Assignment I (15 pts)

Burak Ekici

Assigned : November the 3rd, 23h55
Due : November the 17th, 23h55

1 Definitions

Deterministic finite state automaton (DFA) is a finite state machine that executes, and accepts or rejects a given string by walking through a sequence of states governed by the input string.

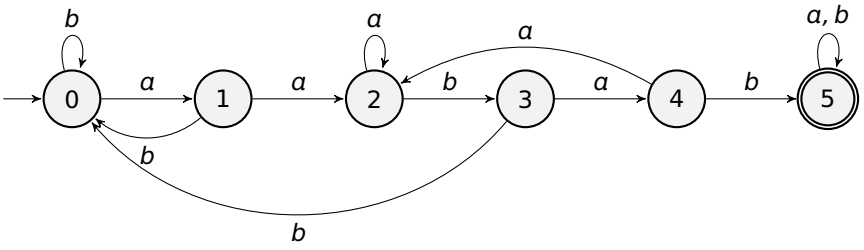
Formally speaking, a DFA is characterized by a quintuple (5-tuple) $M = (Q, \Sigma, \delta, s, F)$ where

1. Q : finite set of states
2. Σ : input alphabet
3. $\delta : Q \to \Sigma \to Q$: transition function
4. $s \in Q$: start state
5. $F \subseteq Q$: final (accept) states.

Intuitively, a state of a DFA amounts to the snapshot of machine's functionality describing its instantaneous behavior frozen in time. Composing every single state of an automaton then answers the question of "what does it do?". The question tagged "how" is replied by the (total) transition function δ, clarifying the working principle or procedure that the machine employs. A DFA computes with strings constructed over the input alphabet Σ which simply is a collection of characters (letters or symbols). Every DFA has a single staring state from which the computations get initiated, and embodies a set of final states that separate accepted and rejected strings. E.g., starting from the initial state if a string $x \in \Sigma^*$ advances the machine to one of the final states, then the string x is said to be accepted, and rejected otherwise.

Note.
Note that Σ^* (aka Kleene closure of Σ) refers to the set of all possible strings that could be constructed over the alphabet Σ.

Example 1.1. A DFA $M_1 := \big(\{0,1,2,3,4,5\}, \{a,b\}, \delta, 0, \{5\}\big)$, with the transition function δ depicted below, accepts the set of strings that contain "$aabab$" as sub-string.



One can of course tabulate the function δ as follows:

δ	a	b
0	1	0
1	2	0
2	2	3
3	4	0
4	2	5
5	5	5

Note.
By convention, the initial state of a DFA is depicted by an incoming arrow "$\rightarrow$" coming from none of the states, and final states are pictured by a pair of circles placed one into the other. In the above example, for instance, the initial state s of the DFA M_1 is 0 while the set of final states F is $\{5\}$.

Definition 1.1. For an arbitrarily given DFA $M = \big(Q, \Sigma, \delta, s, F\big)$, *multi-step transition function* $\hat{\delta}\colon Q \rightarrow \Sigma^* \rightarrow Q$ inputs a state along with a string, and outputs a state handled by chaining single step transitions together.

Formally, speaking $\hat{\delta}$ is recursively defined on the length of the input string as follows:

$$\hat{\delta}(q,\varepsilon) = q \qquad \hat{\delta}(q,ax) = \hat{\delta}(\delta(q,a),x) \qquad \forall q \in Q, a \in \Sigma \text{ and } x \in \Sigma^*$$

Definition 1.2. The acceptance/rejection criterion for any DFA $M = \big(Q, \Sigma, \delta, s, F\big)$ is defined employing the multi-step transition:

- string $x \in \Sigma^*$ is accepted by M if $\hat{\delta}(s,x) \in F$
- string $x \in \Sigma^*$ is rejected by M if $\hat{\delta}(s,x) \notin F$

For instance, the machine M_1 specified in Example 1.1 accepts the string "$baaababa$", rejecting "$bbbabaaaba$".

2 DFA in Haskell

Haskell serves with sufficient machinery to develop deterministic finite state automata therein. We formalize DFA employing a Haskell record as listed in what follows.

```haskell
type State = Int

data DFA =
    DFA
    {   dfa_state    :: Set State,              -- Q
        dfa_alphabet :: Set Char,               -- Σ
        dfa_sigma    :: State -> Char -> State, -- δ : Q → Σ → Q
        dfa_start    :: State,                  -- s ∈ Q
        dfa_finals   :: Set State               -- F ⊆ Q
    }
```

A record is simply a collection of data fields. In more formal terms, records are inductive data types with a single constructor in Haskell. As a direct implication of this fact, following implementation of the `DFA` type

```haskell
data DFA where
    DFA :: Set State -> Set Char -> (State -> Char -> State) -> State -> Set State -> DFA
```

is equivalent to the earlier one employing a record.

