its self-composition $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_2)$ in Figure 2.20 (right). In $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_2)$, one sees a run

$$(q_0, q_0) \xrightarrow{(e_1,e_1)} (q_0, q_0) \xrightarrow{(e_3,e_4)} (q_1, q_2)$$

such that $\ell((e_1, e_1)) = a$ is of positive length and $q_1 \neq q_2$. Then by Theorem 2.13, $\mathcal{S}_2$ is not $*$-strongly detectable. In addition, in $\mathcal{S}_2$, there is a self-loop on q_1, hence also by Theorem 2.13, $\mathcal{S}_2$ is not ω-strongly detectable. □

Verification of weak detectability: an observer method

In order to characterize weak detectability, we define a notion of *observer* for an LFSA $\mathcal{S}$. Actually, an observer is nothing new but the powerset construction used for determinizing nondeterministic finite automata with ε-transitions as used in the proof of Theorem 2.6.

Definition 15: Consider an LFSA $\mathcal{S} = (Q, E, \delta, Q_0, \Sigma, \ell)$. We construct an ε-NFA $\mathcal{S}^\varepsilon = (Q, \Sigma_\varepsilon, \delta_\varepsilon, Q_0)$ (with possibly multiple initial states but no final states) from $\mathcal{S}$ by changing every transition (q, e, q') to $(q, \ell(e), q')$ if $\ell(e) \neq \epsilon$, and to (q, ε, q') if $\ell(e) = \epsilon$. We define the *observer* $\mathcal{S}_{\mathrm{obs}}$ of $\mathcal{S}$ as the DFA $\mathcal{S}^\varepsilon_D = (Q', \Sigma, \delta', q'_0)$, where

1. $Q' = 2^Q$,
2. for all $x \in Q'$ and $a \in \Sigma$, $\delta'(x, a) = \varepsilon\text{-CLO}\left(\bigcup_{q\in x} \delta_\varepsilon(q, a)\right)$,
3. $q'_0 = \varepsilon\text{-CLO}\,(Q_0)$,

where the ε-closure ε-CLO is defined in (2.2).

Example 21: The observer $\mathcal{S}_{2\,\mathrm{obs}}$ of the LFSA $\mathcal{S}_2$ in Figure 2.20 is shown in Figure 2.21. □

With the notion of observer, we give sufficient and necessary conditions for two versions of weak detectability.

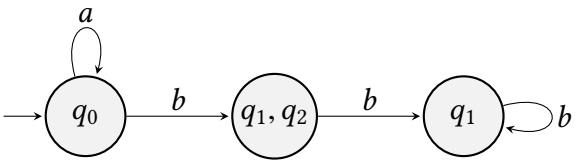


Figure 2.21: Observer $\mathcal{S}_{2\,\mathrm{obs}}$ of the LFSA $\mathcal{S}_2$ in Figure 2.20 (only reachable states illustrated).

Theorem 2.14: *An LFSA $\mathcal{S}$ is ω-weakly detectable if and only if at least one of the following three conditions holds.*

(i) $L^\omega(\mathcal{S}) = \varnothing$.

(ii) $L^\omega(\mathcal{S}) \neq \varnothing$ *and there exists* $s \in L^\omega(\mathcal{S})$ *such that* $\ell(s) \in \Sigma^*$.

(iii) $L^\omega(\mathcal{S}) \neq \varnothing$ *and in its observer* $\mathcal{S}_{\mathrm{obs}}$, *there is a reachable cycle in which all states are singletons.*

Proof. "if": (i) implies that $\mathcal{S}$ is ω-weakly detectable vacuously.

Assume (ii) holds. Choose integer $k > |\ell(s)|$, then one has $\mathcal{S}$ is ω-weakly detectable vacuously.

Assume (iii) holds. Then in $\mathcal{S}_{\mathrm{obs}}$, there is a run $q'_0 \xrightarrow{\gamma_1} q'_1 \xrightarrow{\gamma_2} q'_1$ such that $\gamma_1 \in \Sigma^*$, $\gamma_2 \in \Sigma^+$, and in $q'_1 \xrightarrow{\gamma_2} q'_1$, all states are singletons. Hence in $\mathcal{S}$, there exists an infinite run $q_0 \xrightarrow{s_1} q_1 \xrightarrow{s_2} q_1 (\xrightarrow{s_2} q_1)^\omega$ such that $q_0 \in q'_0$, $\{q_1\} = q'_1$, $\ell(s_1) = \gamma_1$, $\ell(s_2) = \gamma_2$, and $\ell(s_1(s_2)^\omega) = \gamma_1(\gamma_2)^\omega$. For all prefixes $\gamma \sqsubset \gamma_1(\gamma_2)^\omega$ satisfying $|\gamma| \geqslant |\gamma_1|$, one has $|\mathcal{M}(\mathcal{S}, \gamma)| = 1$. Then $\mathcal{S}$ is ω-weakly detectable.

"only if": Assume $\mathcal{S}$ is ω-weakly detectable and neither (i) nor (ii) holds. Then by the finiteness of the number of states of $\mathcal{S}$ and the pigeonhole principle, (iii) holds. □

Similarly, the following result holds.

Theorem 2.15: *An LFSA $\mathcal{S}$ is $*$-weakly detectable if and only if in its observer $\mathcal{S}_{\mathrm{obs}}$, there is a reachable cycle in which all states are singletons.*

Example 22: Reconsider the LFSA $\mathcal{S}_2$ in Figure 2.20 and its observer $\mathcal{S}_{2\,\mathrm{obs}}$ in Figure 2.21. One sees that (iii) of Theorem 2.14 is satisfied, then $\mathcal{S}_2$ is ω-weakly detectable and $*$-weakly detectable by Theorem 2.14 and Theorem 2.15. □

Diagnosability

In order to define *diagnosability* for an LFSA $\mathcal{S} = (Q, E, \delta, Q_0, \Sigma, \ell)$, we specify a subset $E_f \subset E$ of faulty events. Diagnosability describes whether one can use an observed output sequence to determine whether some faulty event has occurred.

Definition 16: Consider an LFSA $\mathcal{S} = (Q, E, \delta, Q_0, \Sigma, \ell)$ and a subset $E_f \subset E$ of faulty events. $\mathcal{S}$ is called *E_f-diagnosable* if

$$(\exists k \in \mathbb{N})(\forall s \in L(\mathcal{S}) \cap E^* E_f)(\forall s' : ss' \in L(\mathcal{S}))$$
$$[(|s'| > k) \implies \mathbf{D}],$$

where $\mathbf{D} = (\forall s'' \in L(\mathcal{S}))[(\ell(s'') = \ell(ss')) \implies (E_f \in s'')]$.

Intuitively, if $\mathcal{S}$ is E_f-diagnosable, then once a faulty event (e.g., the last event in s) occurs, one can make sure that some faulty event has occurred after at least k subsequent events (e.g., s') occur by observing output sequences.

In order to verify E_f-diagnosability of $\mathcal{S}$, we use the concurrent composition $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{\mathrm{f}}, \mathcal{S}_{\mathrm{n}})$ of the faulty subautomaton $\mathcal{S}_{\mathrm{f}}$ and the normal subautomaton $\mathcal{S}_{\mathrm{n}}$, where $\mathcal{S}_{\mathrm{f}}$ is obtained from $\mathcal{S}$ by only keeping faulty transitions and all their predecessors and successors, $\mathcal{S}_{\mathrm{n}}$ is obtained from $\mathcal{S}$ by removing all faulty transitions. $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{\mathrm{f}}, \mathcal{S}_{\mathrm{n}})$ is computed similarly as in Definition 14.

Theorem 2.16: *Consider an LFSA $\mathcal{S} = (Q, E, \delta, Q_0, \Sigma, \ell)$ and a subset $E_f \subset E$ of faulty events. $\mathcal{S}$ is not E_f-diagnosable if and only if in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{\mathrm{f}}, \mathcal{S}_{\mathrm{n}})$, there exists a run*

$$q'_0 \xrightarrow{s'_1} q'_1 \xrightarrow{e'} q'_2 \xrightarrow{s'_2} q'_3 \xrightarrow{s'_3} q'_3 \tag{2.9}$$

satisfying

$$q'_0 \in Q'_0;\ e'(L) \in E_f;\ |s'_3(L)| > 0. \tag{2.10}$$

Proof. By definition, $\mathcal{S}$ is not E_f-diagnosable if and only if

$$(\forall k \in \mathbb{N})(\exists s_k \in L(\mathcal{S}) \cap E^* E_f)(\exists s'_k : s_k s'_k \in L(\mathcal{S}))(\exists s''_k \in L(\mathcal{S}))$$
$$[(|s'_k| > k) \wedge (\ell(s''_k) = \ell(s_k s'_k)) \wedge (E_f \notin s''_k)].$$

Choose sufficiently large k, by the finiteness of the number of states of $\mathcal{S}$ and the pigeonhole principle, $\mathcal{S}$ is not E_f-diagnosable if and only if in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{\mathrm{f}}, \mathcal{S}_{\mathrm{n}})$, there exists a run (2.9) satisfying (2.10). □

Example 23: Consider the LFSA $\mathcal{S}_3$ in Figure 2.22. We compute part of the concurrent composition $\mathrm{CC_A}(\mathcal{S}_{3\mathrm{f}}, \mathcal{S}_{3\mathrm{n}})$ as in Figure 2.23. In $\mathrm{CC_A}(\mathcal{S}_{3\mathrm{f}}, \mathcal{S}_{3\mathrm{n}})$, one sees a run $(q_0, q_0) \xrightarrow{(e_1,e_1)} (q_1, q_2) \xrightarrow{(e_2,e_2)} (q_3, q_4) \xrightarrow{(f,\epsilon)} (q_5, q_4) \xrightarrow{(u,\epsilon)} (q_5, q_4)$, which satisfies (2.10). Then by Theorem 2.16, $\mathcal{S}_3$ is not $\{f\}$-diagnosable. □

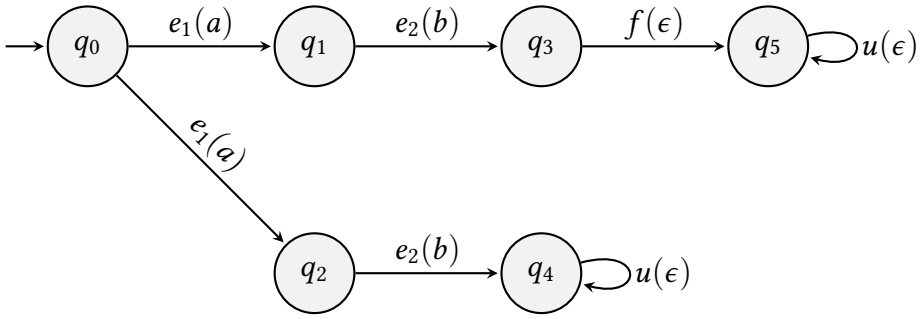


Figure 2.22: An LFSA $\mathcal{S}_3$, where only event f is faulty.

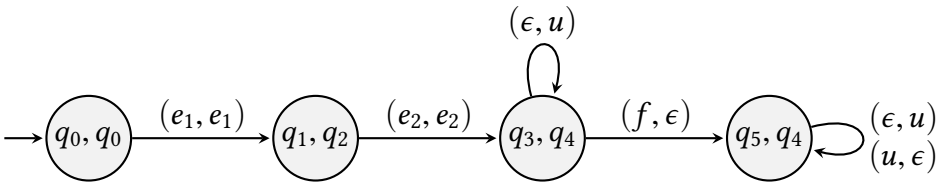


Figure 2.23: Part of $\mathrm{CC_A}(\mathcal{S}_{3\mathrm{f}}, \mathcal{S}_{3\mathrm{n}})$, where $\mathcal{S}_3$ is shown in Figure 2.22.

Predictability

Differently from diagnosability, *predictability* describes whether one can use an observed output sequence to make sure some faulty event will be certain to occur.

Definition 17: Consider an LFSA $\mathcal{S} = (Q, E, \delta, Q_0, \Sigma, \ell)$ and a subset $E_f \subset E$ of faulty events. $\mathcal{S}$ is called *E_f-predictable* if

$$(\exists k \in \mathbb{N})(\forall s \in L(\mathcal{S}) \cap E^* E_f)(\exists s' \sqsubset s : E_f \notin s')(\forall uv \in L(\mathcal{S}))$$
$$[((\ell(s') = \ell(u)) \wedge (E_f \notin u) \wedge (|v| > k)) \implies (E_f \in v)].$$

Intuitively, if $\mathcal{S}$ is E_f-predictable, then once a faulty event will definitely occur, then before any faulty event occurs, one can make sure that after a common time delay (representing the number of occurrences of events, e.g., k), all generated event sequences with the same

observation without any faulty event must be continued by an event sequence containing a faulty event, so as to raise an alarm to definite occurrence of some faulty event.

In order to verify E_f-predictability of $\mathcal{S}$, we use the self-composition $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{\mathrm{n}}, \mathcal{S}_{\mathrm{n}})$ of the normal subautomaton $\mathcal{S}_{\mathrm{n}}$. $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{\mathrm{n}}, \mathcal{S}_{\mathrm{n}})$ is also computed similarly as in Definition 14.

Theorem 2.17: *Consider an LFSA $\mathcal{S} = (Q, E, \delta, Q_0, \Sigma, \ell)$ and a subset $E_f \subset E$ of faulty events. $\mathcal{S}$ is not E_f-predictable if and only if in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{\mathrm{n}}, \mathcal{S}_{\mathrm{n}})$, there exists a run*

$$q'_0 \xrightarrow{s'_1} q'_1 \tag{2.11}$$

such that

$$q'_0 \in Q'_0; \tag{2.12a}$$
$$(q'_1(L), e_f, q) \in \delta \text{ for some } e_f \in E_f \text{ and } q \in Q; \tag{2.12b}$$
$$\text{in } \mathcal{S}_{\mathrm{n}}, \text{ there is a transition cycle reachable from } q'_1(R). \tag{2.12c}$$

Proof. By definition, $\mathcal{S}$ is not E_f-predictable if and only if

$$(\forall k \in \mathbb{N})(\exists s_k \in L(\mathcal{S}) \cap E^* E_f)(\forall s'_k \sqsubset s_k : E_f \notin s'_k)(\exists u_k v_k \in L(\mathcal{S}))$$
$$[(\ell(s'_k) = \ell(u_k)) \wedge (E_f \notin u_k v_k) \wedge (|v_k| > k)].$$

Choose sufficiently large k, by the finiteness of the number of states of $\mathcal{S}$ and the pigeonhole principle, $\mathcal{S}$ is not E_f-predictable if and only if in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{\mathrm{n}}, \mathcal{S}_{\mathrm{n}})$, there exists a run (2.11) satisfying (2.12). □

Example 24: Reconsider the LFSA $\mathcal{S}_3$ in Figure 2.22. Part of the concurrent composition $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{3\mathrm{n}}, \mathcal{S}_{3\mathrm{n}})$ can be obtained from Figure 2.23 by removing the transition with event (f, ϵ). Then in $\mathrm{CC}_{\mathrm{A}}(\mathcal{S}_{3\mathrm{n}}, \mathcal{S}_{3\mathrm{n}})$, one sees a run $(q_0, q_0) \xrightarrow{(e_1, e_1)} (q_1, q_2) \xrightarrow{(e_2, e_2)} (q_3, q_4)$, which satisfies (2.12). Then by Theorem 2.17, $\mathcal{S}_3$ is not $\{f\}$-predictable. □

References

[CT08] F. Cassez and S. Tripakis. "Fault diagnosis with static and dynamic observers". In: *Fundamenta Informaticae* 88.4 (2008), pp. 497–540.

[GJ90] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990.

[GL09] S. Genc and S. Lafortune. "Predictability of event occurrences in partially-observed discrete-event systems". In: *Automatica* 45.2 (2009), pp. 301–311.

[HMU06] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson; 3rd edition, 2006.

[Jia+01] S. Jiang et al. “A polynomial algorithm for testing diagnosability of discrete-event systems”. In: *IEEE Transactions on Automatic Control* 46.8 (Aug. 2001), pp. 1318–1321.

[Mas18] T. Masopust. “Complexity of deciding detectability in discrete event systems”. In: *Automatica* 93 (2018), pp. 257–261.

[RW87] P. J. Ramadge and W.M. Wonham. “Supervisory Control of a Class of Discrete Event Processes”. In: *SIAM Journal on Control and Optimization* 25.1 (1987), pp. 206–230.

[Sam+95] M. Sampath et al. “Diagnosability of discrete-event systems”. In: *IEEE Transactions on Automatic Control* 40.9 (Sept. 1995), pp. 1555–1575.

[Sip96] M. Sipser. *Introduction to the Theory of Computation*. 1st. International Thomson Publishing, 1996.

