Laporan Tugas I
IF2220 Teori Bahasa Formal dan Automata

Jonathan Christopher / 13515001 / K-01

September 15, 2016

1 Deskripsi Persoalan

Deskripsi persoalana lalalal

2 *Deterministic Finite Automata*

3 Daftar Asumsi

4 *Source Code*

4.1 dfa.h

```h
#ifndef DFA_H
#define DFA_H

#define SYMBOL_COUNT_LIMIT 255
#define SYMBOL_SCAN_STR "%255s"

#define STATE_NAME_LENGTH_LIMIT 1023
#define STATE_SCAN_STR "%1023s"

typedef struct {

    /* Jumlah simbol (ukuran alfabet) dan array berisi karakter-karaketer simbol */
    int n_symbols;
    char *symbols;

    /* Jumlah state dan array berisi nama-nama state */
    int n_states;
    char **states;

    /* Jumlah final state dan array berisi indeks-indeks final state */
    int n_final_states;
    int *final_state_indexes;

    /* Indeks state awal */
    int start_state_index;

    /* Array 2-dimensi berisi transition table (baris: indeks state, kolom: indeks
        simbol, isi: indeks state tujuan */
    int *transition_table;

} dfa;

/* Load spesifikasi DFA dari file dengan path yang diberikan.
 * Menghasilkan DFA tersebut jika berhasil, dan DFA dengan start_state -1 jika gagal.
 * Format file DFA ada di file README.md */
dfa load_dfa(const char *path);

/* Deallocate memori yang digunakan oleh DFA */
void unload_dfa(dfa *automaton);

/* Menghasilkan indeks state tujuan dari DFA jika diberikan keadaan indeks state dan
    indeks symbol input */
int delta(dfa automaton, int current_state_index, int symbol_index);

/* Menghasilkan nomor state dari nama state yang diberikan dan -1 jika nama state
    tersebut tidak ditemukan */
int get_state_index(dfa automaton, const char *state);
```

```c
/* Menghasilkan nomor symbol dari karakter symbol yang diberikan dan -1 jika nama
    symbol tersebut tidak ditemukan */
int get_symbol_index(dfa automaton, char symbol);

/* Menghasilkan 1 jika nomor state yang diberikan adalah final state, 0 jika tidak
    atau state tidak ditemukan */
unsigned char is_final_state(dfa automaton, int current_state_index);

#endif
```

4.2 dfa.c

```c
#include "dfa.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Load spesifikasi DFA dari file dengan path yang diberikan, dengan asumsi format
    file valid.
 * Menghasilkan DFA tersebut jika berhasil, dan DFA dengan start_state -1 jika gagal.
 * Format file DFA ada di file README.md */
dfa load_dfa(const char *path) {

    dfa new_dfa;
    char symbols_buf[SYMBOL_COUNT_LIMIT + 1];
    char state_buf[STATE_NAME_LENGTH_LIMIT + 1];
    int state_len;
    int i, j;
    FILE *fin;

    /* Buka file */
    fin = fopen(path, "r");
    if (fin == NULL) {
        /* Failed to open file */
        new_dfa.start_state_index = -1;
        return new_dfa;
    }

    /* Baca simbol, asumsi simbol unik, dalam bentuk string tanpa whitespace */
    fscanf(fin, SYMBOL_SCAN_STR, symbols_buf);
    new_dfa.n_symbols = strlen(symbols_buf);
    new_dfa.symbols = malloc(sizeof(char) * new_dfa.n_symbols);
    strncpy(new_dfa.symbols, symbols_buf, new_dfa.n_symbols); /* new_dfa.symbols
        tidak null-terminated */

    /* Baca jumlah state dan nama-nama state */
    fscanf(fin, "%d", &new_dfa.n_states);
    new_dfa.states = malloc(sizeof(char *) * new_dfa.n_states);
    for (i = 0; i < new_dfa.n_states; i++) {
        fscanf(fin, STATE_SCAN_STR, state_buf);
        state_len = strlen(state_buf);
        new_dfa.states[i] = malloc(sizeof(char) * (state_len + 1));
        strncpy(new_dfa.states[i], state_buf, state_len + 1);
    }

    /* Baca jumlah final state dan nama-nama final state */
    fscanf(fin, "%d", &new_dfa.n_final_states);
    new_dfa.final_state_indexes = malloc(sizeof(int) * new_dfa.n_final_states);
    for (i = 0; i < new_dfa.n_final_states; i++) {
```

```
        fscanf(fin, STATE_SCAN_STR, state_buf);
        new_dfa.final_state_indexes[i] = get_state_index(new_dfa, state_buf);
    }

    /* Baca nama state awal */
    fscanf(fin, STATE_SCAN_STR, state_buf);
    new_dfa.start_state_index = get_state_index(new_dfa, state_buf);

    /* Baca transition table */
    new_dfa.transition_table = malloc(sizeof(int) * new_dfa.n_states * new_dfa.
        n_symbols);
    for (i = 0; i < new_dfa.n_states; i++) {
        for (j = 0; j < new_dfa.n_symbols; j++) {
            fscanf(fin, STATE_SCAN_STR, state_buf);
            new_dfa.transition_table[i*new_dfa.n_symbols + j] = get_state_index(
                new_dfa, state_buf);
        }
    }

    fclose(fin);
    return new_dfa;
}

/* Deallocate memori yang digunakan oleh DFA */
void unload_dfa(dfa *automaton) {
    int i;

    free((*automaton).symbols);
    (*automaton).n_symbols = 0;

    for (i = 0; i < (*automaton).n_states; i++) {
        free((*automaton).states[i]);
    }
    free((*automaton).states);
    (*automaton).n_states = 0;

    free((*automaton).final_state_indexes);
    (*automaton).n_final_states = 0;

    (*automaton).start_state_index = -1;

    free((*automaton).transition_table);
}

/* Menghasilkan indeks state tujuan dari DFA jika diberikan keadaan indeks state dan
    indeks symbol input */
int delta(dfa automaton, int current_state_index, int symbol_index) {
    return automaton.transition_table[current_state_index*automaton.n_symbols +
        symbol_index];
}

/* Menghasilkan nomor state dari nama state yang diberikan dan -1 jika nama state
    tersebut tidak ditemukan */
int get_state_index(dfa automaton, const char *state) {
    int i = 0;
    unsigned char found = 0;
    while (i < automaton.n_states && !found) {
        if (strcmp(automaton.states[i], state) == 0) {
            found = 1;
        } else {
            i++;
        }
```

```
    }
    return found ? i : -1;
}

/* Menghasilkan nomor symbol dari karakter symbol yang diberikan dan -1 jika nama
    symbol tersebut tidak ditemukan */
int get_symbol_index(dfa automaton, char symbol) {
    int i = 0;
    unsigned char found = 0;
    while (i < automaton.n_symbols && !found) {
        if (automaton.symbols[i] == symbol) {
            found = 1;
        } else {
            i++;
        }
    }
    return found ? i : -1;
}

/* Menghasilkan 1 jika nomor state yang diberikan adalah final state, 0 jika tidak
    atau state tidak ditemukan */
unsigned char is_final_state(dfa automaton, int current_state_index) {
    int i = 0;
    unsigned char found = 0;
    while (i < automaton.n_final_states && !found) {
        if (automaton.final_state_indexes[i] == current_state_index) {
            found = 1;
        } else {
            i++;
        }
    }
    return found;
}
```

