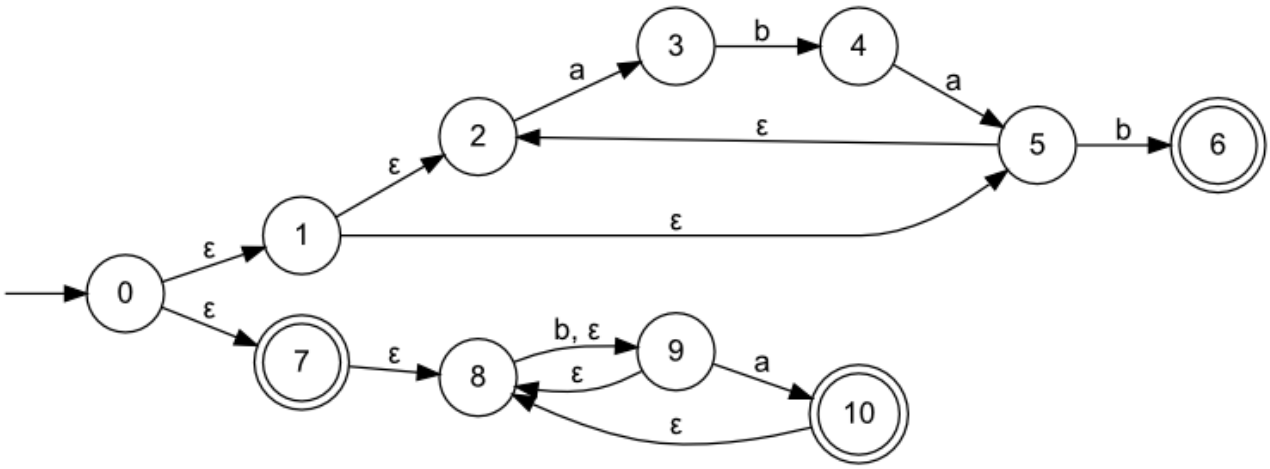
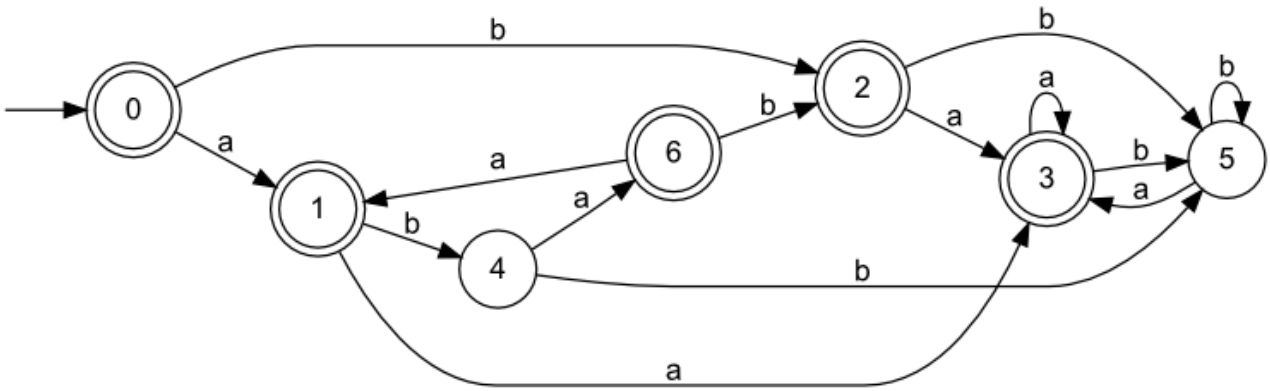
COMP2022 Formal Languages and Logic Assignment 1