[SL11] S. Shu and F. Lin. “Generalized detectability for discrete event systems”. In: *Systems & Control Letters* 60.5 (2011), pp. 310–317.

[SLY07] S. Shu, F. Lin, and H. Ying. “Detectability of Discrete Event Systems”. In: *IEEE Transactions on Automatic Control* 52.12 (Dec. 2007), pp. 2356–2359.

[YL02] T.-S. Yoo and S. Lafortune. “Polynomial-time verification of diagnosability of partially observed discrete-event systems”. In: *IEEE Transactions on Automatic Control* 47.9 (Sept. 2002), pp. 1491–1495.

[Zha17] K. Zhang. “The problem of determining the weak (periodic) detectability of discrete event systems is PSPACE-complete”. In: *Automatica* 81 (2017), pp. 217–220.

[Zha21] K. Zhang. “A unified method to decentralized state detection and fault diagnosis/prediction of discrete-event systems”. In: *Fundamenta Informaticae* 181 (2021), pp. 339–371.

3 Boolean (control) networks

3.1 Introduction

Boolean networks (BNs) were proposed by Kauffman [Kau69] to model genetic regulatory networks in 1969. When external regulation or perturbation were considered, BNs were naturally extended to *Boolean control networks* (BCNs) [IGH01]. In a BN/BCN, nodes can be in one of two discrete states "1" and "0", which represent a gene state "on" (high concentration of a protein) and "off" (low concentration), respectively. Every node updates its value according to a Boolean function of the values of the network nodes. According to the intrinsic function principle of BNs/BCNs, they are discrete-time and discrete-space dynamical systems. Although BNs or BCNs are simplified models of genetic regulatory networks, they can be used to characterize many important complex phenomena of biological systems, e.g., cell cycles [Fau+06], cell apoptosis [Sri+12]. Since in 2007 Akutsu et al. [Aku+07] proved that the problem of verifying *controllability* of BCNs is NP-hard in the number of nodes and pointed out that "One of the major goals of systems biology is to develop a control theory for complex biological systems", the control problems of BCNs have drawn wide attention. For related interesting topics in control problems and dynamics of (probabilistic) BNs/BCNs, we recommend monographs [Aku18; CQL11; ZZX20; KS97; Ros15; SD10].

3.2 Preliminaries

In order to characterize control properties of BCNs, *graph theory* and the *semitensor product* (STP) *of matrices* are two useful tools. Basic definitions of graph theory and *finite automata* (see Section 2) will be used to give necessary and sufficient conditions for controllability and *observability* of BCNs, basic definitions of graph theory and the STP of matrices will be used to give necessary and sufficient conditions for the *disturbance decoupling* problem to be solvable.

3.2.1 Graph theory

A *directed graph* is a 2-tuple $(\mathcal{V}, \mathcal{E}) =: \mathcal{G}$, where a finite set $\mathcal{V}$ denotes its *vertex* set, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ denotes its *edge* set. Given two vertices $v_1, v_2 \in \mathcal{V}$, if $(v_1, v_2) \in \mathcal{E}$, then we say "there is an edge from v_1 to v_2", and also denote (v_1, v_2) by $v_1 \to v_2$. Vertices v_1 and v_2 are

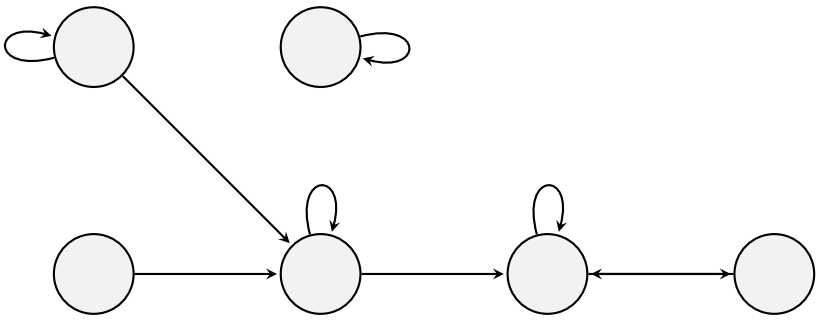


Figure 3.1: A directed graph.

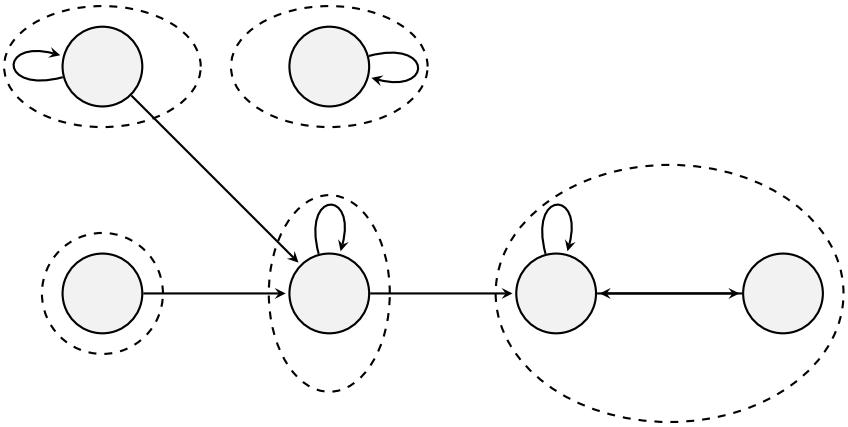


Figure 3.2: Strongly connected components of the directed graph in Figure 3.1.

called the *tail* and the *head* of edge $v_1 \to v_2$, respectively. Vertex v_1 is called a *parent* of v_2 and similarly, vertex v_2 is called a *child* of v_1. Given $v_0, v_1, \ldots, v_p \in \mathcal{V}$ with $p \in \mathbb{N}$, if for all $i \in [\![0, p-1]\!]$, $(v_i, v_{i+1}) \in \mathcal{E}$, then $v_0 \to \cdots \to v_p$ is called a *path*, and p is called the *length* of the path. Particularly, if $v_0 = v_p$, path $v_0 \to \cdots \to v_p$ is called a *cycle*. Cycle $v_0 \to \cdots \to v_p$ is called *simple* if $v_0, \ldots, v_{p-1}$ are pairwise different. An edge from a vertex to itself is called a *self-loop*. A *subgraph* of $\mathcal{G}$ *generated by* $\mathcal{V}_p \subset \mathcal{V}$ is defined as the graph $(\mathcal{V}_p, \mathcal{E}_p)$, where $\mathcal{E}_p = (\mathcal{V}_p \times \mathcal{V}_p) \cap \mathcal{E}$. A subgraph is *strongly connected* if for every two vertices v, v' in the subgraph, there is a path from v to v' also in the subgraph. A *strongly connected component* of $\mathcal{G}$ is a vertex that does not belong to any cycle or a largest strongly connected subgraph of $\mathcal{G}$.

A *weighted directed graph* $(\mathcal{V}, \mathcal{E}, \mathcal{W}, 2^\Sigma)$ is a directed graph $(\mathcal{V}, \mathcal{E})$ such that each edge $e \in \mathcal{E}$ is labeled by a *weight* $w \subsetneq \Sigma$, represented by a function $\mathcal{W} : \mathcal{E} \to 2^\Sigma \backslash \{\varnothing\}$, where Σ is a finite set of symbols. Given vertex $v \in \mathcal{V}$, $\left| \cup_{u \in \mathcal{V}, (v,u) \in \mathcal{E}} \mathcal{W}((v,u)) \right|$ is called the *outdegree* of vertex v, denoted by $\mathrm{outdeg}(v)$; similarly $\left| \cup_{u \in \mathcal{V}, (u,v) \in \mathcal{E}} \mathcal{W}((u,v)) \right|$ is called the *indegree* of vertex v, denoted by $\mathrm{indeg}(v)$. A subgraph is called complete if in the subgraph, each vertex has outdegree $|\Sigma|$. See Figure 3.3 for an illustrative example.

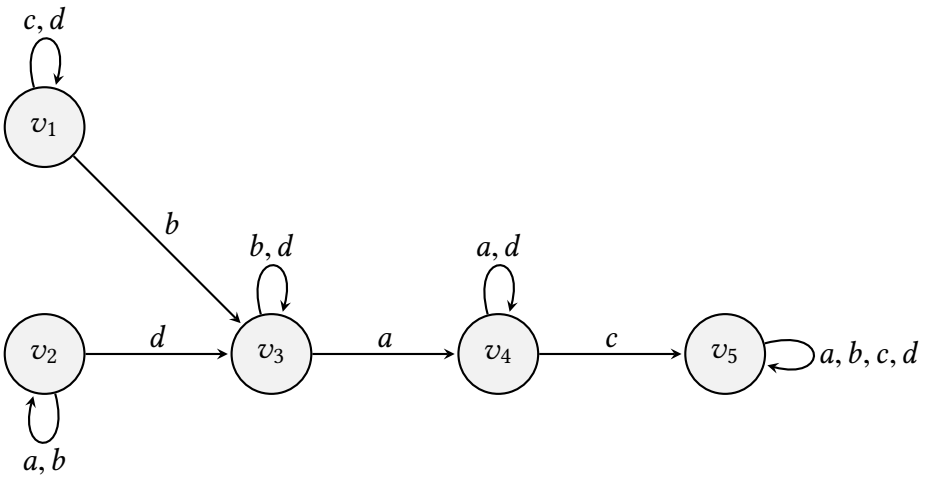


Figure 3.3: A weighted directed graph, where $\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5\}$, $\Sigma = \{a, b, c, d\}$, $\text{indeg}(v_1) = 2$, $\text{outdeg}(v_1) = 3$, $\text{indeg}(v_5) = 5$, $\text{outdeg}(v_5) = 4$.

3.2.2 The semitensor product of matrices

Matrices can be seen as linear operators/transformations on sets (e.g., topological/linear spaces). Then products of matrices are compositions of operators on sets. The conventional matrix product applies to a pair (A, B) of matrices satisfying that the number of columns of A and the number of rows of B are equal, and preserves many properties, such as the associative law, the distributive law, and several inverse-order laws (e.g., $(AB)^{\text{Tr}} = B^{\text{Tr}}A^{\text{Tr}}$). In 2001, Cheng [Che01] proposed a generalization of the conventional matrix product, called the *semitensor product* (STP) *of matrices*, which applies to any pair of matrices, and preserves the associate law, the distributive law, several inverse-order laws, etc. Using STP, logic variables can be represented as vectors, and then logic operations can be represented as the STP of vectors and the so-called logical matrices (a special class of Boolean matrices). Based on this intuitive representation, an equivalent algebraic framework for BNs/BCNs was constructed [CQ09]. Under this framework, results in matrix theory can be used to study BNs/BCNs.

Definition 18: Let $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, and $l = \text{lcm}(n, p)$ be the least common multiple of n and p. The STP of A and B is defined as

$$A \ltimes B = \left(A \otimes I_{\frac{l}{n}}\right)\left(B \otimes I_{\frac{l}{p}}\right),$$

where $\otimes$ denotes the Kronecker product.

Proposition 3.1 (associative law): *Let $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, and $C \in \mathbb{R}^{r\times s}$. Then*

$$A \ltimes (B \ltimes C) = (A \ltimes B) \ltimes C.$$

Proof. Denote $\operatorname{lcm}(n,p) = nn_1 = pp_1$, $\operatorname{lcm}(q,r) = qq_1 = rr_1$, $\operatorname{lcm}(r,qp_1) = qp_1p_2 = rr_2$, and $\operatorname{lcm}(n,pq_1) = nn_2 = pq_1q_2$. Then one has

$$\begin{aligned}
&(A \ltimes B) \ltimes C\\
=&((A \otimes I_{n_1})(B \otimes I_{p_1})) \ltimes C\\
=&(((A \otimes I_{n_1})(B \otimes I_{p_1})) \otimes I_{p_2})(C \otimes I_{r_2})\\
=&(A \otimes I_{n_1} \otimes I_{p_2})(B \otimes I_{p_1} \otimes I_{p_2})(C \otimes I_{r_2})\\
=&(A \otimes I_{n_1p_2})(B \otimes I_{p_1p_2})(C \otimes I_{r_2})
\end{aligned} \tag{3.1}$$

and

$$\begin{aligned}
&A \ltimes (B \ltimes C)\\
=&A \ltimes ((B \otimes I_{q_1})(C \otimes I_{r_1}))\\
=&(A \otimes I_{n_2})(((B \otimes I_{q_1})(C \otimes I_{r_1})) \otimes I_{q_2})\\
=&(A \otimes I_{n_2})(B \otimes I_{q_1} \otimes I_{q_2})(C \otimes I_{r_1} \otimes I_{q_2})\\
=&(A \otimes I_{n_2})(B \otimes I_{q_1q_2})(C \otimes I_{r_1q_2}).
\end{aligned} \tag{3.2}$$

By the associativity law of the least common multiple, one has

$$\begin{aligned}
\operatorname{lcm}(qn, \operatorname{lcm}(pq,pr)) &= \operatorname{lcm}(\operatorname{lcm}(qn,pq),pr),\\
\operatorname{lcm}(qn, p\operatorname{lcm}(q,r)) &= \operatorname{lcm}(q\operatorname{lcm}(n,p),pr),\\
\operatorname{lcm}(qn, pqq_1) &= \operatorname{lcm}(qpp_1,pr),\\
q\operatorname{lcm}(n,pq_1) &= p\operatorname{lcm}(qp_1,r),\\
\frac{\operatorname{lcm}(n,pq_1)}{p} &= \frac{\operatorname{lcm}(r,qp_1)}{q},\\
q_1q_2 &= p_1p_2.
\end{aligned}$$

Furthermore,

$$\begin{aligned}
n_1p_2 &= \frac{\operatorname{lcm}(n,p)}{n}\frac{\operatorname{lcm}(r,qp_1)}{qp_1}\\
&= \frac{\operatorname{lcm}(n,p)}{n}\frac{\operatorname{lcm}(n,pq_1)}{pp_1}\\
&= \frac{\operatorname{lcm}(n,pq_1)}{n}\\
&= n_2,
\end{aligned} \tag{3.3}$$

$$\begin{aligned} r_1 q_2 &= \frac{\operatorname{lcm}(q,r)}{r}\frac{\operatorname{lcm}(n,pq_1)}{pq_1} \\ &= \frac{\operatorname{lcm}(q,r)}{r}\frac{\operatorname{lcm}(r,qp_1)}{qq_1} \\ &= \frac{\operatorname{lcm}(r,qp_1)}{r} \\ &= r_2. \end{aligned} \tag{3.4}$$

Consequently,

$$(A \ltimes B) \ltimes C = A \ltimes (B \ltimes C).$$

□

We have proved that STP preserves the associative law, hence later on we can omit the symbol $\ltimes$, and write $A \ltimes B$ as AB for short.

Proposition 3.2 (inverse-order law)**:** *Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$. Then*

$$(A \ltimes B)^{\mathrm{Tr}} = B^{\mathrm{Tr}} \ltimes A^{\mathrm{Tr}}.$$

Proof. Let $\alpha = \operatorname{lcm}(n,p)$. Then

$$\begin{aligned} (A \ltimes B)^{\mathrm{Tr}} &= \left(\left(A \otimes I_{\alpha/n}\right)\left(B \otimes I_{\alpha/p}\right)\right)^{\mathrm{Tr}} \\ &= \left(B \otimes I_{\alpha/p}\right)^{\mathrm{Tr}}\left(A \otimes I_{\alpha/n}\right)^{\mathrm{Tr}} \\ &= \left(B^{\mathrm{Tr}} \otimes I_{\alpha/p}\right)\left(A^{\mathrm{Tr}} \otimes I_{\alpha/n}\right) \\ &= B^{\mathrm{Tr}} \ltimes A^{\mathrm{Tr}}. \end{aligned}$$

□

Proposition 3.3 (pseudocommutative law)**:** *Let $A \in \mathbb{R}^{m\times n}$ and $z \in \mathbb{R}^{t}$. Then*

$$\begin{aligned} A \ltimes z^{\mathrm{Tr}} &= z^{\mathrm{Tr}} \ltimes (I_t \otimes A), \\ z \ltimes A &= (I_t \otimes A) \ltimes z. \end{aligned}$$

Proof. We only verify $z \ltimes A = (I_t \otimes A) \ltimes z$. The other naturally holds by the inverse-order law.

$$z \ltimes A = (z \otimes I_m) A$$

$$
\begin{aligned}
&= (I_t \otimes A)(z \otimes I_n) \\
&= (I_t \otimes A) \ltimes z.
\end{aligned}
$$

□

Definition 19: The swap matrix, $W_{[m,n]}$, is an $mn \times mn$ matrix defined by

$$W_{[m,n]} := \left[\delta_n^1\delta_m^1, \delta_n^2\delta_m^1, \ldots, \delta_n^n\delta_m^1 \ldots, \delta_n^1\delta_m^m, \delta_n^2\delta_m^m, \ldots, \delta_n^n\delta_m^m\right].$$

Proposition 3.4: *Let $W_{[m,n]}$ be a swap matrix, $P \in \mathbb{R}^m$, and $Q \in \mathbb{R}^n$. Then*

$$W_{[m,n]}^{\mathrm{Tr}} = W_{[m,n]}^{-1} = W_{[n,m]}, \tag{3.5}$$

$$W_{[m,n]}PQ = QP, \tag{3.6}$$

$$P^{\mathrm{Tr}}Q^{\mathrm{Tr}}W_{[m,n]} = Q^{\mathrm{Tr}}P^{\mathrm{Tr}}. \tag{3.7}$$

Proof. By definition, we have

$$
\begin{aligned}
W_{[m,n]}W_{[n,m]} &= \left[\delta_n^1\delta_m^1, \delta_n^1\delta_m^2, \ldots, \delta_n^1\delta_m^m, \ldots, \delta_n^n\delta_m^1, \delta_n^n\delta_m^2, \ldots, \delta_n^n\delta_m^m\right] \\
&= I_{mn}.
\end{aligned}
$$

Then we have $W_{[n,m]}W_{[m,n]} = I_{mn}$, and $W_{[m,n]}^{-1} = W_{[n,m]}$.