Attention.

We embark on the former implementation technique here as it serves relatively easier access to DFA fields. For instance, given a DFA `d :: DFA`, it suffices to type `state_set d` to handle the state set; similarly, typing `dfa_finals d` returns the final states of `d`. The same extraction is definitely possible with the latter approach but always necessitates the destruction of the DFA `d` via **`case d of`** construct, and might be a bit more cumbersome.

2.1 The `Set a` Type in Haskell

The Haskell built-in type `Set a` implements the finite set of type `a` inhabitants. It lets one uniquely store elements and benefit from efficient set theoretic operations as itemized below.

`toList`	`:: Set a -> [a]`	enumerating set members into a list
`fromList`	`:: Ord a => [a] -> Set a`	forming a set out of a given list
`member`	`:: Ord a => a -> Set a -> Bool`	membership checking
`union`	`:: Ord a => Set a -> Set a -> Set a`	taking union of a pair of sets
`intersection`	`:: Ord a => Set a -> Set a -> Set a`	taking intersection of a pair of sets
`difference`	`:: Ord a => Set a -> Set a -> Set a`	taking difference of a pair of sets

Please refer to this web-page for further useful functions and instructions.

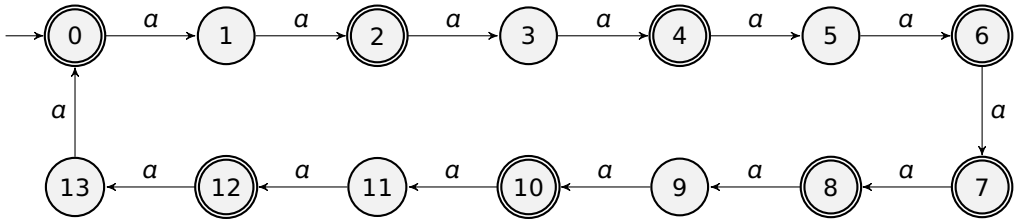
Hint.

Try not to destruct `Set a` instances. Namely, it is not suggested to employ the `case-of` construct over instances of the `Set a` type. Instead, make use of above stated functions. In a scenario where it is inevitable to destruct `Set a` instances, first enumerate them into lists (with `toList` function) and only then apply the `case-of` construct. Also, benefit from the `fromList` function to form `Set a` instances out of `[a]` instances.

3 Tasks

1. (**7 pts**) For an arbitrary DFA instance `d :: DFA`, implement in Haskell
   - (**5 pts**) the multi-step transition function `dfa_multiStep :: DFA -> State -> String -> State`;
   - (**2 pts**) the acceptance criterion `dfa_acceptance :: DFA -> String -> Bool`

   in the way they are stated in Definitions 1.1 and 1.2.
2. (**4 pts**) Implement and simulate, via `dfa_acceptance` function, the DFA M_1 of Example 1.1 in Haskell. E.g., `dfa_acceptance dfa1 "baaababa"` returns `True` while `dfa_acceptance dfa1 "bbbabaaaba"` returns `False`.
3. (**4 pts**) Let $M_2 := \big(\{0,1,2,3,4,5,6,7,8,9,10\}, \{a\}, \delta, 0, \{0,2,4,6,7,8,10,12\}\big)$ a DFA, with the transition function δ depicted below, that accepts stings of length divisible either by 2 or by 7.



   Implement and simulate, via `dfa_acceptance` function, the DFA M_2 in Haskell. E.g., `dfa_acceptance dfa2 "aaaaaaa"` returns `True` while `dfa_acceptance dfa2 "aaaaa"` returns `False`.

Nota Bene (in general).

1. receive support from helper functions if needed;
2. the attached file (`DFA.hs`) could be a good starting point;
3. do not remove or modify the line `{-#LANGUAGE GADTs#-}`, and those reserved for importing external libraries contained in the file `DFA.hs`.

Important Notice.

- Collaboration is strictly and positively prohibited; lowers your score to 0 if detected.
- Any submission after 23h55 on November the 17^{th} will NOT be accepted. Please beware and respect the deadline!
- Implement your code within a file named `yourname_surname.hs`, and submit it either in the raw form as it is or in the ZIP compressed form. Do not RAR files.