1. Devise an NFA accepting *(aba)*b* | *(b*a)** and draw it



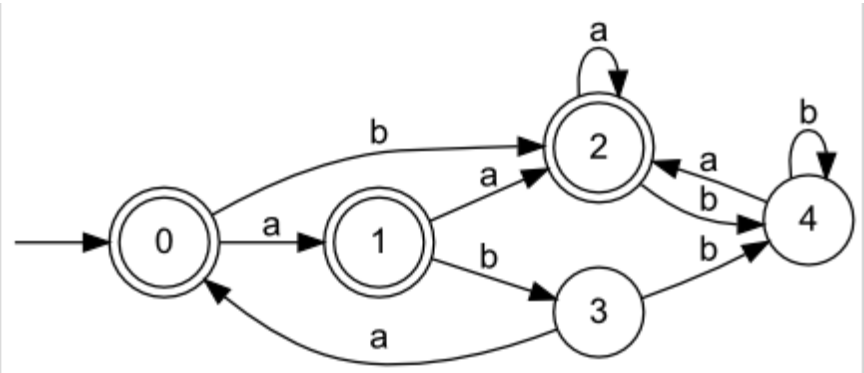
2. Transform the NFA into a DFA

DFA	NFA	a	b
0	{0,1,2,5,7,8,9}	{3,8,9,10}	{6,8,9}
1	{3,8,9,10}	{8,9,10}	{4,8,9}
2	{6,8,9}	{8,9,10}	{8,9}
3	{8,9,10}	{8,9,10}	{8,9}
4	{4,8,9}	{2,5,8,9,10}	{8,9}
5	{8,9}	{8,9,10}	{8,9}
6	{2,5,8,9,10}	{3,8,9,10}	{6,8,9}