As $W_{[m,n]}$ is a logical invertible matrix, it satisfies $W_{[m,n]}W_{[m,n]}^{\mathrm{Tr}} = I_{mn}$, which implies $W_{[m,n]}^{-1} = W_{[m,n]}^{\mathrm{Tr}}$. Hence (3.5) holds.

Let $P = [p_1, \ldots, p_m]^{\mathrm{Tr}}$ and $Q = [q_1, \ldots, q_n]^{\mathrm{Tr}}$. Then we have

$$
\begin{aligned}
PQ &= \sum_{i\in[\![1,m]\!],j\in[\![1,n]\!]} p_iq_j\delta_m^i\delta_n^j, \\
W_{[m,n]}PQ &= \sum_{i\in[\![1,m]\!],j\in[\![1,n]\!]} p_iq_j\delta_n^j\delta_m^i = QP.
\end{aligned}
$$

Hence (3.6) holds. Similarly by the inverse-order law, (3.7) also holds. □

Definition 20: The matrix $M_{k_r} = \delta_k^1 \oplus \cdots \oplus \delta_k^k = [\delta_k^1\delta_k^1, \ldots, \delta_k^k\delta_k^k]$ is called the *power-reducing matrix*. Particularly, we denote $M_{2_r} := M_r$.

By definition, the following proposition holds.

Proposition 3.5: *For power-reducing matrix M_{k_r}, we have*

$$P^2 = M_{k_r} P$$

for each $P \in \Delta_k$.

A matrix $A \in \mathbb{R}^{m\times n}$ is called *Boolean* if each of its entries is either 0 or 1. More particularly, a Boolean matrix $A \in \mathbb{R}^{m\times n}$ is called *logical* if each of its columns is some column of the identity matrix I_m. Hence swap matrices and power-reducing matrices are logical matrices. The set of $m \times n$ logical matrices is denoted by $\mathcal{L}_{m\times n}$. Hence the STP of any two logical matrices is still a logical matrix.

Identifying $1 \sim \delta_2^1$, $0 \sim \delta_2^2$, where δ_2^i is the i-th column of the identity matrix I_2, using STP, a Boolean function $f : \mathcal{D}^n \to \mathcal{D}$ can be uniquely represented by a logical matrix.

Proposition 3.6: *For a Boolean function $f : \mathcal{D}^n \to \mathcal{D}$, there exists a unique logical matrix $F \in \mathcal{L}_{2\times 2^n}$ such that*

$$f(x_1, \ldots, x_n) = F\tilde{x}_1 \ltimes \cdots \ltimes \tilde{x}_n,$$

where $x_i \in \mathcal{D}$, $\tilde{x}_i \in \Delta$, and $x_i \sim \tilde{x}_i$, $i \in [\![1, n]\!]$.

Definition 21: Let $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{p\times n}$. The *Khatri-Rao product* of A and B is defined by

$$A * B = \left[\mathrm{col}_1(A) \ltimes \mathrm{col}_1(B), \ldots, \mathrm{col}_n(A) \ltimes \mathrm{col}_n(B)\right].$$

Proposition 3.7: *For two Boolean functions $FA_1 \ltimes \cdots \ltimes A_n$ and $GA_1 \ltimes \cdots \ltimes A_n$, where $A_i \in \Delta$, $i \in [\![1, n]\!]$, $F \in \mathcal{L}_{2^{m_1}\times 2^n}$, $G \in \mathcal{L}_{2^{m_2}\times 2^n}$, one has*

$$(FA_1 \cdots A_n) \ltimes (GA_1 \cdots A_n) = HA_1 \cdots A_n,$$

where $H = F * G$. One also has

$$F = (I_{2^{m_1}} \otimes \mathbf{1}^{\mathrm{Tr}}_{2^{m_2}})H,$$
$$G = (\mathbf{1}^{\mathrm{Tr}}_{2^{m_1}} \otimes I_{2^{m_2}})H.$$

3.3 The definition of Boolean (control) network

Definition 22: A *Boolean network* (BN) is formulated as the following logical form

$$\begin{aligned} x_1(t+1) &= f_1(x_1(t), \cdots, x_n(t)), \\ x_2(t+1) &= f_2(x_1(t), \cdots, x_n(t)), \\ &\vdots \\ x_n(t+1) &= f_n(x_1(t), \cdots, x_n(t)), \end{aligned} \tag{3.8}$$

where $t = 0, 1, 2, \ldots$ denote discrete time steps, $x_i(t) \in \mathcal{D} := \{0, 1\}$ denotes the value of state node x_i at time step t, $f_i : \mathcal{D}^n \to \mathcal{D}$ is a Boolean function, $i \in [\![1, n]\!]$.

The BN (3.8) can be briefly denoted as

$$x(t+1) = f(x(t)), \tag{3.9}$$

where $t = 0, 1, 2, \ldots; x(t) \in \mathcal{D}^n$ stands for the state at time step t; $f : \mathcal{D}^n \to \mathcal{D}^n$ is a logical mapping.

Definition 23: The *dependency graph* of a BN (3.8) is a directed graph $(\mathcal{V}, \mathcal{E})$, where the vertex set $\mathcal{V}$ is $\{x_1, \ldots, x_n\}$, the edge set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is such that there is an edge from x_i to x_j if and only if the value $x_i(t)$ affects the value $x_j(t+1)$, where $i, j \in [\![1, n]\!]$.

The *state-transition graph* of a BN (3.9) is a directed graph $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{D}^n$; $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is as follows: for all $x, x' \in \mathcal{D}^n$, $(x, x') \in \mathcal{E}$ if and only if $x' = f(x)$.

Example 25 (A simple BN)**:**

$$\begin{aligned} A(t+1) &= B(t) \wedge C(t), \\ B(t+1) &= \neg A(t), \\ C(t+1) &= B(t) \vee C(t), \end{aligned} \tag{3.10}$$

where $t = 0, 1, \ldots; A(t), B(t), C(t) \in \mathcal{D}$. Its dependency graph and state-transition graph are shown in Figure 3.4 and Figure 3.5, respectively.

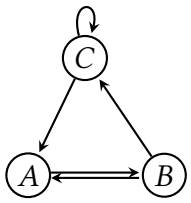


Figure 3.4: Dependency graph of BN (3.10).

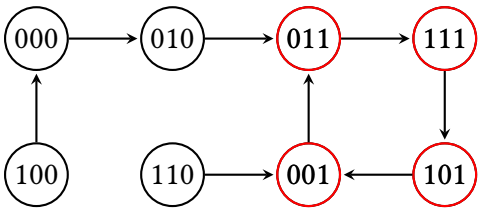


Figure 3.5: State-transition graph of BN (3.10).

□

Definition 24: A *Boolean control network* (BCN) is formulated as the following logical form

$$\begin{aligned}
x_1(t+1) &= f_1(x_1(t),\dots,x_n(t),u_1(t),\dots,u_m(t)),\\
x_2(t+1) &= f_2(x_1(t),\dots,x_n(t),u_1(t),\dots,u_m(t)),\\
&\vdots\\
x_n(t+1) &= f_n(x_1(t),\dots,x_n(t),u_1(t),\dots,u_m(t)),\\
y_1(t) &= h_1(x_1(t),\dots,x_n(t)),\\
y_2(t) &= h_2(x_1(t),\dots,x_n(t)),\\
&\vdots\\
y_q(t) &= h_n(x_1(t),\dots,x_n(t)),
\end{aligned} \tag{3.11}$$

where $t = 0, 1, 2, \dots$ denote discrete time steps; $x_i(t), u_j(t)$, and $y_k(t) \in \mathcal{D}$ denote the values of state node x_i, input node u_j, and output node y_k at time step t, respectively, $f_i : \mathcal{D}^{m+n} \to \mathcal{D}$ and $h_k : \mathcal{D}^n \to \mathcal{D}$ are Boolean functions, $i \in [\![1, n]\!]$, $j \in [\![1, m]\!]$, $k \in [\![1, q]\!]$.

The BCN (3.11) is represented in the compact form

$$\begin{aligned}
x(t+1) &= f(x(t), u(t)),\\
y(t) &= h(x(t)),
\end{aligned} \tag{3.12}$$

where $t = 0, 1, 2, \dots$; $x(t) \in \mathcal{D}^n$, $u(t) \in \mathcal{D}^m$, and $y(t) \in \mathcal{D}^q$ stand for the state, input, and output of (3.12) at time step t; $f : \mathcal{D}^{n+m} \to \mathcal{D}^n$ and $h : \mathcal{D}^n \to \mathcal{D}^q$ are logical mappings.

Definition 25: The *dependency graph* of a BCN (3.11) is a directed graph $(\mathcal{V}, \mathcal{E})$, where the vertex set $\mathcal{V}$ is equal to $\{x_1, \dots, x_n, u_1, \dots, u_m, y_1, \dots, y_q\}$, the edge set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is such that there is an edge from x_i (resp., u_j) to x_k (resp., y_l) if and only if the value $x_i(t)$ (resp., $u_j(t)$) affects the value $x_k(t+1)$ (resp., $y_l(t)$), where $i, k \in [\![1, n]\!]$, $j \in [\![1, m]\!]$, $l \in [\![1, q]\!]$.

The *state-transition graph* of a BCN (3.12) is a weighted directed graph $(\mathcal{V}, \mathcal{E}, \mathcal{W})$,

where $\mathcal{V} = \mathcal{D}^n$; $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is as follows: for all $x, x' \in \mathcal{D}^n$, $(x, x') \in \mathcal{E}$ if and only if there exists $u \in \mathcal{D}^m$ such that $x' = f(x, u)$; and for every edge $e = (x, x') \in \mathcal{E}$, $\mathcal{W}(e) = \{u \in \mathcal{D}^m | x' = f(x, u)\}$.

Example 26 (A simple BCN)**:**

$$\begin{aligned} A(t+1) &= B(t) \wedge u(t), \\ B(t+1) &= \neg A(t), \\ y(t) &= A(t), \end{aligned} \tag{3.13}$$

where $t \in \mathbb{N}, A(t), B(t), u(t), y(t) \in \mathcal{D}$. Its dependency graph and state-transition graph are shown in Figure 3.6 and Figure 3.7, respectively.

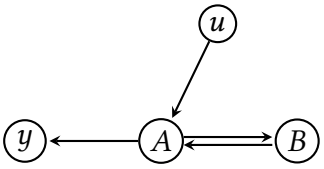


Figure 3.6: Dependency graph of BCN (3.13).

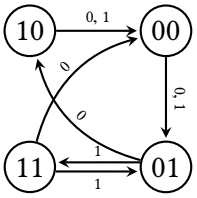


Figure 3.7: State-transition graph of BCN (3.13).

□

3.4 Fixed points and attractors of BNs

A *fixed point* of a BN is a state such that once the BN is in this state, it will never leave the state. It is a fundamental problem to check whether a BN has a fixed point and to compute its fixed points (if any). *Attractors* are cycles of the state-transition graph of a BN. Once a BN enters an attractor, it will never leave the attractor. Fixed points can be regarded as attractors of length 1. Fixed points and attractors are of particular importance in systems biology, because they correspond to certain cellular phenotypes [Alb04].

Definition 26: Consider a BN (3.9). A state $x^* \in \mathcal{D}^n$ is called a *fixed point* of (3.9) if $x^* = f(x^*)$. An *attractor* of (3.9) is a cycle of the state-transition graph of (3.9).

Because a BN is deterministic, starting from every state, it will enter some attractor after finite steps of iterations. Hence a BN has at least one attractor.

Example 27: Consider BN (3.10) in Example 25, it has a unique attractor 011 → 111 → 101 → 001 → 011. When the BN is in state 100, it will enter the attractor after 3 steps of iterations. After that, the BN will be in the attractor forever. □

3.5 Controllability of BCNs

Controllability is a basic control property, which implies that whether there exist control inputs driving a system from each state to each state.

Definition 27: A BCN (3.12) is called *controllable* if for all states $x_0, x_d \in \mathcal{D}^n$, if $x(0) = x_0$, then $x(p) = x_d$ for some $p \in \mathbb{Z}_+$ and some inputs $u(0), u(1), \ldots, u(p-1) \in \mathcal{D}^m$.

By definition, the following result holds.

Proposition 3.8: *A BCN (3.12) is controllable if and only if its state-transition graph is strongly connected.*

Example 28: Consider BCN (3.13) in Example 26, its state-transition graph (in Figure 3.7) is strongly connected, so (3.13) is controllable. For example, the input sequence 001 drives (3.13) from state 10 to 11 via $10 \xrightarrow{0} 00 \xrightarrow{0} 01 \xrightarrow{1} 11$. □

3.6 Observability of BCNs

Observability is also a fundamental control property, which means that whether one can use a control input and the corresponding output to determine the initial state of a partially-observed system. For linear time-invariant control systems, observability is dual to controllability [Kal63; Won85], because observability has nothing to do with how to choose input (this is because the sum of two solutions of such systems is still a solution), so different definitions of observability in such systems are equivalent. However, because BCNs are highly nonlinear (they are polynomial systems over vector space $\mathcal{D}^n$ with $(\mathcal{D}, \oplus, \otimes)$ a field, where $\oplus$ and $\otimes$ denote the addition and multiplication module 2), observability is not dual to controllability, and there exist nonequivalent definitions of observability according to different ways of choosing input [ZZ16]. In this section, we introduce four pairwise nonequivalent definitions of observability in BCNs and their verification methods.

3.6.1 Different definitions of observability

The first definition of observability is as follows.

Definition 28: A BCN (3.12) is called *multiple-experiment observable* if for very two different initial states $x_0, x_0' \in \mathcal{D}^n$, there is an input sequence $U \in (\mathcal{D}^m)^p$[1] for some $p \in \mathbb{N}$ such that the output sequence generated by x_0 and U is different from the one generated by x_0' and U. Such an input sequence U is called a *distinguishing input sequence* of x_0 and x_0'.

If a BCN is multiple-experiment observable, then every two different initial states have a distinguishing input sequence, and then the initial state can be determined by doing a finite number of experiments. In the first experiment, assume two different initial states as candidates, and feed one of their distinguishing input sequences into the BCN, then at least one of the obtained output sequences differs from the real output sequence (generated by the real initial state and the input sequence), hence at least one of the two candidates is not the real initial state, and mark each candidate that is not real. In the second experiment, repeat the first experiment by assuming two different initial states as candidates from the unmarked initial states. Repeat the experiment until there is only one unmarked initial state left. Then the only state is the real initial state.

The second is as follows.

Definition 29: A BCN (3.12) is called *strongly multiple-experiment observable* if for every initial state $x_0 \in \mathcal{D}^n$, there exists an input sequence $U \in (\mathcal{D}^m)^p$ for some $p \in \mathbb{N}$ such that for each initial state $x_0' \in \mathcal{D}^n$ different from x_0, the output sequence generated by x_0 and U is different from the one generated by x_0' and U. Such an input sequence U is called a *distinguishing input sequence* of x_0.

If a BCN is strongly multiple-experiment observable, then every initial state has a distinguishing input sequence, and then the initial state can be determined also by doing a finite number of experiments (sometimes only one experiment is enough). In the first experiment, assume one initial state x_0' as a candidate, and feed one of its distinguishing input sequences into the BCN, then x_0' is the real initial state x_0 if and only if the output sequence generated by x_0' and U is the same as the real output sequence (generated by x_0 and U). With any luck, after the first experiment, the real initial state can be found. With no luck, after repeating this experiment for finitely many times for different candidates, the real initial state will be determined. Intuitively, Definition 29 is stronger than Definition 28. Check whether they are equivalent.

