NOTES 8: FIXING THE POINTS

Cameron Moy

Tuesday June 26th

1 Into the woods

In the regular expression interpreter project, you are asked to write several functions relating to NFAs. First let's define what an NFA is formally. Understanding this basic definition is crucial to writing programs manipulating them.

Definition. A **non-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 a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

We translate this mathematical definition into OCaml types.

```
type ('q, 's) transition = 'q * 's option * 'q
type ('q, 's) nfa = {
    qs : 'q list;
    sigma : 's list;
```

```
    delta : ('q, 's) transition list;
    q0 : 'q;
    fs : 'q list;
}
```

Here we depart from the mathematical definition in a number of ways.

1. We use `list` instead of sets, although you should be modifying these lists via your set functions from P2A.
2. ε transitions are denoted with `None` and non-ε transitions are denoted with `Some`.
3. Instead of a transition function we have an associative list. This makes modifications easier to handle.

2 The fantastic four

You're required to write four functions over NFAs.

1. `move (m : ('q, 's) nfa_t) (l : 'q list) (c : 's option) : 'q list`.
   Moves forward from all the states in `l` on symbol `c`.
2. `e_closure (m : ('q, 's) nfa_t) (l : 'q list) : 'q list`.
   Yields the list of states reachable in zero or more epsilon transitions from the states in `l`.
3. `accept (m : ('q, char) nfa_t) (s : string) : bool`.
   Returns whether `s` is accepted by NFA `m`.
4. `nfa_to_dfa (m : ('q, 's) nfa_t) : ('q list, 's) nfa_t`.
   Converts NFA `m` to an equivalent DFA.

The `move` and `accept` functions typically don't give people too much trouble. However, `e_closure` and `nfa_to_dfa` usually do. Let's take a look at a general technique and apply it to help us solve both problems.

3 The fixed point

Definition. A **fixed point** (abbreviated fixpoint) of function $f : X \to X$ is an element $a \in X$ such that $f(a) = a$.

Example. Let $f(x) = x^2 - 3x + 4$. Notice that $f(2) = 2$. Therefore, 2 is a fixpoint of f. ❑

This is all well and good, but how does one actually compute a fixpoint of a function? With certain kinds of functions we can start with a guess, repeatedly apply our function, and reach a fixpoint. In other words, $g(g(...g(x_0)...)) = a$ where $g(a) = a$ for some amount of iteration.

Claim. The fixpoint of $g(x) = \frac{1}{2}(x + \frac{a}{x})$ is $\sqrt{a}$.

Example. Let's try to use this fact and fixpoint iteration to calculate the $\sqrt{2}$ using $g_2(x) = \frac{1}{2}(x + \frac{2}{x})$.

x_0	2.0
$g_2(x_0)$	1.5
$g_2(g_2(x_0))$	1.416666
$g_2(g_2(g_2(x_0)))$	1.4142158
$g_2(g_2(g_2(g_2(x_0))))$	1.4142156

In only a few iterations, we get very close to the actual value of $\sqrt{2}$.

❑

4 In OCaml

So, let's see if we can write a function for finding square roots using this technique. First, we will write a general utility called `fix` that will find the fixpoint of a function `g` via iteration.

```
let rec fix (eq : 'a -> 'a -> bool) (g : 'a -> 'a) (x0 : 'a) : 'a =
```

```
  let x1 = g x0 in
  if eq x0 x1 then x0 else fix eq g x1
```

This is a high-order function that handles the fixpoint iteration for us. Here we determine when two successive steps are equal using a provided `eq` predicate. Depending on the type of data you're computing over (i.e. your `'a`) this may or may not be OCaml's built-in `(=)`.

Now we simply need to write our g.

```
let my_sqrt (a : float) : float =
  let g x = (1.0 /. 2.0) *. (x +. (a /. x)) in
  let eq x y = (abs_float (x -. y)) <= 0.0001 in
  fix eq g a
```

The only tricky thing here is our notion of equality. Because of floating point imprecision we just say two floats are equal if they are within 0.0001 of each other. Tracing out g we see something similar to the table above.

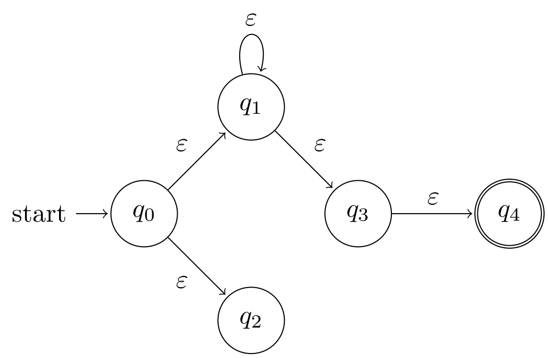
```
g <-- 2.
g --> 1.5
g <-- 1.5
g --> 1.41666666666666652
g <-- 1.41666666666666652
g --> 1.41421568627450966
g <-- 1.41421568627450966
g --> 1.41421356237468987
- : float = 1.41421568627450966
```

5 Back to NFAs

At this point you may wonder what any of this has to do with NFAs. What if I told you that the solutions to both `e_closure` and `nfa_to_dfa` are just the fixpoint of certain functions. It's true!

