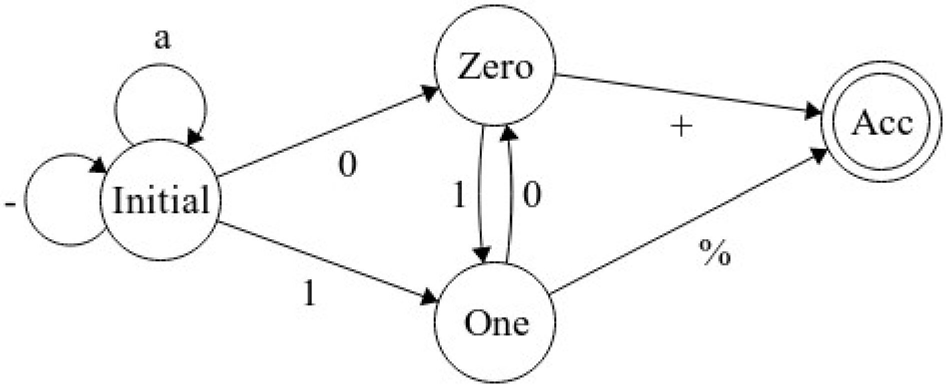
TP N°1

Considérons un langage imaginaire qui accepte *a* et - au début du mot, puis après 1 ou 0 n'accepte que des uns et des zéros. De plus, pour accepter la phrase, elle doit se terminer par + si elle était dans l'état zéro ou % si elle était dans l'état un. Pour clarifier, les automates du langage sont décrits dans l'image suivante :



Le code suivant crée et décrit cet automate :

```
DFA *automata = create_automata();

  DFA_State *initial_state = create_state(INITIAL_STATE, false, automata);
  DFA_State *zero_state = create_state(ZERO_STATE, false, automata);
  DFA_State *one_state = create_state(ONE_STATE, false, automata);
  DFA_State *accept_state = create_state(ACCEPT_STATE, true, automata);

  generate_transitions("a-", initial_state, initial_state);
  generate_transitions("0", initial_state, zero_state);
  generate_transitions("1", initial_state, one_state);
  generate_transitions("0", one_state, zero_state);
  generate_transitions("1", zero_state, one_state);
  generate_transitions("+", zero_state, accept_state);
  generate_transitions("%", one_state, accept_state);

  automata->initial_state = initial_state;
```

Pour utiliser le DFA, vous devez d'abord créer les automates via la fonction **create_automata().** Ensuite, vous pouvez créer des états et ajouter des transitions via **create_state(int id, bool is_accept_state?, automates DFA)** et **create_transition(char trigger, DFA_State origin, DFA_State destination)** respectivement.

Vous pouvez également créer plusieurs transitions via **generate_transitions(string, DFA_State origin, DFA_State destination)**. Cette fonction parcourt tous les caractères de la chaîne et génère une transition pour chacun.

Écrire un programme en langage C qui peut reconnaitre de les mots de ce langage

DFA.h

```
#ifndef DETERMINISTIC_FINITE_AUTOMATA_H
#define DETERMINISTIC_FINITE_AUTOMATA_H

#include <stdbool.h>

/* Structs to represent State, Transition and the automata itself */
typedef struct DETERMINISTIC_FINITE_AUTOMATA DFA;
typedef struct DETERMINISTIC_FINITE_AUTOMATA_STATE DFA_State;
typedef struct DETERMINISTIC_FINITE_AUTOMATA_TRANSITION DFA_Transition;

/* Define an automata transition */
struct DETERMINISTIC_FINITE_AUTOMATA_TRANSITION
{
    DFA_State *origin_state;
    DFA_State *destination_state;
    char trigger_value;     // The value that triggers the transition
};

/* Define an automata state */
struct DETERMINISTIC_FINITE_AUTOMATA_STATE
{
    int state_identifier;   // Integer number to help end user identify the state
    bool accept_state;      // If this state is an accept state or not
    int transitions_count;
    DFA_Transition **transitions;
};

/* Define the automata */
struct DETERMINISTIC_FINITE_AUTOMATA
```

```
{
   int states_count;
   DFA_State **states;
   DFA_State *current_state;
   DFA_State *initial_state;
};

/* ====== Functions ====== */
/* Automata state manipulation */
bool init_automata (DFA *automata);
bool update_automata (char ch, DFA *automata);

/* Automata abstraction of state updates */
bool belongs_to_language (char *string, DFA *automata);

/* Alloc abstaction */
DFA_State* create_state (int state_identifier, bool accept_state, DFA *automata);
DFA_Transition* create_transition (char trigger_value, DFA_State *origin_state,
DFA_State *destination_state);
DFA* create_automata ();

/* Allow creation of several transitions simply */
bool generate_transitions(char *string, DFA_State *origin_state, DFA_State
*destination_state);

/* Dealloc abstraction */
bool free_automata (DFA *automata);

/* ====== DEBUG FUNCTIONS ====== */
void describe_automata (DFA *automata);
```

```
#endif  // DETERMINISTIC_FINITE_AUTOMATA_H
```