The third is as follows.

Definition 30: A BCN (3.12) is called *single-experiment observable* if there exists an input sequence $U \in (\mathcal{D}^m)^p$ for some $p \in \mathbb{N}$ such that for every two different initial states $x_0, x_0' \in \mathcal{D}^n$, the output sequence generated by x_0 and U is different from the one generated by x_0' and U. Such an input sequence U is called a *distinguishing input sequence* of (3.12).

If a BCN is single-experiment observable, then the initial state can be determined by every distinguishing input sequence. Hence, in order to determine the initial state, one only needs to do one experiment. Definition 30 is stronger than Definition 29. Check whether they are equivalent.

The fourth is as follows.

Definition 31: A BCN (3.12) is called *arbitrary-experiment observable* if for every two different initial states $x_0, x_0' \in \mathcal{D}^n$, for every input sequence $U \in (\mathcal{D}^m)^p$ with p sufficiently large, the output sequence generated by x_0 and U is different from the one generated by x_0' and U.

If a BCN is arbitrary-experiment observable, then the initial state can be determined by every sufficiently long input sequence. Hence, in order to determine the initial state, one also only needs to do one experiment. Definition 31 is stronger than Definition 30. Check whether they are equivalent.

3.6.2 The notion of observability graph

In order to verify the four definitions of observability, we introduce the tool *observability graph*.

Definition 32: Consider a BCN (3.12). A weighted directed graph $\mathcal{G}_o = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ (with weight set $2^{\mathcal{D}^m}$) is called the *observability graph* of (3.12) if the vertex set $\mathcal{V}$ is $\{\{x, x'\} \in \mathcal{D}^n \times \mathcal{D}^n | h(x) = h(x')\}$, the edge set $\mathcal{E}$ is $\{(\{x_1, x_1'\}, \{x_2, x_2'\}) \in \mathcal{V} \times \mathcal{V} | (\exists u \in \mathcal{D}^m)[(f(x_1, u) = x_2 \wedge f(x_1', u) = x_2') \vee (f(x_1, u) = x_2' \wedge f(x_1', u) = x_2)]\} \subset \mathcal{V} \times \mathcal{V}$, and the weight function $\mathcal{W} : \mathcal{E} \to 2^{\mathcal{D}^m}$ maps each edge $(\{x_1, x_1'\}, \{x_2, x_2'\}) \in \mathcal{E}$ to a set $\{u \in \mathcal{D}^m | (f(x_1, u) = x_2 \wedge f(x_1', u) = x_2') \vee (f(x_1, u) = x_2' \wedge f(x_1', u) = x_2)\}$ of inputs.

A vertex $\{x, x'\}$ is called *diagonal* if $x = x'$, and *non-diagonal* otherwise. Note that for all vertices $\{x, x'\} \in \mathcal{V}$, we have $\{x, x'\} = \{x', x\}$. In each observability graph, the children of a diagonal vertex are always diagonal. In addition, each diagonal vertex will go into a cycle consisting of diagonal vertices, since there are totally finitely many vertices. We then use symbol $\diamond$ to denote the subgraph of an observability graph generated by all diagonal

vertices, and call $\diamond$ *diagonal subgraph*. The subgraph of an observability graph generated by all non-diagonal vertices is called the *non-diagonal subgraph*.

Intuitively, an observability graph collects all state trajectory pairs over the same input sequence and generating the same output sequence.

Example 29: Consider BCN

$$\begin{aligned} x(t+1) &= f(x(t), u(t)), \\ y(t) &= h(x(t)), \end{aligned} \tag{3.14}$$

where $t \in \mathbb{N}$, $x(t) \in \mathcal{D}^2$, $u(t), y(t) \in \mathcal{D}$, f and h are as follows:

(a) f

u \ x	00	01	10	11
0	11	00	10	11
1	11	10	10	11

(b) h

x	00	01	10	11
y	0	0	0	1

The observability graph of (3.14) is shown in Figure 3.8.

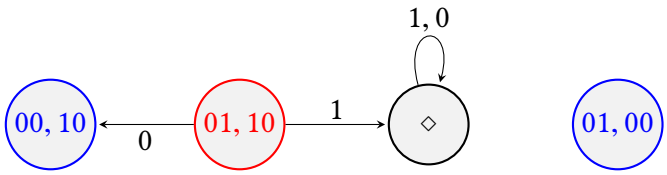


Figure 3.8: Observability graph of BCN (3.14).

□

3.6.3 Verifying different definitions of observability

With the observability graph, we start to verify the above four definitions of observability for BCNs one by one. The idea is fundamental: (1) Given a BCN and construct its observability graph; (2) use the graph to construct a deterministic finite automaton (DFA) (Section 2.3) according to a specific definition of observability; and (3) use the DFA to verify observability (see Figure 3.9). With respect to different definitions of observability, different DFAs will be constructed. Particularly for the multiple-experiment observability (Definition 28) and arbitrary-experiment observability (Definition 31), one can directly use the observability graph to do verification.

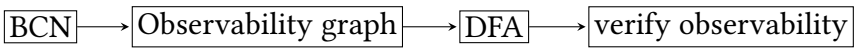


Figure 3.9: Sketch of using the notion of observability graph and finite automata to verify observability of BCNs.

Recall a DFA $A = (Q, \Sigma, \delta, q_0, F)$ as in Definition 1. If $F = Q$, then A recognizes formal language Σ^*. Note that δ is defined at each pair (q, a) in $Q \times \Sigma$. Such standard DFAs are called *complete*. In order to conveniently verify observability of BCNs, *incomplete* DFAs are also considered in this section, in which δ is not defined at at least one pair in $Q \times \Sigma$. The words w accepted by an incomplete DFA A are also defined as those satisfying $\delta(q_0, w) \in F$, and the language recognized by an incomplete DFA A is also defined as the set of all words accepted by A. The following direct proposition will be useful in verifying observability of BCNs.

Proposition 3.9: *Consider a (complete or incomplete) DFA $A = (Q, \Sigma, \delta, q_0, Q)$ (all states are final). Assume each state is reachable from q_0. Then A recognizes language Σ^* if and only if A is complete.*

Proof. "if": Because each state is final and A is complete, it holds that every word is accepted by A, i.e., A recognizes language Σ^*.

"only if": Assume A is incomplete. Choose a state $q \in Q$ such that δ is not defined at (q, a) for some $a \in \Sigma$. Choose a word $w \in \Sigma^*$ such that $\delta(q_0, w) = q$ (because each state is reachable from q_0), then wa is not accepted by A because A is deterministic. That is, the language recognized by A is a strict subset of Σ^*. □

The ideas of verifying different definitions of observability are characterizing their negation, shown in the following four propositions.

Proposition 3.10: *A BCN (3.12) is not multiple-experiment observable (Definition 28) if and only if there exist two different initial states $x_0, x_0' \in \mathcal{D}^n$ such that for each $p \in \mathbb{N}$ and each input sequence $U \in (\mathcal{D}^m)^p$, the corresponding output sequences are the same.*

Proposition 3.11: *A BCN (3.12) is not strongly multiple-experiment observable (Definition 29) if and only if there exists an initial state $x_0 \in \mathcal{D}^n$ such that for each $p \in \mathbb{N}$ and each input sequence $U \in (\mathcal{D}^m)^p$, there exists an initial state $x_0' \in \mathcal{D}^n$ different from x_0 such that the corresponding output sequences are the same.*

Proposition 3.12: *A BCN (3.12) is not single-experiment observable (Definition 30) if and only if for each $p \in \mathbb{N}$ and each input sequence $U \in (\mathcal{D}^m)^p$, there exist two different initial states $x_0, x_0' \in \mathcal{D}^n$ such that the corresponding output sequences are the same.*

Proposition 3.13: *A BCN (3.12) is not arbitrary-experiment observable (Definition 31) if and only if there exist two different initial states $x_0, x_0' \in \mathcal{D}^n$ and an input sequence $U \in (\mathcal{D}^m)^p$ with p sufficiently large such that the corresponding output sequences are the same.*

Verifying multiple-experiment observability

By Definition 28 (multiple-experiment observability) and Definition 32 (observability graph), the following result holds.

Proposition 3.14: *Consider a BCN (3.12). Two different initial states $x_0, x_0' \in \mathcal{D}^n$ with $h(x_0) \neq h(x_0')$ has ϵ as one of their distinguishing input sequences. Two different initial states $x_0, x_0' \in \mathcal{D}^n$ with $h(x_0) = h(x_0')$ has a distinguishing input sequence if and only if in the observability graph $\mathcal{G}_o$ of (3.12) there is a vertex reachable from $\{x_0, x_0'\}$ whose outdegree is less than $|\mathcal{D}^m| = 2^m$.*

Proof. Assume two different initial states $x_0, x_0' \in \mathcal{D}^n$ with $h(x_0) = h(x_0')$. Then $\{x_0, x_0'\}$ is a non-diagonal vertex of $\mathcal{G}_o$. Choose a vertex v (if any) of $\mathcal{G}_o$ that is reachable from $\{x_0, x_0'\}$ and has outdegree less than 2^m. Then v is non-diagonal, because every diagonal vertex has outdegree equal to 2^m. Choose an input sequence U that drives $\{x_0, x_0'\}$ to v. Then the output sequence generated by x_0 and U is the same as the one generated by x_0' and U. Choose an input u such that there is no edge starting from v whose weight contains u, then $h(f(x,u)) \neq h(f(x',u))$, where $\{x, x'\} = v$. That is, Uu is a distinguishing input sequence of x_0 and x_0'. On the other hand, if the above v does not exist, then x_0 and x_0' do not have any distinguishing input sequence. □

By Proposition 3.14, a necessary and sufficient condition for multiple-experiment observability of BCNs can be obtained.

Theorem 3.15: *A BCN (3.12) is multiple-experiment observable if and only if in its observability graph $\mathcal{G}_o$, for every non-diagonal vertex v with outdegree equal to 2^m, there is a path from v to a non-diagonal vertex with outdegree less than 2^m.*

The condition in Theorem 3.15 can be verified in time linear in the size of the observability graph $\mathcal{G}_o = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ as follows: (1) Find the set $V_{<2^m}$ of all vertices of $\mathcal{V}$ that have outdegree less than 2^m, mark the vertices in $V_{<2^m}$, (2) mark all parents of the marked vertices, (3) repeat (2) until there is no unmarked parent of any marked vertex. Then there is no unmarked non-diagonal vertex if and only if the condition holds. The states in every finally unmarked non-diagonal vertex have no distinguishing input sequence.

Example 30: Reconsider BCN (3.14) in Example 29. The non-diagonal vertices of its observability graph are $\{00, 10\}$, $\{01, 10\}$, and $\{01, 00\}$. Because the vertices $\{00, 10\}$ and $\{01, 00\}$ have outdegree less than 2, in the fist step, we mark them; in the second step, we mark $\{01, 10\}$ because it is a parent of $\{00, 10\}$, and there is no unmarked non-diagonal vertex. Then by Theorem 3.15, (3.14) is multiple-experiment observable. By the observability graph (in Figure 3.8), one sees both input sequence 0 and input sequence 1 distinguish initial states 01 and 00, both input sequence 0 and input sequence 1 distinguish initial states 00 and 10, input sequence 01 distinguishes initial states 01 and 10. For example, an illustration of using input sequence 01 to distinguish states 01 and 10 is shown in Table 3.1.

state transition sequence	output sequence
$01 \xrightarrow{0} 00 \xrightarrow{1} 11$	001
$10 \xrightarrow{0} 10 \xrightarrow{1} 10$	000

Table 3.1: An illustration of using input sequence 01 to distinguish initial states 01 and 10.

□

Example 31: Consider BCN

$$\begin{aligned} x(t+1) &= f(x(t), u(t)), \\ y(t) &= h(x(t)), \end{aligned} \tag{3.15}$$

where $t \in \mathbb{N}$, $x(t) \in \mathcal{D}^2$, $u(t), y(t) \in \mathcal{D}$, f and h are as follows:

(a) f

u \ x	00	01	10	11
0	11	10	10	11
1	11	10	10	11

(b) h

x	00	01	10	11
y	0	0	0	1

The observability graph of (3.15) is shown in Figure 3.10.

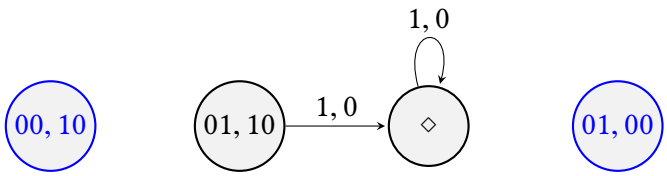


Figure 3.10: Observability graph of BCN (3.15).

To check the condition in Theorem 3.15, we first mark vertices $\{00, 10\}$ and $\{01, 00\}$ because they have outdegree less than 2, and there is no unmarked parent of any of them. Finally, the non-diagonal vertex $\{01, 10\}$ is unmarked, so (3.15) is not multiple-experiment observable. By Figure 3.10, one easily sees that initial states 01 and 10 have no distinguishing input sequence. □

Verifying strong multiple-experiment observability

Verifying strong multiple-experiment observability (Definition 29) is more involved than verifying multiple-experiment observability (Definition 28). By Proposition 3.11, in order to verify strong multiple-experiment observability of a BCN (3.12), one needs a powerset construction from the observability graph $\mathcal{G}_o$ of (3.12).

For every initial state $x_0 \in \mathcal{D}^n$ of (3.12), we consider $\mathcal{G}_o = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ as a (unlabeled) finite-state automaton (FSA, see Definition 9)

$$\mathcal{G}_o^{x_0} = (\mathcal{V}, \mathcal{D}^m, \delta, \mathcal{V}_{x_0}), \tag{3.16}$$

where $\mathcal{V}_{x_0} = \{\{x_0, x_0'\} | h(x_0) = h(x_0'), x_0 \neq x_0'\} \subset \mathcal{V}$ is the set of initial states, δ is a partial function $\mathcal{V} \times \mathcal{D}^m \to \mathcal{V}$ satisfying $\delta(v, u) = v'$ if and only if $(v, v') \in \mathcal{E}$ and $u \in \mathcal{W}((v, v'))$. Over $\mathcal{G}_o^{x_0}$, we construct a (complete or incomplete) DFA

$$\mathcal{G}_{oD}^{x_0} = (2^{\mathcal{V}}, \mathcal{D}^m, \delta', \mathcal{V}_{x_0}, 2^{\mathcal{V}}), \tag{3.17}$$

where $2^{\mathcal{V}}$ is the state set, $\mathcal{V}_{x_0}$ is the initial state, for all $X \in 2^{\mathcal{V}}$ and $u \in \mathcal{D}^m$, $\delta'(X, u) = \{\delta(x, u) | x \in X, \delta \text{ is defined at } (x, u)\}$.

Let $\mathrm{Acc}(\mathcal{G}_{oD}^{x_0})$ be obtained from $\mathcal{G}_{oD}^{x_0}$ by removing all its states that are not reachable from $\mathcal{V}_{x_0}$ and state $\varnothing$.

Theorem 3.16: *A BCN (3.12) is not strongly multiple-experiment observable if and only if there exists an initial state $x_0 \in \mathcal{D}^n$ such that the corresponding DFA $\mathrm{Acc}(\mathcal{G}_{oD}^{x_0})$ recognizes language $(\mathcal{D}^m)^*$, i.e., $\mathrm{Acc}(\mathcal{G}_{oD}^{x_0})$ is complete by Proposition 3.9.*

Proof. For every initial state $x_0 \in \mathcal{D}^n$, by Proposition 3.11 and the construction of DFA $\mathrm{Acc}(\mathcal{G}_{oD}^{x_0})$, a word $U \in (\mathcal{D}^m)^*$ is accepted by $\mathrm{Acc}(\mathcal{G}_{oD}^{x_0})$ if and only if there exists an initial state $x_0' \in \mathcal{D}^n$ different from x_0 such that the output sequence generated by x_0 and U is the same as the one generated by x_0' and U. Hence, each word that is not accepted by $\mathrm{Acc}(\mathcal{G}_{oD}^{x_0})$ can distinguish x_0 from any other initial state. □

Example 32: Reconsider BCN (3.14) in Example 29. The corresponding DFAs $\mathrm{Acc}(\mathcal{G}_{oD}^{10})$, $\mathrm{Acc}(\mathcal{G}_{oD}^{01})$, and $\mathrm{Acc}(\mathcal{G}_{oD}^{00})$ are shown in Figure 3.11 from left to right.