Claim. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. The ε-closure of $S \subseteq Q$ is the fixpoint of $g : \mathcal{P}(Q) \to \mathcal{P}(Q)$ where $g(S) = S \cup \{q \mid s \in S,\ q \in \delta(s, \varepsilon)\}$. Specifically, the fixpoint of g where $x_0 = S$.

Example. We can see this in action on an actual NFA.



Let's walk through each iteration of our g function to calculate ε-closure($\{q_0\}$).

x_0	$\{q_0\}$
$g(x_0)$	$\{q_0, q_1, q_2\}$
$g(g(x_0))$	$\{q_0, q_1, q_2, q_3\}$
$g(g(g(x_0)))$	$\{q_0, q_1, q_2, q_3, q_4\}$
$g(g(g(g(x_0))))$	$\{q_0, q_1, q_2, q_3, q_4\}$

Remark. Notice that g is only a minor variation on `move`.

❏

6 The subset construction

Finally, we arrive at `nfa_to_dfa`. It too can be calculated by fixpoint iteration.

Claim. Let $M = (Q, \Sigma, \delta, q_0, F)$ and $h : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$ where $h(q, t) =$ ε-closure(move(q, t)). In other words, given subset q, the function h computes the subset associated with transitioning on input t.

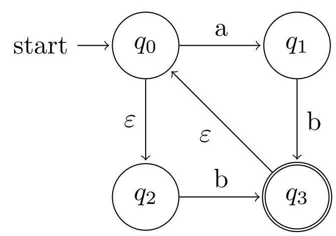
Our desired DFA is a fixpoint of $g : \mathcal{M} \to \mathcal{M}$ where

1. $g(\bar{Q}, \Sigma, \bar{\delta}, \bar{q_0}, \bar{F}) = (\bar{Q} \cup \bar{Q}', \Sigma, \bar{\delta} \cup \bar{\delta}', \bar{q_0}, \bar{F} \cup \bar{F}')$,
2. $\bar{Q}' = \{h(q,t) \mid q \in \bar{Q}, t \in \Sigma\}$,
3. $\bar{\delta}' = \{(q, t, h(q,t)) \mid q \in \bar{Q}, t \in \Sigma\}$, and
4. $\bar{F}' = \{q \mid q \in \bar{Q}', q \cap F \neq \emptyset\}$.

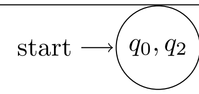
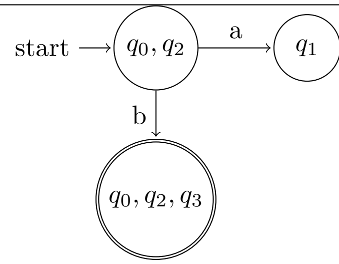
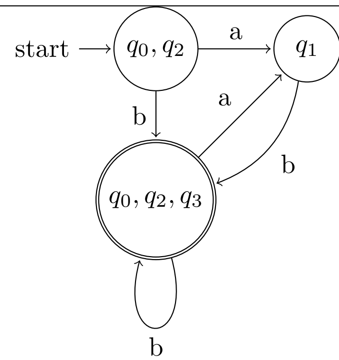
Specifically, if $\bar{q_0} = \varepsilon\text{-closure}(q_0)$ then our DFA is the fixpoint of g where $x_0 = (\{\bar{q_0}\}, \Sigma, \emptyset, \bar{q_0}, \{q \mid q = \bar{q_0}, q \cap F \neq \emptyset\})$.

Example.

Here is an NFA that we will convert to a DFA.



We will do the same process as before applying g each time.

x_0	
$g(x_0)$	
$g(g(x_0))$	
$g(g(g(x_0)))$	Same as above.

❏

Remark. When implementing `nfa_to_dfa` we recommend the following division of labor.

1. `nfa_to_dfa` calls `fix` to find the appropriate fixpoint of `step_dfa`.
2. `step_dfa` (analogous to g above) computes one step of the subset construction (i.e. applies `step_state` over all states in the DFA).
3. `step_state` computes one step of the subset construction on a given state (i.e. applies `step_symbol` over all symbols in Σ).
4. `step_symbol` adds new subset, transition, and possibly final state, given a state and input symbol (i.e. applies `step` and unions into DFA).
5. `step` is equivalent to h described above.