DFA.c

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

bool init_automata (DFA *automata)
{
  if (!automata->initial_state) {
    printf("[-] Error: Initial state of automata not set.\n");
    exit(1);
  }

  return (automata->current_state = automata->initial_state);
}

bool update_automata (char ch, DFA *automata)
{
  DFA_State *state = automata->current_state;
  bool made_transition = false;

  for (int j = 0; j < state->transitions_count; j++) {
    if (state->transitions[j]->trigger_value == ch) {
      automata->current_state = state->transitions[j]->destination_state;
      made_transition = true;
      break;  // Finite automata does not have 'choices'. If trigger_value matches, then the automata must not have another transition with the same value.
    }
  }

  if (!made_transition) { // Automata will always execute a transition if value is accepted. Even if the transition result in the same stage.
    return false;
  }

  return true;
}

bool belongs_to_language (char *string, DFA *automata)
{
  if (strlen(string) == 0) {
    printf("[-] Error: Empty string sent to analyse in belongs_to_language.\n");
    exit(1);
  }

  init_automata(automata);
  for (int i = 0; i < strlen(string); i++) {
    if (!update_automata(string[i], automata)) {
      return false;
    }
  }

  return automata->current_state->accept_state;
}

bool set_transition_to_state (DFA_Transition **transition, DFA_State *origin_state)
{
  DFA_Transition **tmp = realloc(origin_state->transitions, (origin_state->transitions_count + 1) * sizeof(DFA_Transition));
  if (tmp == NULL) {
    printf("[-] Error during reallocation of state transitions. Trying again.\n");
    tmp = realloc(origin_state->transitions, (origin_state->transitions_count + 1) *
sizeof(DFA_Transition));

    if (tmp == NULL) {
      printf("[-] Error during reallocation of state transitions.\n");
      exit(1);
    }
  }

  origin_state->transitions = tmp;

  origin_state->transitions[origin_state->transitions_count] = *transition;    // Add new
transitions to array of transitions
  origin_state->transitions_count += 1;

  return true;
}

bool set_state_to_automata (DFA_State **state, DFA *automata)
{
  automata->states = realloc(automata->states, (automata->states_count + 1) *
sizeof(DFA_State));

  if (!automata->states) {
    printf("[-] Error during reallocation of automata states.\n");
    exit(1);
  }
  automata->states[automata->states_count] = *state;  // Add new state to the final of the array
  automata->states_count += 1;

  return true;
}

DFA_State* create_state (int state_identifier, bool accept_state, DFA *automata)
{
  DFA_State *state = malloc(sizeof(DFA_State));
  if (!state) {
    printf("[-] Error during allocation of state.\n");
    exit(1);
  }

  state->accept_state = accept_state;
  state->transitions_count = 0;
  state->state_identifier = state_identifier;
  state->transitions = NULL;

  set_state_to_automata(&state, automata);

  return state;
}

DFA_Transition* create_transition (char trigger_value, DFA_State *origin_state, DFA_State *destination_state)
{
  DFA_Transition *transition = malloc(sizeof(DFA_Transition));
  if (!transition) {
    printf("[-] Error during allocation of transition.\n");
    exit(1);
  }

  transition->trigger_value = trigger_value;
  transition->origin_state = origin_state;
  transition->destination_state = destination_state;

  set_transition_to_state(&transition, origin_state);

  return transition;
}

bool generate_transitions (char *string, DFA_State *origin_state, DFA_State
*destination_state)
{
  for (int i = 0; i < strlen(string); i++) {
    if (!create_transition(string[i], origin_state, destination_state)) {
      return false;
    }
  }

  return true;
}

DFA* create_automata ()
{
  DFA *automata = malloc(sizeof(DFA));
  if (!automata) {
    printf("[-] Error during allocation of automata.\n");
    exit(1);
  }
  automata->states_count = 0;
  automata->initial_state = NULL;
  automata->states = NULL;

  return automata;
}

bool free_automata (DFA *automata)
{
  for (int i = 0; i < automata->states_count; i++) {
    for (int j = 0; j < automata->states[i]->transitions_count; j++) {
      free(automata->states[i]->transitions[j]);
    }

    free(automata->states[i]);
  }

  free(automata);
  return true;
}

/* ========== DEBUG FUNCTIONS ========== */
void describe_transition (DFA_Transition *transition)
{
  printf("------ Origin state: %p\n", transition->origin_state);
  printf("------ Destination state: %p\n", transition->destination_state);
  printf("------ Trigger value: %c\n", transition->trigger_value);
}

void describe_state (DFA_State *state)
```

```
{
  printf("--- Accept state: %d\n", state->accept_state);

  if (state->transitions_count > 0) {
    printf("--:: Transitions dump (%d):\n", state->transitions_count);
    for (int i = 0; i < state->transitions_count; i++) {
      printf("------ Transition %d\n", i);
      describe_transition(state->transitions[i]);
    }
  } else {
    printf("--:: State has no transitions to dump.\n");
  }
}

void describe_automata (DFA *automata)
{
  printf("Automata dump:\n");

  if (automata->states_count > 0) {
    printf(":: States dump (%d):\n", automata->states_count);
    for (int i = 0; i < automata->states_count; i++) {
      printf(":: State %d\n", i);
      describe_state(automata->states[i]);
    }
  } else {
    printf("--- Automata has no states to dump.\n");
  }
}
```

Écrire le programme principale main.c pour vérifie ces 2 expression régulière

- **a-aaaa--0101%**
- **100101011**