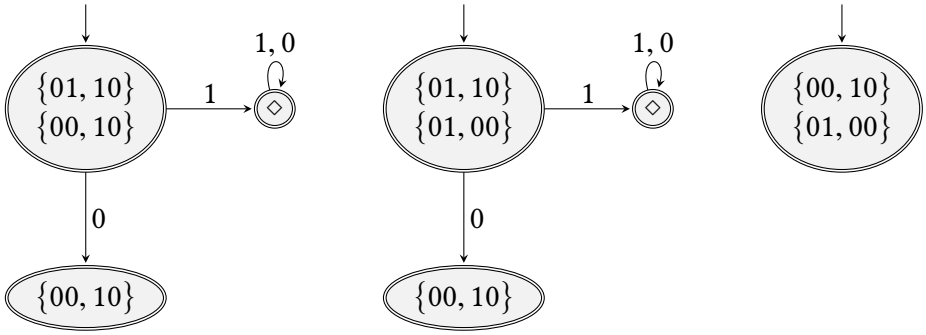


Figure 3.11: DFAs $\mathrm{Acc}(\mathcal{G}_{oD}^{10})$, $\mathrm{Acc}(\mathcal{G}_{oD}^{01})$, and $\mathrm{Acc}(\mathcal{G}_{oD}^{00})$ corresponding to BCN (3.14).

From DFA $\mathrm{Acc}(\mathcal{G}_{oD}^{10})$ (resp., $\mathrm{Acc}(\mathcal{G}_{oD}^{01})$), one sees that for each positive-length input sequence U, $0U$ distinguishes initial state 10 (resp., 01) from any other initial state, but $1U$ cannot do that.

Taking $U = 1$ for example, one has results shown in Table 3.2.

state transition sequence	output sequence
$10 \xrightarrow{0} 10 \xrightarrow{1} 10$	000
$00 \xrightarrow{0} 11 \xrightarrow{1} 11$	011
$01 \xrightarrow{0} 00 \xrightarrow{1} 11$	001
$11 \xrightarrow{0} 11 \xrightarrow{1} 11$	111

Table 3.2: An illustration of using sequence 01 to distinguish initial state 10 from any other initial state.

From DFA $\mathrm{Acc}(\mathcal{G}_{oD}^{00})$, one sees that each positive-length input sequence distinguishes initial state 00 from any other initial state.

Hence BCN (3.14) is strongly multiple-experiment observable by Theorem 3.16. □

Verifying single-experiment observability

To verify single-experiment observability (Definition 30) of a BCN (3.12), by Proposition 3.12, one also needs a powerset construction from the observability graph $\mathcal{G}_o$ of (3.12), but this construction is different from $\mathcal{G}_{oD}^{x_0}$ (3.17) used for verifying strong multiple-experiment observability.

We also consider $\mathcal{G}_o = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ as an FSA

$$\mathcal{G}_o^{\mathcal{V}_{nd}} = (\mathcal{V}, \mathcal{D}^m, \delta, \mathcal{V}_{nd}), \tag{3.18}$$

where $\mathcal{V}_{nd} = \{\{x, x'\} | h(x) = h(x'), x \neq x'\} \subset \mathcal{V}$ is the set of initial states, i.e., the initial states are exactly the non-diagonal vertices of $\mathcal{G}_o$, δ is a partial function $\mathcal{V} \times \mathcal{D}^m \to \mathcal{V}$ satisfying $\delta(v, u) = v'$ if and only if $(v, v') \in \mathcal{E}$ and $u \in \mathcal{W}((v, v'))$. Over $\mathcal{G}_o^{\mathcal{V}_{nd}}$, we construct a DFA

$$\mathcal{G}_{oD}^{\mathcal{V}_{nd}} = (2^{\mathcal{V}}, \mathcal{D}^m, \delta', \mathcal{V}_{nd}, 2^{\mathcal{V}}), \tag{3.19}$$

where $2^{\mathcal{V}}$ is the state set, $\mathcal{V}_{nd}$ is the initial state, for all $X \in 2^{\mathcal{V}}$ and $u \in \mathcal{D}^m$, $\delta'(X, u) = \{\delta(x, u) | x \in X, \delta \text{ is defined at } (x, u)\}$.

Let $\text{Acc}(\mathcal{G}_{oD}^{\mathcal{V}_{nd}})$ be obtained from $\mathcal{G}_{oD}^{\mathcal{V}_{nd}}$ by removing all its states that are not reachable from $\mathcal{V}$ and state $\varnothing$.

Theorem 3.17: *A BCN (3.12) is not single-experiment observable if and only if the corresponding DFA* $\text{Acc}(\mathcal{G}_{oD}^{\mathcal{V}_{nd}})$ *recognizes language* $(\mathcal{D}^m)^*$*, i.e.,* $\text{Acc}(\mathcal{G}_{oD}^{\mathcal{V}_{nd}})$ *is complete by Proposition 3.9.*

Proof. By Proposition 3.12 and the construction of DFA $\text{Acc}(\mathcal{G}_{oD}^{\mathcal{V}_{nd}})$, a word $U \in (\mathcal{D}^m)^*$ is accepted by $\text{Acc}(\mathcal{G}_{oD}^{\mathcal{V}_{nd}})$ if and only if there exist two different initial states $x_0, x_0' \in \mathcal{D}^n$ such that the output sequence generated by x_0 and U is the same as the one generated by x_0' and U. Hence, each word that is not accepted by $\text{Acc}(\mathcal{G}_{oD}^{\mathcal{V}_{nd}})$ can distinguish every pair of different initial states. □

Example 33: Recall BCN (3.14) in Example 29. The corresponding DFA $\text{Acc}(\mathcal{G}_{oD}^{\mathcal{V}_{nd}})$ is shown in Figure 3.12. This DFA is not complete, then by Theorem 3.17, (3.14) is single-experiment observable, and for every positive-length input sequence U, input sequence $0U$ distinguishes every pair of different initial states.

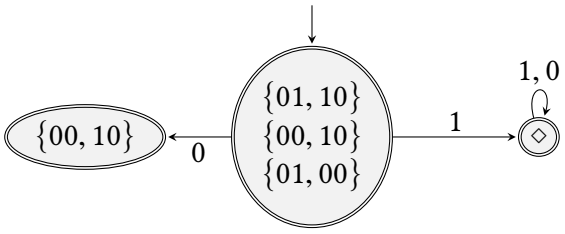


Figure 3.12: DFA $\mathrm{Acc}(\mathcal{G}_{oD}^{\mathcal{V}_{nd}})$ corresponding to BCN (3.14).

Taking $U = 1$ for example, one also has results shown in Table 3.2. □

Verifying arbitrary-experiment observability

Similarly to verifying multiple-experiment observability (Definition 28), one can directly use the observability graph $\mathcal{G}_o$ of a BCN (3.12) to verify its arbitrary-experiment observability (Definition 31). By Proposition 3.13, the following result holds.

Theorem 3.18: *A BCN (3.12) is not arbitrary-experiment observable if and only if in its observability graph $\mathcal{G}_o$, there exist a non-diagonal vertex v, a cycle C, and a path from v to C.*

Proof. "if": Assume in $\mathcal{G}_o$, there exists a path

$$v \xrightarrow{u_1} v_1 \xrightarrow{u_2} \cdots \xrightarrow{u_r} v_r \xrightarrow{u_{r+1}} \cdots \xrightarrow{u_{r+s-1}} v_{r+s-1} \xrightarrow{u_{r+s}} v_r,$$

where $r, s \in \mathbb{Z}_+$, $v = \{x_0, x_0'\}$, $x_0, x_0' \in \mathcal{D}^n$, $x_0 \neq x_0'$, $h(x_0) = h(x_0')$. Then for sufficiently long input sequence $u_1 \dots u_r (u_{r+1} \dots u_{r+s})^p =: U$, where p is sufficiently large, the output sequence generated by x_0 and U is the same as the one generated by x_0' and U. Hence (3.12) is not arbitrary-experiment observable by Proposition 3.13.

"only if": Assume that (3.12) is not arbitrary-experiment observable. By Proposition 3.13, there exist different initial states $x_0, x_0' \in \mathcal{D}^n$ and a sufficiently input sequence $U \in (\mathcal{D}^m)^*$ such that the output sequence generated by x_0 and U is the same as the one generated by x_0' and U. Then $\{x_0, x_0'\} =: v$ is a non-diagonal vertex of the observability graph $\mathcal{G}_o$ of (3.12), and in $\mathcal{G}_o$ there is a sufficiently long path starting from v. Since there are totally finitely many vertices in $\mathcal{G}_o$, the path contains a cycle by the Pigeonhole Principle. □

Example 34: Reconsider BCN (3.14) in Example 29. In its observability graph (Figure 3.8), there is a path from non-diagonal vertex $\{01, 10\}$ to a cycle in its diagonal subgraph, then (3.14) is not arbitrary-experiment observable by Theorem 3.18. □

3.7 Disturbance decoupling of BCNs

In order to study the disturbance decoupling problem of BCNs, we use the STP framework of BCNs. Next we briefly show how to (equivalently) transform a BN (3.8) to its algebraic form (3.21) and how to transform a BCN (3.11) to its algebraic form (3.23), based on STP.

Identifying $1 \sim \delta_2^1$, $0 \sim \delta_2^2$, using the STP of matrices, by Proposition 3.6, a BN (3.8) can be equivalently represented as the following algebraic form

$$\begin{aligned} x_1(t+1) &= L_1 \ltimes_{i=1}^{n} x_i(t), \\ x_2(t+1) &= L_2 \ltimes_{i=1}^{n} x_i(t), \\ &\vdots \\ x_n(t+1) &= L_n \ltimes_{i=1}^{n} x_i(t), \end{aligned} \tag{3.20}$$

where $t = 0, 1, 2, \ldots$, $x_i(t) \in \Delta$, $L_i \in \mathcal{L}_{2\times 2^n}$, $i \in [\![1, n]\!]$.

Then by Definition 21 and Proposition 3.7, (3.20) can be equivalently represented as the algebraic form

$$x(t+1) = Lx(t), \tag{3.21}$$

where $t = 0, 1, \ldots$, $x(t) = \ltimes_{i=1}^{n} x_i(t) \in \Delta_{2^n}$, $x_i(t) \in \Delta$, $i \in [\![1, n]\!]$, $L = L_1 * \cdots * L_n \in \mathcal{L}_{2^n \times 2^n}$ is called its *structure matrix*.

By (3.21), the following result holds.

Proposition 3.19: *Consider a BN (3.21). The number of its fixed points is the trace of L.*

Similarly, by Proposition 3.6, a BCN (3.11) can be equivalently represented as

$$\begin{aligned} x_1(t+1) &= L_1 \ltimes_{i=1}^{n} x_i(t) \ltimes \ltimes_{j=1}^{m} u_j(t), \\ x_2(t+1) &= L_2 \ltimes_{i=1}^{n} x_i(t) \ltimes \ltimes_{j=1}^{m} u_j(t), \\ &\vdots \\ x_n(t+1) &= L_n \ltimes_{i=1}^{n} x_i(t) \ltimes \ltimes_{j=1}^{m} u_j(t), \\ y_1(t) &= H_1 \ltimes_{i=1}^{n} x_i(t), \\ y_2(t) &= H_2 \ltimes_{i=1}^{n} x_i(t), \\ &\vdots \\ y_q(t) &= H_q \ltimes_{i=1}^{n} x_i(t), \end{aligned} \tag{3.22}$$

where $t = 0, 1, 2, \ldots$, $x_i(t) \in \Delta$, $u_j(t) \in \Delta$, $y_k(t) \in \Delta$, $L_i \in \mathcal{L}_{2\times 2^{n+m}}$, $H_k \in \mathcal{L}_{2\times 2^n}$, $i \in [\![1, n]\!]$, $j \in [\![1, m]\!]$, $k \in [\![1, q]\!]$.

Then also by Definition 21 and Proposition 3.7, (3.22) can be equivalently represented as the algebraic form

$$\begin{aligned} x(t+1) &= Lx(t)u(t), \\ y(t) &= Hx(t), \end{aligned} \tag{3.23}$$

where $t = 0, 1, \ldots$, $x(t) = \ltimes_{i=1}^{n} x_i(t) \in \Delta_{2^n}$, $u(t) = \ltimes_{j=1}^{m} u_j(t) \in \Delta_{2^m}$, and $y(t) = \ltimes_{k=1}^{q} y_k(t) \in \Delta_{2^q}$, $x_i(t), u_j(t), y_k(t) \in \Delta$, $i \in [\![1, n]\!]$, $j \in [\![1, m]\!]$, $k \in [\![1, q]\!]$, $L \in \mathcal{L}_{2^n \times 2^{n+m}}$ and $H \in \mathcal{L}_{2^q \times 2^n}$ are called its *structure matrices*.

With the algebraic form (3.23) of a BCN (3.11), we study its disturbance decoupling problem.

We now formulate the disturbance decoupling problem for a BCN with disturbance:

$$\begin{aligned} x(t+1) &= Lx(t)u(t)\xi(t), \\ y(t) &= Hx(t), \end{aligned} \tag{3.24}$$

where $t = 0, 1, 2 \ldots$, $x(t) \in \Delta_{2^n}$ denotes the state, $u(t) \in \Delta_{2^m}$ the input, $\xi(t) \in \Delta_{2^p}$ the disturbance, $y(t) \in \Delta_{2^q}$ the output at time step t, respectively, $L \in \mathcal{L}_{2^n \times 2^{n+m+p}}$, $H \in \mathcal{L}_{2^q \times 2^n}$.

The disturbance decoupling problem means that, whether some state-feedback controller and some coordinate transformation can make the output of (3.24) not be affected by its disturbance. A *state-feedback controller* is $u(t) = Gx(t)$, where $G \in \mathcal{L}_{2^m \times 2^n}$. A *coordinate transformation* is a bijection from $\Delta_{2^n} \to \Delta_{2^n}$, equivalently a mapping $z = Tx$, where $z, x \in \Delta_{2^n}$, $T \in \mathcal{L}_{2^n \times 2^n}$ is a nonsingular matrix. Hence its inverse transformation is $x = T^{-1}z = T^{\mathrm{Tr}}z$.

For brevity, denote $N := 2^n$, $M := 2^m$, $P := 2^p$, $Q := 2^q$.

Problem 1: Consider a BCN (3.24) with disturbance. Its *disturbance decoupling problem* is to check whether there exists a state-feedback controller

$$u(t) = Gx(t) \tag{3.25}$$

and a coordinate transformation

$$z(t) = Tx(t), \tag{3.26}$$

where $G \in \mathcal{L}_{M \times N}$, $T \in \mathcal{L}_{N \times N}$, such that after feeding them into (3.24), one obtains the following BCN (in which ξ does not affect y)

$$z^1(t+1) = \bar{L}_1 z(t)\xi(t), \tag{3.27a}$$

$$z^2(t+1) = \bar{L}_2 z^2(t), \tag{3.27b}$$

$$y(t) = \bar{H} z^2(t), \tag{3.27c}$$

where $t = 0, 1, 2, \ldots$, $z^1(t) \in \Delta_S$, $S = 2^s$ for some $s \in [\![1, n]\!]$, $z^2(t) \in \Delta_{\bar{S}}$, $\bar{S} = N/S = 2^{\bar{s}}$ with $\bar{s} = n - s$, $\bar{L}_1 \in \mathcal{L}_{S \times NP}$, $\bar{L}_2 \in \mathcal{L}_{\bar{S} \times \bar{S}}$, $\bar{H} \in \mathcal{L}_{Q \times \bar{S}}$.

Sometimes, we write "a state-feedback controller $G \in \mathcal{L}_{M\times N}$" for short, and write "a coordinate transformation $T \in \mathcal{L}_{N\times N}$" for short, where T is nonsingular.

In order to solve Problem 1, we first put a state-feedback controller (3.25) and a coordinate transformation (3.26) into (3.24), and obtain the following

$$
\begin{aligned}
z(t+1) &= TLx(t)Gx(t)\xi(t), && (3.28a)\\
&= TL(I_N \otimes G)M_{N_r}x(t)\xi(t) \text{ (by Propositions 3.3 and 3.5)}, && (3.28b)\\
&= TL(I_N \otimes G)M_{N_r}T^{\mathrm{Tr}}z(t)\xi(t), && (3.28c)\\
&= T[L_1,\dots,L_N]\begin{bmatrix} G & & \\ & \ddots & \\ & & G \end{bmatrix}\begin{bmatrix} \delta_N^1 & & \\ & \ddots & \\ & & \delta_N^N \end{bmatrix}T^{\mathrm{Tr}}z(t)\xi(t), && (3.28d)\\
&= T[L_1,\dots,L_N]\begin{bmatrix} G_1 & & \\ & \ddots & \\ & & G_N \end{bmatrix}T^{\mathrm{Tr}}z(t)\xi(t), && (3.28e)\\
&= T[L_1(G_1 \otimes I_P),\dots,L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P)z(t)\xi(t), && (3.28f)\\
y(t) &= HT^{\mathrm{Tr}}z(t), && (3.28g)
\end{aligned}
$$

where $L_i \in \mathcal{L}_{N\times MP}$, $G_i = \mathrm{col}_i(G) \in \mathcal{L}_{M\times 1}$, $i \in [\![1, N]\!]$.