3. Minimise the DFA and draw it

0							
1	X						
2	X	X					
3	X	X					
4	X	X	X	X			
5	X	X	X	X	X		
6		X	X	X	X	X	
	0	**1**	**2**	**3**	**4**	**5**	**6**



4. Implement a program that simulates the computation of your minimal DFA

Consult the README for a list of features, how to compile and test, usage of the program, description of each file and how testing is conducted.

```
/* dfa: Run and visualise DFAs defined by the DOT language. */
#define _GNU_SOURCE
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sysexits.h>
#include <err.h>
#include <getopt.h>
#include <stdbool.h>
#include "graphviz/cgraph.h"
#include "graphviz/gvc.h"

bool test;
char *usages = "usage: dfa [-tv?] file\n";

void init(Agraph_t *g);
void vis(Agraph_t *g);
```

```
Agnode_t *delta(Agraph_t *g, Agnode_t *n, char c);
void run(Agraph_t *g);
void usage(void);

/* Try to find the delta transition for a given state and character. Returns
 * the next state if found, else returns NULL. */
Agnode_t *delta(Agraph_t *g, Agnode_t *n, char c)
{
	Agedge_t *e;

	for (e = agfstout(g, n); e; e = agnxtout(g, e))
		if (strchr(agget(e, "label"), c))
			return e->node;
	return NULL;
}

/* Run the DFA across strings provided from stdin. */
void run(Agraph_t *g)
{
	char c, *alphabet, *prompt, *str = NULL;
	Agnode_t *n, *m, *fst;
	size_t linecap;
	ssize_t len;

	/* Check for an alphabet and display it to the user. */
	alphabet = agget(g, "alphabet");
	if (!alphabet)
		errx(EX_DATAERR, "error: no alphabet");
	if (!test)
		printf("alphabet = {%s}\n", alphabet);

	/* Get the start state and find the state it points to. */
	fst = agnode(g, "start", FALSE);
	n = agfstout(g, fst)->node;

	if (!test)
		prompt="> ";
	else
		prompt="";

	/* Continously prompt the user for strings to run the DFA with until
	 * `exit' is provided or the end-of-file is reached. */
	while (printf("%s", prompt), (len = getline(&str, &linecap, stdin)) > 0) {
		/* Remove the newline character. */
		str[--len] = '\0';

		/* Check for `exit' from user */
		if (strcmp(str, "exit") == 0)
			exit(0);

		/* Check that all characters in the provided string are from
		 * the given alphabet. */
		alphabet = agget(g, "alphabet");
		for (int i = 0; i < len; i++)
			if (!strchr(alphabet, str[i])) {
				warnx("warning: %c isn't in the alphabet",
					str[i]);
				goto input_exception;
			}
```

```
		/* Run the DFA across the string. */
		for (int i = 0; i < len; i++) {
			m = delta(g, n, str[i]);

			/* There was no delta transition, warn the user and
			 * prompt for another string. */
			if (!m) {
				warnx("warning: no delta(%s, %c)", agnameof(n),
						str[i]);
				goto input_exception;
			}

			if (!test) {
				/* Scanned intput */
				for (int j = 0; j < i; j++)
					putchar(str[j]);

				/* Transition; it isn't safe to use `agnameof'
				 * twice in the same statement as a temporary
				 * buffer is used between calls. */
				printf("\t%s -- %c --> ", agnameof(n), str[i]);
				printf("%s\t", agnameof(m));

				/* Unscanned input */
				for (int j = i + 1; j < len; j++)
					putchar(str[j]);
				putchar('\n');
			}

			n = m;
		}

		if (!strcmp(agget(n, "shape"), "doublecircle"))
			if (!test)
				printf("Accepted\n");
			else
				printf("%s\n", str);
		else
			if (!test)
				printf("Declined\n");
			else
				;

input_exception:
		n = agfstout(g, fst)->node;
	}
}

/* Initialise attributes for the different types of nodes to reduce the
 * verboseness of the DFA definition files. */
void init(Agraph_t *g)
{
	Agnode_t *n;

	/* Make the following attributes available and provide a default. */
	agattr(g, AGRAPH, "rankdir", "LR");
	agattr(g, AGNODE, "shape", "circle");
	agattr(g, AGNODE, "width", ".5");
```

```
	agattr(g, AGNODE, "height", ".5");
	agattr(g, AGNODE, "label", "");

	/* Set the label for all nodes except the `start' node. */
	for (n = agfstnode(g); n; n = agnxtnode(g, n))
		if (strcmp("start", agnameof(n)) != 0)
			agset(n, "label", agnameof(n));

	/* The accept states is a comma and/or space seperated set of states. */
	char *sep = ", ";
	char *accept = agget(g, "accept");
	if (!accept)
		errx(EX_DATAERR, "error: no accept states specified");

	/* Make the final states' shape a double circle. */
	for (char *q = strtok(accept, sep); q; q = strtok(NULL, sep)) {
		n = agnode(g, q, FALSE);
		if (!n)
			errx(EX_DATAERR, "error: accept: no state %s", q);
		agset(n, "shape", "doublecircle");
	}

	/* Set start state attributes; the node shouldn't be displayed. */
	n = agnode(g, "start", FALSE);
	if (!n)
		errx(EX_DATAERR, "error: no start state");
	agset(n, "shape", "none");
	agset(n, "width", "0");
	agset(n, "height", "0");
}

/* Draw the graph as an SVG using the DOT layout engine. */
void vis(Agraph_t *g)
{
	GVC_t *gvc = gvContext();
	gvLayout(gvc, g, "dot");
	gvRender(gvc, g, "svg", stdout);
	gvFreeLayout(gvc, g);
	gvFreeContext(gvc);
}

void usage()
{
	fprintf(stderr, "%s", usages);
	exit(1);
}

int main(int argc, char *argv[])
{
	Agraph_t *g;
	FILE *fd;
	char ch;
	bool v, t;

	while ((ch = getopt(argc, argv, "tv?")) != -1) {
		switch (ch) {
		case 'v':
			v = true;
			break;
```

```
                case 't':
                        test = true;
                        break;
                case '?':
                        usage();
                }
        }

        if (optind < argc)
                fd = fopen(argv[optind], "r");
        if (!fd) {
                warnx("error: no input file");
                usage();
        }
        g = agread(fd, NULL);
        fclose(fd);

        init(g);
        if (v)
                vis(g);
        else
                run(g);

        agclose(g);
        return 0;
}
```