4.3 main.c

```
#include "dfa.h"
#include <stdio.h>
#include <string.h>

#define MAX_FILENAME_LENGTH 255
#define FILENAME_SCAN_STR "%255s"
#define MAX_INPUT_STRING_LENGTH 1023
#define INPUT_STRING_SCAN_STR "%1023s"
#define MAX_REPEAT_LENGTH 100
#define REPEAT_SCAN_STR "%100s"

int main() {

    dfa automaton;
    int i, input_len;
    unsigned char repeat_input = 0;

    /* Cetak header */
    printf("\nDFA\n===\n\n");
    printf("NIM       : 13515001\nNama      : Jonathan Christopher\nKelas     : K-01,
        TBFO IF2220\nTanggal   : 12 September 2016\nTopik     : DFA\nDeskripsi : 
        Program sederhana untuk mengecek apakah suatu string diterima oleh DFA 
        tertentu.\n");
    printf("\n");
```

```
/* Baca nama file DFA, ulangi hingga file dapat dibuka */
unsigned char load_file_ok = 0;
do {
    char path_buf[MAX_FILENAME_LENGTH + 1];
    printf("Path ke file DFA (untuk tugas TBFO, gunakan dfa/marble.dfa): ");
    scanf(FILENAME_SCAN_STR, path_buf);
    /* Load file DFA */
    automaton = load_dfa((char *) path_buf);
    if (automaton.start_state_index == -1) {
        printf("Gagal membaca file DFA.\n");
    } else {
        load_file_ok = 1;
    }
} while (!load_file_ok);

/* Cetak alfabet (simbol-simbol yang diterima) untuk bantuan */
printf("DFA berhasil dibaca.\n");
printf("Simbol-simbol yang diterima: ");
for (i = 0; i < automaton.n_symbols; i++) {
    printf("%c ", automaton.symbols[i]);
}
printf("\n");

do {

    /* Baca input string */
    char input_buf[MAX_INPUT_STRING_LENGTH + 1];
    printf("\n");
    printf("Masukkan input string: ");
    scanf(INPUT_STRING_SCAN_STR, input_buf);
    input_len = strlen(input_buf);

    /* Jalankan DFA */
    int current_state_index = automaton.start_state_index;
    int next_symbol_index;
    printf("%s", automaton.states[current_state_index]);
    for (i = 0; i < input_len; i++) {
        next_symbol_index = get_symbol_index(automaton, input_buf[i]);
        /* Jika terbaca sebuah simbol yang tidak dikenali, maka tidak akan
            berpindah state */
        if (next_symbol_index >= 0) {
            current_state_index = delta(automaton, current_state_index,
                next_symbol_index);
            printf(" -> %s", automaton.states[current_state_index]);
        }
    }
    printf("\n");
    printf("String diterima? %s\n", is_final_state(automaton, current_state_index
        ) ? "Ya (berakhir di final state)" : "Tidak (tidak berakhir di final
        state)");

    /* Tanya apakah ingin mengulang */
    char repeat_buf[MAX_REPEAT_LENGTH + 1];
    printf("Coba lagi? (y/n): ");
    scanf(REPEAT_SCAN_STR, repeat_buf);
    repeat_input = (strlen(repeat_buf) > 0 && (repeat_buf[0] == 'y' || repeat_buf
        [0] == 'Y'));

} while (repeat_input);

unload_dfa(&automaton);
```

```
    return 0;
}
```

4.4 marble.dfa

```
AB

13
lllc
rllc
lrrc
lrlc
rrrc
lrld
rrlc
llrd
rrld
llld
rlrd
rlrc
rlld

6
lrld
llrd
rrld
llld
rlrd
rlld

lllc

rllc lrrc
lrlc rrrc
rrrc lrld
rrlc llrd
llrd rrld
rrlc llrd
llld rlrd
rlrc llld
llld rlrd
rllc lrrc
lrrc rlld
lrrc rlld
lrlc rrrc
```

5 Contoh Interaksi dengan Program