In order to make Problem 1 solvable, one need assume (3.28) coincides with (3.27), then one has

$$
\begin{aligned}
\bar{L}_1 &= (I_S \otimes \mathbf{1}_{\bar{S}}^{\mathrm{Tr}})T[L_1(G_1 \otimes I_P),\dots,L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P), && (3.29a)\\
\mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}} &= (\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})T[L_1(G_1 \otimes I_P),\dots,L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P), && (3.29b)\\
\underbrace{[\bar{H},\dots,\bar{H}]}_{S} &= HT^{\mathrm{Tr}}, && (3.29c)
\end{aligned}
$$

where (3.29a) and (3.29b) hold by Proposition 3.7 (the right-hand side of (3.29a) is the structure matrix of the first s Boolean functions of (3.28f), the right-hand side of (3.29b) is the structure matrix of the last $n-s$ Boolean functions of (3.28f)), and because $\mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}}$ is obtained from (3.27b) as follows:

$$
\begin{aligned}
\bar{L}_2 z^2(t) &= (\mathbf{1}_S^{\mathrm{Tr}} z^1(t)) \otimes (\bar{L}_2 z^2(t)) \otimes (\mathbf{1}_P^{\mathrm{Tr}}\xi(t)),\\
&= (\mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}})(z^1(t) \otimes z^2(t) \otimes \xi(t)),\\
&= (\mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}})(z^1(t) \ltimes z^2(t) \ltimes \xi(t)),\\
&= (\mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}})z(t)\xi(t).
\end{aligned}
$$

Moreover, one has

$$
\underbrace{[\bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}},\dots,\bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}}]}_{S} = \mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}}. \tag{3.30}
$$

$$
\begin{aligned}
&[L_1(G_1\otimes I_P),\dots,L_N(G_N\otimes I_P)](T^{\mathrm{Tr}}\otimes I_P)\\
=&[L_1(G_1\otimes I_P),\dots,L_N(G_N\otimes I_P)]\begin{bmatrix}T_1^{\mathrm{Tr}}\otimes I_P\\ \vdots\\ T_N^{\mathrm{Tr}}\otimes I_P\end{bmatrix}\\
=&\sum_{i=1}^{N}L_i(G_i\otimes I_P)(T_i^{\mathrm{Tr}}\otimes I_P)\\
=&\sum_{i=1}^{N}L_i((G_iT_i^{\mathrm{Tr}})\otimes I_P),
\end{aligned}
$$

where $T_i=\mathrm{col}_i(T)$, $G_iT_i^{\mathrm{Tr}}$ is an $M\times N$ matrix such that only the (i_{G_i},i_{T_i})-entry is equal to 1, all other entries are equal to 0, $G_i=\delta_M^{i_{G_i}}$, $T_i=\delta_N^{i_{T_i}}$, $i\in[\![1,N]\!]$. Then

$$
\sum_{i=1}^{N}L_i((G_iT_i^{\mathrm{Tr}})\otimes I_P)=[\tilde L_1,\dots,\tilde L_N]=:\tilde L, \tag{3.32}
$$

where $\tilde L_j=\mathrm{Blk}_{i_{G_i}}(L_i)\in\mathcal{L}_{N\times P}$ with $L_i=[\mathrm{Blk}_1(L_i),\dots,\mathrm{Blk}_M(L_i)]$, $i_{T_i}=j$, $i,j\in[\![1,N]\!]$. (Because T is nonsingular, for every different i,j, it holds that $i_{T_i}\neq j_{T_j}$.)

By (3.29b), (3.30), (3.31), and (3.32),

$$
[(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T\tilde L_1,\dots,(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T\tilde L_N]=\underbrace{[\bar L_2\otimes\mathbf{1}_P^{\mathrm{Tr}},\dots,\bar L_2\otimes\mathbf{1}_P^{\mathrm{Tr}}]}_{S}, \tag{3.33}
$$

where $(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T\tilde L_i\in\mathcal{L}_{\bar S\times P}$, $i\in[\![1,N]\!]$. Then

$$
[(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T\tilde L_{(i-1)\bar S+1},\dots,(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T\tilde L_{(i-1)\bar S+\bar S}]=\bar L_2\otimes\mathbf{1}_P^{\mathrm{Tr}}, \tag{3.34}
$$

where $i\in[\![1,S]\!]$. That is, for every $i\in[\![1,\bar S]\!]$, $(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T\tilde L_i=(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T\tilde L_{\bar S+i}=\cdots=(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T\tilde L_{(S-1)\bar S+i}$ and all columns of $(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T\tilde L_i$ are the same.

To sum up, the following result holds.

Theorem 3.20: *Problem 1 has a solution if and only if there exists a state-feedback controller $G\in\mathcal{L}_{M\times N}$ and a coordinate transformation $T\in\mathcal{L}_{N\times N}$ such that*

$$
HT^{\mathrm{Tr}}=\underbrace{[\bar H,\dots,\bar H]}_{S}, \tag{3.35a}
$$

$$
(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar S})T[\mathrm{Blk}_{1_{G_1}}(L_1),\dots,\mathrm{Blk}_{N_{G_N}}(L_N)](T^{\mathrm{Tr}}\otimes I_P)=\mathbf{1}_S^{\mathrm{Tr}}\otimes\bar L_2\otimes\mathbf{1}_P^{\mathrm{Tr}}, \tag{3.35b}
$$

where $S=2^s$ for some $s\in[\![1,n]\!]$, $\bar S=N/S$, $\bar H\in\mathcal{L}_{Q\times\bar S}$, $G=[\delta_M^{1_{G_1}},\dots,\delta_M^{N_{G_N}}]$, $L=[L_1,\dots,L_N]$ with $L_i\in\mathcal{L}_{N\times MP}$, $L_i=[\mathrm{Blk}_1(L_i),\dots,\mathrm{Blk}_M(L_i)]$, $\mathrm{Blk}_j(L_i)\in\mathcal{L}_{N\times P}$, $i\in[\![1,N]\!]$, $j\in[\![1,M]\!]$, $\bar L_2\in\mathcal{L}_{\bar S\times\bar S}$.

Because there are finitely many state-feedback controllers and finitely many coordinate transformations, one can verify whether Problem 1 has a solution by using Theorem 3.20.

The *closed-loop BCN* in Theorem 3.20 is

$$z(t+1) = T\big[\mathrm{Blk}_{1_{G_1}}(L_1),\dots,\mathrm{Blk}_{N_{G_N}}(L_N)\big](T^{\mathrm{Tr}}\otimes I_P)z(t)\xi(t), \tag{3.36a}$$
$$y(t) = HT^{\mathrm{Tr}}z(t), \tag{3.36b}$$

where $t = 0,1,2,\dots$. One directly observes that for each $i \in [\![1,N]\!]$, all columns of $\mathrm{Blk}_{i_{G_i}}(L_i)$ are the same; for all $i,i' \in [\![1,S]\!]$, $j \in [\![1,\bar{S}]\!]$, and $k,k' \in [\![1,P]\!]$, it holds that $HT^{\mathrm{Tr}}\delta_S^i\delta_{\bar{S}}^j = HT^{\mathrm{Tr}}\delta_S^{i'}\delta_{\bar{S}}^j$ and

$$(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar{S}})T\big[\mathrm{Blk}_{1_{G_1}}(L_1),\dots,\mathrm{Blk}_{N_{G_N}}(L_N)\big](T^{\mathrm{Tr}}\otimes I_P)\delta_S^i\delta_{\bar{S}}^j\delta_P^k \tag{3.37}$$
$$=(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar{S}})T\big[\mathrm{Blk}_{1_{G_1}}(L_1),\dots,\mathrm{Blk}_{N_{G_N}}(L_N)\big](T^{\mathrm{Tr}}\otimes I_P)\delta_S^{i'}\delta_{\bar{S}}^j\delta_P^{k'}. \tag{3.38}$$

Then one can construct a partition (See Definition 5 in Section 2.4) of the state space Δ_N as $\{\{\delta_S^i\delta_{\bar{S}}^j | i \in [\![1,S]\!]\} | j \in [\![1,\bar{S}]\!]\}$ with $\bar{S}$ cells. Denote each cell $\{\delta_S^i\delta_{\bar{S}}^j | i \in [\![1,S]\!]\}$ by $[\delta_{\bar{S}}^j]$, then one can construct the following *quotient BCN*:

$$[z(t+1)] = T\big[\mathrm{col}_1(\mathrm{Blk}_{1_{G_1}}(L_1)),\dots,\mathrm{col}_1(\mathrm{Blk}_{N_{G_N}}(L_N))\big]T^{\mathrm{Tr}}[z(t)], \tag{3.39a}$$
$$y(t) = HT^{\mathrm{Tr}}[z(t)], \tag{3.39b}$$

where $t = 0,1,2,\dots$, $z \in \Delta_{\bar{S}}$, $T\big[\mathrm{col}_1(\mathrm{Blk}_{1_{G_1}}(L_1)),\dots,\mathrm{col}_1(\mathrm{Blk}_{N_{G_N}}(L_N))\big]T^{\mathrm{Tr}}[z(t)] := [(\mathbf{1}_S^{\mathrm{Tr}}\otimes I_{\bar{S}})T\big[\mathrm{Blk}_{1_{G_1}}(L_1),\dots,\mathrm{Blk}_{N_{G_N}}(L_N)\big](T^{\mathrm{Tr}}\otimes I_P)\delta_S^i z(t)\delta_P^k]$ for any $i \in [\![1,S]\!]$ and any $k \in [\![1,P]\!]$, $HT^{\mathrm{Tr}}[z(t)] := HT^{\mathrm{Tr}}\delta_S^i z(t)$ for any $i \in [\![1,S]\!]$.

Because the functionality of a coordinate transformation T is renaming the vertices in the state-transition graph of a BCN and the quotient BCN (3.39) is equivalently obtained from a BCN (3.24) whose disturbance decoupling problem has a solution, we next give a new sufficient and necessary condition for Problem 1 to have a solution without using T. After putting a state-feedback controller (3.25) into (3.24), we obtain the following closed-loop BCN

$$x(t+1) = [L_1(G_1\otimes I_P),\dots,L_N(G_N\otimes I_P)]x(t)\xi(t), \tag{3.40a}$$
$$y(t) = Hx(t), \tag{3.40b}$$

where $t = 0,1,2,\dots$, G_i and L_i are the same as those in (3.28), $L_i(G_i\otimes I_P) = \mathrm{Blk}_{i_{G_i}}(L_i)$ as in (3.36a), $i \in [\![1,N]\!]$.

Theorem 3.21: *Problem 1 has a solution if and only if there exists a state-feedback controller $G \in \mathcal{L}_{M\times N}$ and a partition $\mathcal{P}$ of the state space Δ_N such that*

(i) every two cells of $\mathcal{P}$ have the same cardinality, denoted as $S = 2^s$ for some $s \in [\![1,n]\!]$,

(ii) for every two states x, x' of (3.40) in the same cell of $\mathcal{P}$, it holds that $Hx = Hx'$,

(iii) for every two states x, x' of (3.40) in the same cell of $\mathcal{P}$, for every $k, k' \in [\![1, P]\!]$, $[L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)]x\delta_P^k$ and $[L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)]x'\delta_P^{k'}$ are also in the same cell of $\mathcal{P}$.

Proof. "if": By (i), write the partition $\mathcal{P}$ as

$$\{\{\delta_N^{1_1}, \ldots, \delta_N^{1_S}\}, \ldots, \{\delta_N^{\bar{S}_1}, \ldots, \delta_N^{\bar{S}_S}\}\},$$

where $\bigcup_{i=1}^{\bar{S}} \{\delta_N^{i_1}, \ldots, \delta_N^{i_S}\} = \Delta_N$, for all $i \in [\![1, \bar{S}]\!]$ and all $j, j' \in [\![1, S]\!]$, $H\delta_N^{i_j} = H\delta_N^{i_{j'}}$. Define coordinate transformation

$$z(t) = Tx(t), \tag{3.41}$$

with $T \in \mathcal{L}_{N\times N}$ as follows:

$$\mathrm{col}_{i_j}(T) = \delta_N^{(j-1)\bar{S}+i}, \tag{3.42}$$

$$\mathrm{col}_{(j-1)\bar{S}+i}(T^{\mathrm{Tr}}) = \delta_N^{i_j}, \tag{3.43}$$

where $i \in [\![1, \bar{S}]\!]$, $j \in [\![1, S]\!]$. Then by (ii), for all $j, j' \in [\![1, S]\!]$ and $i \in [\![1, \bar{S}]\!]$,

$$HT^{\mathrm{Tr}}\delta_S^j\delta_{\bar{S}}^i = H\delta_N^{i_j} = H\delta_N^{i_{j'}} = HT^{\mathrm{Tr}}\delta_S^{j'}\delta_{\bar{S}}^i.$$

That is,

$$HT^{\mathrm{Tr}} =: \underbrace{[\bar{H}, \ldots, \bar{H}]}_{S}, \tag{3.44}$$

i.e., (3.35a) is satisfied. By (iii), for all $i \in [\![1, \bar{S}]\!]$, $j, j' \in [\![1, S]\!]$, and $k, k' \in [\![1, P]\!]$,

$$[L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)]\delta_N^{i_j}\delta_P^k \tag{3.45a}$$

$$= [L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)]T^{\mathrm{Tr}}\delta_N^{(j-1)\bar{S}+i}\delta_P^k \tag{3.45b}$$

$$= [L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P)\delta_S^j\delta_{\bar{S}}^i\delta_P^k \tag{3.45c}$$

and

$$[L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)]\delta_N^{i_{j'}}\delta_P^{k'} \tag{3.46a}$$

$$= [L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)]T^{\mathrm{Tr}}\delta_N^{(j'-1)\bar{S}+i}\delta_P^{k'} \tag{3.46b}$$

$$= [L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P)\delta_S^{j'}\delta_{\bar{S}}^i\delta_P^{k'}, \tag{3.46c}$$

are in the same cell of $\mathcal{P}$. Then we can denote the right-hand side of (3.45c) and the right-hand side of (3.46c) by $\delta_N^{l_t}$ and $\delta_N^{l_{t'}}$, respectively, where $l \in [\![1, \bar{S}]\!]$, $t, t' \in [\![1, S]\!]$. Then

$$\begin{aligned} T\delta_N^{l_t} &= \delta_N^{(t-1)\bar{S}+l} = \delta_S^t \delta_{\bar{S}}^l, \\ T\delta_N^{l_{t'}} &= \delta_N^{(t'-1)\bar{S}+l} = \delta_S^{t'} \delta_{\bar{S}}^l. \end{aligned}$$

Hence

$$\begin{aligned} \delta_S^t \delta_{\bar{S}}^l &= T[L_1(G_1 \otimes I_P), \dots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P)\delta_S^j \delta_{\bar{S}}^i \delta_P^k, \\ \delta_S^{t'} \delta_{\bar{S}}^l &= T[L_1(G_1 \otimes I_P), \dots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P)\delta_S^{j'} \delta_{\bar{S}}^i \delta_P^{k'}, \end{aligned}$$

furthermore,

$$\begin{aligned} \delta_{\bar{S}}^l &= (\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})\delta_S^t \delta_{\bar{S}}^l = (\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})T[L_1(G_1 \otimes I_P), \dots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P)\delta_S^j \delta_{\bar{S}}^i \delta_P^k, \\ \delta_{\bar{S}}^l &= (\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})\delta_S^{t'} \delta_{\bar{S}}^l = (\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})T[L_1(G_1 \otimes I_P), \dots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P)\delta_S^{j'} \delta_{\bar{S}}^i \delta_P^{k'}. \end{aligned}$$

That is, the value of $(\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})T[L_1(G_1 \otimes I_P), \dots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P)\delta_S^j \delta_{\bar{S}}^i \delta_P^k$ does not depend on j, k. Hence

$$(\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})T[L_1(G_1 \otimes I_P), \dots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P) = \mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}} \tag{3.47}$$

for some $\bar{L}_2 \in \mathcal{L}_{\bar{S}\times\bar{S}}$, i.e., (3.35b) is satisfied.

By Theorem 3.20, Problem 1 has a solution as state-feedback controller G and coordinate transformation (3.41).

"only if": By Theorem 3.20, we construct the partition

$$\mathcal{P} = \{\{T^{\mathrm{Tr}}\delta_N^i, T^{\mathrm{Tr}}\delta_N^{\bar{S}+i}, \dots, T^{\mathrm{Tr}}\delta_N^{(S-1)\bar{S}+i}\} | i \in [\![1, \bar{S}]\!]\} \tag{3.48}$$

of Δ_N which satisfies (i). By (3.35a), for all $i \in [\![1, \bar{S}]\!]$ and $j, j' \in [\![1, S]\!]$,

$$\begin{aligned} HT^{\mathrm{Tr}}\delta_N^{(j-1)\bar{S}+i} &= HT^{\mathrm{Tr}}\delta_S^j \delta_{\bar{S}}^i = \bar{H}\delta_{\bar{S}}^i = \\ HT^{\mathrm{Tr}}\delta_N^{(j'-1)\bar{S}+i} &= HT^{\mathrm{Tr}}\delta_S^{j'} \delta_{\bar{S}}^i = \bar{H}\delta_{\bar{S}}^i, \end{aligned}$$

which satisfies (ii). By (3.35b),

$$(\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})T[L_1(G_1 \otimes I_P), \dots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P) = \mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}},$$

where $\bar{L}_2 \in \mathcal{L}_{\bar{S}\times\bar{S}}$. For all $i \in [\![1, \bar{S}]\!]$, $j, j' \in [\![1, S]\!]$, and $k, k' \in [\![1, P]\!]$, choose $T^{\mathrm{Tr}}\delta_N^{(j-1)\bar{S}+i}$ and $T^{\mathrm{Tr}}\delta_N^{(j'-1)\bar{S}+i}$ in the same cell of $\mathcal{P}$ (3.48),

$$\begin{aligned} &(\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})T[L_1(G_1 \otimes I_P), \dots, L_N(G_N \otimes I_P)]T^{\mathrm{Tr}}\delta_N^{(j-1)\bar{S}+i}\delta_P^k \\ =&(\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}})T[L_1(G_1 \otimes I_P), \dots, L_N(G_N \otimes I_P)](T^{\mathrm{Tr}} \otimes I_P)\delta_S^j \delta_{\bar{S}}^i \delta_P^k \end{aligned}$$

$$
\begin{aligned}
&= (\mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}}) \delta_S^j \delta_{\bar{S}}^i \delta_P^k \\
&= \bar{L}_2 \delta_{\bar{S}}^i, \\
&\quad (\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}}) T [L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)] T^{\mathrm{Tr}} \delta_N^{(j'-1)\bar{S}+i} \delta_P^{k'} \\
&= (\mathbf{1}_S^{\mathrm{Tr}} \otimes I_{\bar{S}}) T [L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)] (T^{\mathrm{Tr}} \otimes I_P) \delta_S^{j'} \delta_{\bar{S}}^i \delta_P^{k'} \\
&= (\mathbf{1}_S^{\mathrm{Tr}} \otimes \bar{L}_2 \otimes \mathbf{1}_P^{\mathrm{Tr}}) \delta_S^{j'} \delta_{\bar{S}}^i \delta_P^{k'} \\
&= \bar{L}_2 \delta_{\bar{S}}^i,
\end{aligned}
$$

hence

$$
[L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)] T^{\mathrm{Tr}} \delta_N^{(j-1)\bar{S}+i} \delta_P^k
$$

and

$$
[L_1(G_1 \otimes I_P), \ldots, L_N(G_N \otimes I_P)] T^{\mathrm{Tr}} \delta_N^{(j'-1)\bar{S}+i} \delta_P^{k'}
$$

are in the same cell of $\mathcal{P}$ as in (3.48), which satisfies (iii). □

References

[Aku+07] T. Akutsu et al. "Control of Boolean networks: Hardness results and algorithms for tree structured networks". In: *Journal of Theoretical Biology* 244.4 (2007), pp. 670–679.

[Aku18] T. Akutsu. *Algorithms for Analysis, Inference, and Control of Boolean Networks*. World Scientific, 2018.

[Alb04] R. Albert. "Boolean modeling of genetic regulatory networks". In: *Complex Networks*. Ed. by Eli Ben-Naim, Hans Frauenfelder, and Zoltan Toroczkai. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 459–481.

[Che01] D. Cheng. "Semi-tensor product of matrices and its application to Morgen's problem". In: *Science in China Series : Information Sciences* 44.3 (2001), pp. 195–212.

[CQ09] D. Cheng and H. Qi. "Controllability and observability of Boolean control networks". In: *Automatica* 45.7 (2009), pp. 1659–1667.

[CQL11] D. Cheng, H. Qi, and Z. Li. *Analysis and Control of Boolean Networks: A Semi-tensor Product Approach*. Springer-Verlag London, 2011.

[Fau+06] A. Fauré et al. "Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle". In: *Bioinformatics* 22.14 (2006), e124.

[IGH01] T. Ideker, T. Galitski, and L. Hood. "A new approach to decoding life: systems biology". In: *Annual Review of Genomics and Human Genetics* 2 (2001), pp. 343–372.

[Kal63] R.E. Kalman. "Mathematical description of linear dynamical systems". In: *Journal of the Society for Industrial and Applied Mathematics Series A Control* 1.12 (1963), pp. 152–192.

[Kau69] S.A. Kauffman. "Metabolic stability and epigenesis in randomly constructed genetic nets". In: *Journal of Theoretical Biology* 22.3 (1969), pp. 437–467.

[KS97] W. Kunz and D. Stoffel. *Reasoning in Boolean Networks*. Springer, Boston, MA, 1997.

[Ros15] D. P. Rosin. *Dynamics of Complex Autonomous Boolean Networks*. Springer International Publishing, 2015.

[SD10] I. Shmulevich and E. R. Dougherty. *Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks*. SIAM, 2010.

[Sri+12] S. Sridharan et al. “Boolean modeling and fault diagnosis in oxidative stress response”. In: *BMC Genomics* 13(Suppl 6), S4 (2012), pp. 1–16.

[Won85] W.M. Wonham. *Linear Multivariable Control: a Geometric Approach, 3rd Ed.* Springer-Verlag New York, 1985.

[ZZ16] K. Zhang and L. Zhang. “Observability of Boolean control networks: A unified approach based on finite automata”. In: *IEEE Transactions on Automatic Control* 61.9 (Sept. 2016), pp. 2733–2738.

[ZZX20] K. Zhang, L. Zhang, and L. Xie. *Discrete-Time and Discrete-Space Dynamical Systems*. Communications and Control Engineering. Springer International Publishing, 2020.

4 Weighted automata over monoids and semirings

4.1 Introduction

Weighted automata are finite automata every transition of which carries a weight. According to the structures that the weights belong to, finite automata could be over various algebraic structures, e.g., semigroups, semirings, etc. Weighted automata over (idempotent) semirings have been studied over half a century and plenty of theoretical results and application results can be found in the literature, e.g., see the handbook [DKV09]. While the results on weighted automata over monoids are relatively rare. When the weights are nonnegative rational numbers, they could be interpreted as the time consumptions of the executions of the transitions, so that weighted automata over monoids could be regarded as real-time systems, and weighted automata over dioids (idempotent semirings) could be regarded as max-plus timed systems. When the weights could be negative, they can be interpreted as position deviations of a moving object along with the executions of the transitions. When the weights are chosen between 0 and 1, they can also represent the probabilities of the executions of the transitions, just to name a few. In this chapter, only basic definitions are introduced.

4.2 Monoids and semirings

Definition 33: A *monoid* is a triple $\mathfrak{M} = (T, \otimes, \mathbf{1})$, where $\mathbf{1} \in T$, for all $a, b, c \in T$,

- $a \otimes b \in T$,
- (associativity) $(a \otimes b) \otimes c = a \otimes (b \otimes c)$,
- $a \otimes \mathbf{1} = \mathbf{1} \otimes a = a$ ($\mathbf{1}$ is called *identity* of $\mathfrak{M}$).

Particularly, if there exists an element $\mathbf{0} \in T$ such that $\mathbf{0} \otimes a = a \otimes \mathbf{0} = \mathbf{0}$ for all $a \in T$, then we call $\mathbf{0}$ *zero* of $\mathfrak{M}$.

Any monoid has exactly one identity and at most one zero.

Example 35: $(\mathbb{N}, +, 0)$, $(\mathbb{N}, \cdot, 1)$, $(\mathbb{Z}, +, 0)$, $(\mathbb{Z}, \cdot, 1)$ are monoids. □

Example 36: The $n \times n$ matrix space $(\mathbb{R}^{n\times n}, \cdot, I_n)$ is a monoid, where $\cdot$ denotes the conventional product of matrices. □

Example 37: By the associative law of STP (Proposition 3.1),

$$\left(\bigcup_{m=1}^{\infty}\bigcup_{n=1}^{\infty}\mathbb{R}^{m\times n}, \ltimes, 1\right)$$

is a monoid. □

Example 38: Let Σ be an alphabet. By Proposition 2.8 and Theorem 2.10,

$$(\{L \subset \Sigma^* | L \text{ is regular}\}, \cup, \varnothing) \text{ and } (\{L \subset \Sigma^* | L \text{ is regular}\}, \circ, \{\epsilon\})$$

are monoids. □

Definition 34: A *semiring* is a tuple $\mathfrak{S} = (T, \oplus, \otimes, \mathbf{0}, \mathbf{1})$, where binary operations $\oplus$ and $\otimes$ are called *addition* and *multiplication*, respectively, such that

- $(T, \oplus)$ is a commutative monoid with identity $\mathbf{0} \in T$ (also called *zero*): for all $a, b, c \in T$,
  - $a \oplus b \in T$,
  - $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ (associativity),
  - $\mathbf{0} \oplus a = a \oplus \mathbf{0} = a$,
  - $a \oplus b = b \oplus a$ (commutativity),
- $(T, \otimes)$ is a monoid with identity $\mathbf{1} \in T$ (also called *one*): for all $a, b, c \in T$,
  - $a \otimes b \in T$,
  - $(a \otimes b) \otimes c = a \otimes (b \otimes c)$ (associativity),
  - $\mathbf{1} \otimes a = a \otimes \mathbf{1} = a$,
- $\mathbf{0}$ is absorbing: $\mathbf{0} \otimes a = a \otimes \mathbf{0} = \mathbf{0}$ for all $a \in T$,
- $\otimes$ distributes over $\oplus$: for all $a, b, c \in T$,
  - $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$,
  - $c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$.

Example 39: The following structures

$$\underline{\mathbb{Q}} := (\mathbb{Q} \cup \{-\infty\}, \max, +, -\infty, 0), \tag{4.1a}$$

$$\underline{\mathbb{Q}_{\geqslant 0}} := (\mathbb{Q}_{\geqslant 0} \cup \{-\infty\}, \max, +, -\infty, 0), \tag{4.1b}$$

$$\underline{\mathbb{N}} := (\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0) \tag{4.1c}$$

are semirings, where for all $a \in \mathbb{Q}$, $(-\infty) + a = a + (-\infty) = -\infty$. □

Example 40: Let Σ be an alphabet. By Proposition 2.8 and Theorem 2.10,

$$(\{L \subset \Sigma^* | L \text{ is regular}\}, \cup, \circ, \varnothing, \{\epsilon\})$$

is a semiring. □

Consider a semiring $\mathfrak{S} = (T, \oplus, \otimes, \mathbf{0}, \mathbf{1})$, the operations $\oplus$ and $\otimes$ are extended to matrices over $\mathfrak{S}$ as follows: for all $A = (a_{ij}), B = (b_{ij}) \in T^{m\times n}$ and $C = (c_{jk}) \in T^{n\times p}$,

$$A \oplus B := (d_{ij})_{m\times n},$$
$$B \otimes C := (e_{ik})_{m\times p},$$

where $d_{ij} = a_{ij} \oplus b_{ij}$, $e_{ik} = \bigoplus_{j=1}^{n} b_{ij} \otimes c_{jk}$.

One directly sees the following results.

Proposition 4.1: *Consider a semiring $\mathfrak{S} = (T, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ and matrices $A, B, C \in T^{n\times n}$, it holds that*

$$(A \oplus B) \oplus C = A \oplus (B \oplus C),$$
$$(A \otimes B) \otimes C = A \otimes (B \otimes C),$$
$$(A \oplus B) \otimes C = A \otimes C \oplus B \otimes C,$$
$$A \otimes (B \oplus C) = A \otimes B \oplus A \otimes C.$$

4.3 Labeled weighted automata over monoids and their behavior

Definition 35: A *labeled weighted automaton over monoid* $\mathfrak{M} = (T, \otimes, \mathbf{1})$ is a tuple

$$\mathcal{A}^{\mathfrak{M}} = (\mathfrak{M}, Q, E, Q_0, \Delta, \alpha, \mu, \Sigma, \ell), \tag{4.2}$$

where

1. Q is a finite set of *states*,
2. E an *alphabet* (elements of E are called *events*),
3. $Q_0 \subset Q$ a set of *initial states*,
4. $\Delta \subset Q \times E \times Q$ a set of *transitions*,

5. α assigns to each initial state $q_0 \in Q_0$ a nonzero *weight* $\alpha(q_0) \in T$,
6. μ assigns to each transition $(q, e, q') \in \Delta$ (or rewritten as $q \xrightarrow{e} q'$) a nonzero *weight* $\mu(e)_{qq'} \in T$, where the transition is also denoted by $q \xrightarrow{e/\mu(e)_{qq'}} q'$,
7. Σ a finite set of *outputs/labels*,
8. $\ell : E \to \Sigma \cup \{\epsilon\}$ a *labeling function*.

Particularly, $\mathcal{A}^{\mathbb{Q}^k}$, $\mathcal{A}^{\mathbb{Q}_{\geqslant 0}}$, $\mathcal{A}^{\mathbb{N}}$ denote $\mathcal{A}^{\mathfrak{M}}$ in which $\mathfrak{M}$ is specified as the monoids $(\mathbb{Q}^k, +)$, $(\mathbb{Q}_{\geqslant 0}, +)$, $(\mathbb{N}, +)$, respectively.

Events in $E_{uo} = \{e \in E | \ell(e) = \epsilon\}$ are called *unobservable*, events in $E_o = \{e \in E | \ell(e) \neq \epsilon\}$ are called *observable*. When an observable event $e \in E_o$ occurs, $\ell(e)$ is observed; but when an unobservable event $e \in E_{uo}$ occurs, nothing is observed. A transition $q \xrightarrow{e/\mu(e)_{qq'}} q'$ is called *instantaneous* if $\mu(e)_{qq'} = \mathbf{1}$, and called *noninstantaneous* otherwise. A transition $q \xrightarrow{e/\mu(e)_{qq'}} q'$ is called *observable (resp., unobservable)* if e is observable (resp., unobservable). Automaton $\mathcal{A}^{\mathfrak{M}}$ is called *unambiguous* if under every event sequence, there exists at most one path from the initial states to any given state. Automaton $\mathcal{A}^{\mathfrak{M}}$ is called *deterministic* if (1) $|Q_0| = 1$ and (2) for all states $q, q', q'' \in Q$ and events $e \in E$, if $(q, e, q') \in \Delta$ and $(q, e, q'') \in \Delta$ then $q' = q''$ (hence one also has $\mu(e)_{qq'} = \mu(e)_{qq''}$). If $\mathcal{A}^{\mathfrak{M}}$ is deterministic then it is unambiguous.

Particularly for $\mathcal{A}^{\mathbb{Q}_{\geqslant 0}}$, for initial state $q \in Q_0$, $\alpha(q)$ denotes its initial time delay, and in a transition $q \xrightarrow{e/\mu(e)_{qq'}} q'$, $\mu(e)_{qq'}$ denotes its time delay (i.e., the time consumption of its execution). Hence the execution of an instantaneous transition has time delay 0, i.e., does not cost time, while the execution of a noninstantaneous transition has time delay a positive rational number $\mu(e)_{qq'}$, i.e., costs time $\mu(e)_{qq'}$.

For $q_0, \ldots, q_n \in Q$ and $e_1, \ldots, e_n \in E$, $n \in \mathbb{Z}_+$, a finite sequence

$$\pi := q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} q_n \tag{4.3}$$

of transitions is called a (*finite*) *path*.

The *weighted word* of path π is defined by

$$\tau(\pi) := (e_1, t_1)(e_2, t_2) \ldots (e_n, t_n), \tag{4.4}$$

where for all $i \in [\![1, n]\!]$, $t_i = \bigotimes_{j=1}^{i} \mu(e_j)_{q_{j-1}q_j}$. The *weight* of path π is defined by $t_n =: \mathrm{WT}_\pi$. A path π is called *instantaneous* if $t_1 = \cdots = t_n = \mathbf{1}$, and called *noninstantaneous* otherwise.

Particularly for $\mathcal{A}^{\mathbb{Q}_{\geqslant 0}}$, one has $t_i = \sum_{j=1}^{i} \mu(e_j)_{q_{j-1}q_j}$, hence the t_i in $\tau(\pi)$ can be used to denote the total time consumptions for the first i transitions in path π, $i \in [\![1, n]\!]$. Hence a weighted word of automaton $\mathcal{A}^{\mathbb{Q}_{\geqslant 0}}$ is also called *timed word*. If $\mathcal{A}^{\mathbb{Q}_{\geqslant 0}}$ generates a path π

as in (4.3), consider its timed word $\tau(\pi)$ as in (4.4), then at instant t_i, one observes $\ell(e_i)$ if $\ell(e_i) \neq \epsilon$; and observes nothing otherwise, where $i \in [\![1, n]\!]$. We simply say one observes $\ell(\tau(\pi))$.

We use $L(\mathcal{A}^{\mathfrak{M}})$ to denote the set of weighted words of all paths of $\mathcal{A}^{\mathfrak{M}}$ starting from initial states.

For $q_0, q_1, \ldots \in Q$ and $e_1, e_2, \ldots \in E$, an infinite sequence

$$\pi := q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \cdots \tag{4.5}$$

of transitions is called an *infinite path*. The *ω-weighted word* of infinite path π is defined by

$$\tau(\pi) := (e_1, t_1)(e_2, t_2)\ldots, \tag{4.6}$$

where for all $i \in \mathbb{Z}_+$, $t_i = \bigotimes_{j=1}^{i} \mu(e_j)_{q_{j-1}q_j}$.

We use $L^{\omega}(\mathcal{A}^{\mathfrak{M}})$ by to denote the set of ω-weighted words of all infinite paths of $\mathcal{A}^{\mathfrak{M}}$ starting from initial states. Particularly, we also call an ω-weighted word of automaton $\mathcal{A}^{\mathbb{Q}_{\geqslant 0}}$ *ω-timed word*.

Labeling function ℓ is recursively extended to $E^* \cup E^{\omega} \to \Sigma^* \cup \Sigma^{\omega}$ as $\ell(e_1e_2\ldots) = \ell(e_1)\ell(e_2)\ldots$. It is also extended as follows: for all $(e, t) \in E \times T$, $\ell((e, t)) = (\ell(e), t)$ if $\ell(e) \neq \epsilon$, and $\ell((e, t)) = \epsilon$ otherwise. Hence ℓ is also recursively extended to $(E \times T)^* \cup (E \times T)^{\omega} \to (\Sigma \times T)^* \cup (\Sigma \times T)^{\omega}$. For a weighted word $\tau(\pi)$, where π is a path of $\mathcal{A}^{\mathfrak{M}}$, $\ell(\tau(\pi))$ is called the *weighted label/output sequence* of both π and $\tau(\pi)$. The previously defined function τ is also extended as follows: for all $\gamma = (\sigma_1, t_1)\ldots(\sigma_n, t_n) \in (\Sigma \times T)^*$,

$$\tau(\gamma) = (\sigma_1, t_1')\ldots(\sigma_n, t_n'), \tag{4.7}$$

where $t_j' = \bigotimes_{i=1}^{j} t_i$ for all $j \in [\![1, n]\!]$. Moreover, τ is also extended to $(\Sigma \times T)^{\omega}$ recursively.

The *weighted language* $\mathcal{L}(\mathcal{A}^{\mathfrak{M}})$ and *ω-weighted language* $\mathcal{L}^{\omega}(\mathcal{A}^{\mathfrak{M}})$ generated by $\mathcal{A}^{\mathfrak{M}}$ are defined by

$$\mathcal{L}(\mathcal{A}^{\mathfrak{M}}) := \{\gamma \in (\Sigma \times T)^* | (\exists w \in L(\mathcal{A}^{\mathfrak{M}}))[\ell(w) = \gamma]\} \tag{4.8}$$

and

$$\mathcal{L}^{\omega}(\mathcal{A}^{\mathfrak{M}}) := \{\gamma \in (\Sigma \times T)^{\omega} | (\exists w \in L^{\omega}(\mathcal{A}^{\mathfrak{M}}))[\ell(w) = \gamma]\}, \tag{4.9}$$

respectively. Particularly, $\mathcal{L}(\mathcal{A}^{\mathbb{Q}_{\geqslant 0}})$ and $\mathcal{L}^{\omega}(\mathcal{A}^{\mathbb{Q}_{\geqslant 0}})$ are also called *timed language* and *ω-timed language*, respectively.

Example 41: Consider labeled weighted automaton $\mathcal{A}_1^{\mathbb{N}}$ shown in Fig. 4.1, where only q_0 is initial, event u is unobservable, events a and b are observable, $\ell(a) = \ell(b) = \rho$. Automaton $\mathcal{A}_1^{\mathbb{N}}$ is ambiguous, because q_3 can be reached from q_0 through two paths $q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3$ and $q_0 \xrightarrow{a} q_2 \xrightarrow{b} q_3$ under the same event sequence ab.

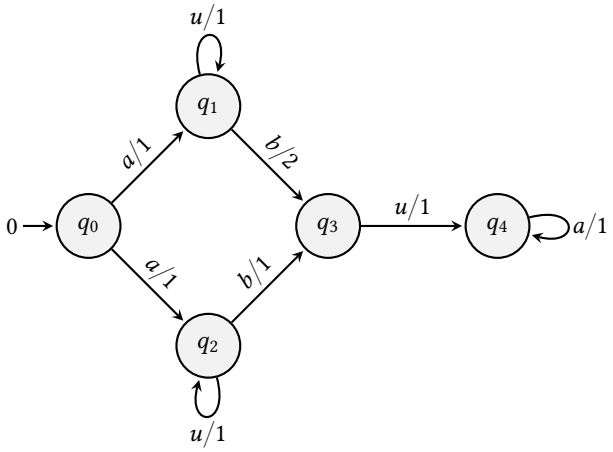


Figure 4.1: Labeled ambiguous weighted automaton $\mathcal{A}_1^{\mathbb{N}}$, where $\ell(u) = \epsilon$, $\ell(a) = \ell(b) = \rho$.

Consider paths

$$\pi_1 = q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3, \quad \pi_2 = q_0 \xrightarrow{a} q_2 \xrightarrow{b} q_3, \tag{4.10a}$$

$$\pi_3 = q_0 \xrightarrow{a} q_1 \xrightarrow{u} q_1 \xrightarrow{b} q_3, \quad \pi_4 = q_0 \xrightarrow{a} q_2 \xrightarrow{u} q_2 \xrightarrow{b} q_3, \tag{4.10b}$$

$$\pi_5 = q_0 \xrightarrow{a} q_2 \xrightarrow{b} q_3 \xrightarrow{u} q_4, \tag{4.10c}$$

one then has

$$\tau(\pi_1) = (a,1)(b,3), \quad \ell(\tau(\pi_1)) = (\rho,1)(\rho,3), \tag{4.11a}$$

$$\tau(\pi_2) = (a,1)(b,2), \quad \ell(\tau(\pi_2)) = (\rho,1)(\rho,2), \tag{4.11b}$$

$$\tau(\pi_3) = (a,1)(u,2)(b,4), \quad \ell(\tau(\pi_3)) = (\rho,1)(\rho,4), \tag{4.11c}$$

$$\tau(\pi_4) = (a,1)(u,2)(b,3), \quad \ell(\tau(\pi_4)) = (\rho,1)(\rho,3), \tag{4.11d}$$

$$\tau(\pi_5) = (a,1)(b,2)(u,3), \quad \ell(\tau(\pi_5)) = (\rho,1)(\rho,2). \tag{4.11e}$$

Path π_1 has the following meaning: $\mathcal{A}_1^{\mathbb{N}}$ starts at initial state q_0; when event a occurs after time segment 1, $\mathcal{A}_1^{\mathbb{N}}$ transitions to state q_1, one observes ρ at time instant 1; when event b occurs after time segment 2 since the occurrence of the previous event a, $\mathcal{A}_1^{\mathbb{N}}$ transitions to state q_3, and one observes ρ at time instant 3. The other paths have similar interpretations. □

4.4 Labeled weighted automata over semirings and their behavior

Definition 36: A *labeled weighted automaton over semiring* $\mathfrak{S} = (T, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a tuple

$$\mathcal{A}^{\mathfrak{S}} = (\mathfrak{S}, Q, E, Q_0, \Delta, \alpha, \mu, \Sigma, \ell), \tag{4.12}$$

where

1. Q is a finite set of *states*,
2. E an *alphabet* (elements of E are called *events*),
3. $Q_0 \subset Q$ a set of *initial states*,
4. $\Delta \subset Q \times E \times Q$ a set of *transitions*,
5. α assigns to each initial state $q_0 \in Q_0$ a nonzero *weight* $\alpha(q_0) \in T$ and assigns to each non-initial state $q \in Q \backslash Q_0$ the *weight* $\alpha(q) = \mathbf{0}$,
6. μ assigns to each transition $(q, e, q') \in \Delta$ (or rewritten as $q \xrightarrow{e} q'$) a nonzero *weight* $\mu(e)_{qq'} \in T$ and assigns to each triple $(q, e, q') \in (Q \times E \times Q) \backslash \Delta$ the *weight* $\mu(e)_{qq'} = \mathbf{0}$,
7. Σ a finite set of *outputs/labels*,
8. $\ell : E \to \Sigma \cup \{\epsilon\}$ a *labeling function.*

By definition, α is a mapping from Q to T (i.e., $\alpha \in T^Q$), μ is a mapping from $Q \times E \times Q$ to T (i.e., $\mu \in T^{Q \times E \times Q}$). In addition, for each $e \in E$, $\mu(e)$ can also be regarded as a mapping from $Q \times Q$ to T (i.e., $\mu(e) \in T^{Q \times Q}$) such that $\mu(e)((q, q')) = \mu((q, e, q'))$. Then for $\mu(e), \mu(e') \in T^{Q \times Q}$, define

$$\alpha \otimes \mu(e) =: \alpha' \in T^Q, \tag{4.13}$$

$$\mu(e) \otimes \mu(e') =: \mu(ee') \in T^{Q \times Q}, \tag{4.14}$$

as follows: for all $q, q' \in Q$,

$$\alpha'(q) = \bigoplus_{q'' \in Q} \alpha(q'') \otimes \mu(e)(q'', q), \tag{4.15}$$

$$\mu(ee')(q, q') = \bigoplus_{q'' \in Q} \mu(e)(q, q'') \otimes \mu(e')(q'', q'), \tag{4.16}$$

particularly, $\mu(\epsilon)$ is such that $\mu(\epsilon)(q, q') = \mathbf{1}$ if $q = q'$ and $= \mathbf{0}$ if $q \neq q'$. Recursively, $\mu(e_1 \dots e_n) := \mu(e_1) \otimes \cdots \otimes \mu(e_n)$ with $n \in \mathbb{Z}_+$.

One directly sees the following associative law.

Proposition 4.2: *Consider an automaton* $\mathcal{A}^{\mathfrak{S}}$ (4.12) *and events* $e_1, e_2, e_3 \in E$, *then*

$$(\alpha \otimes \mu(e_1)) \otimes \mu(e_2) = \alpha \otimes (\mu(e_1) \otimes \mu(e_2)), \tag{4.17}$$

$$(\mu(e_1) \otimes \mu(e_2)) \otimes \mu(e_3) = \mu(e_1) \otimes (\mu(e_2) \otimes \mu(e_3)). \tag{4.18}$$

Particularly, $\mathcal{A}^{\underline{\mathbb{Q}}}$, $\mathcal{A}^{\underline{\mathbb{Q}_{\geqslant 0}}}$, $\mathcal{A}^{\underline{\mathbb{N}}}$ denote $\mathcal{A}^{\mathfrak{S}}$ in which $\mathfrak{S}$ is specified as the semirings $\underline{\mathbb{Q}}$, $\underline{\mathbb{Q}_{\geqslant 0}}$, $\underline{\mathbb{N}}$ as in (4.1), respectively.

Due to the additional operation $\oplus$ in $\mathcal{A}^{\mathfrak{S}}$, the behavior of $\mathcal{A}^{\mathfrak{S}}$ remarkably differs from that of $\mathcal{A}^{\mathfrak{M}}$.

Consider a path

$$\pi := q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} q_n \tag{4.19}$$

of $\mathcal{A}^{\mathfrak{S}}$, where $q_0 \in Q_0$, $n \in \mathbb{Z}_+$, its *weighted sequence* is

$$\sigma(\pi) = (e_1, t_1')(e_2, t_2') \dots (e_n, t_n'), \tag{4.20}$$

where for every $i \in [\![1, n]\!]$, $t_i' = (\alpha \otimes \mu(e_1 \dots e_i))(q_i)$. The *weighted label sequence* $\ell(\sigma(\pi))$ is defined by replacing each (e_i, t_i') by $(\ell(e_i), t_i')$ if e_i is observable, erasing (e_i, t_i') otherwise.

Particularly for $\mathcal{A}^{\underline{\mathbb{Q}_{\geqslant 0}}}$, in a path (4.19) and its weighted sequence (4.20), where for every $i \in [\![1, n]\!]$, t_i' is the maximum among the weights of all paths from Q_0 to q_i under event sequence $e_1 \dots e_i$.

Example 42: Now we use automaton $\mathcal{A}_1^{\mathbb{N}}$ (which can also be regarded as automaton $\mathcal{A}_1^{\underline{\mathbb{N}}}$ over semiring $\underline{\mathbb{N}}$) in Fig. 4.1 to show differences between labeled weighed automata over monoids and over semirings. Reconsider the paths $\pi_1, \dots, \pi_5$ in (4.10). By definition, one has

$$\sigma(\pi_1) = (a,1)(b,3), \quad \ell(\sigma(\pi_1)) = (\rho,1)(\rho,3), \tag{4.21a}$$
$$\sigma(\pi_2) = (a,1)(b,3), \quad \ell(\sigma(\pi_2)) = (\rho,1)(\rho,3), \tag{4.21b}$$
$$\sigma(\pi_3) = (a,1)(u,2)(b,4), \quad \ell(\sigma(\pi_3)) = (\rho,1)(\rho,4), \tag{4.21c}$$
$$\sigma(\pi_4) = (a,1)(u,2)(b,4), \quad \ell(\sigma(\pi_4)) = (\rho,1)(\rho,4), \tag{4.21d}$$
$$\sigma(\pi_5) = (a,1)(b,3)(u,4), \quad \ell(\sigma(\pi_5)) = (\rho,1)(\rho,3). \tag{4.21e}$$

Compared with the results in Example 41, one sees that for path π_2 in (4.10), its weighted word $\tau(\pi_2)$ (shown in (4.11)) is not the same as its weighted sequence $\sigma(\pi_2)$ in (4.21). This is because $\mathcal{A}_1^{\underline{\mathbb{N}}}$ is not unambiguous: under event sequence ab, q_3 is reachable from the initial states via two paths π_1 and π_2,

$$\alpha = \begin{matrix} q_0 & q_1 & q_2 & q_3 & q_4 \end{matrix} \\ \begin{bmatrix} 0 & -\infty & -\infty & -\infty & -\infty \end{bmatrix},$$

$$\mu(a) = \begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} \begin{array}{c} \begin{array}{ccccc} q_0 & q_1 & q_2 & q_3 & q_4 \end{array} \\ \begin{bmatrix} -\infty & 1 & 1 & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & 1 \end{bmatrix} \end{array},$$

$$\alpha \otimes \mu(a) = \begin{array}{c} \begin{array}{ccccc} q_0 & q_1 & q_2 & q_3 & q_4 \end{array} \\ \begin{bmatrix} -\infty & 1 & 1 & -\infty & -\infty \end{bmatrix} \end{array},$$

$$\mu(b) = \begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} \begin{array}{c} \begin{array}{ccccc} q_0 & q_1 & q_2 & q_3 & q_4 \end{array} \\ \begin{bmatrix} -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 2 & -\infty \\ -\infty & -\infty & -\infty & 1 & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty \end{bmatrix} \end{array},$$

$$\mu(a) \otimes \mu(b) = \begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} \begin{array}{c} \begin{array}{ccccc} q_0 & q_1 & q_2 & q_3 & q_4 \end{array} \\ \begin{bmatrix} -\infty & -\infty & -\infty & 3 & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty \end{bmatrix} \end{array},$$

$$\alpha \otimes \mu(a) \otimes \mu(b) = \begin{array}{c} \begin{array}{ccccc} q_0 & q_1 & q_2 & q_3 & q_4 \end{array} \\ \begin{bmatrix} -\infty & -\infty & -\infty & 3 & -\infty \end{bmatrix} \end{array},$$

hence $\sigma(\pi_2) = (a,1)(b,3)$. □

Index