DFAs in Coq

Table of Contents

DFAs are defined as a 5-tuples; A DFA M is defined as $(Q, \Sigma, s, F, \delta)$, where:

Q is the finite set of states.

Σ is the finite set of input symbols, also called the alphabet.

$\delta : Q \times \Sigma \to Q$ is the transition function.

$s \in Q$ is the initial or start state.

$F \subseteq Q$ is the set of final or accept states.

The Coq formalization is as follows:

```
Record dfa (Q Σ : Type) := {
  states : list Q;
  char : list Σ;
  s : Q;
  F : Q → bool;
  δ : Q → Σ → Q;
}.
```

`Type` is effectively infinite, so finiteness is ensured by also asking for a list of the states and characters of the alphabet. As a notion of finite sets has not been developed in this project, the notion of being a final state is defined as a boolean predicate; if `M.(F) q` is `true` then it is a final state, else it is not.

Computation

Computation is defined here using the δ^* function which is defined in Coq as `delta_star`:

```
Fixpoint delta_star {Q Σ : Type} (M : dfa Q Σ) (p : Q) (x : list Σ) :=
  match x with
  | [] ⇒ p
  | x :: xs ⇒ delta_star M (M.(δ) p x) xs
end.
```

Some properties about δ^* are now proved.

```
Section DeltaStar.

  Variables Q Σ : Type.
  Variable M : dfa Q Σ.

  Lemma delta_cons : forall p a x,
    delta_star M (δ M p a) x = delta_star M p (a :: x).
  Proof. trivial. Qed.

  Lemma delta_cat : forall p x y,
    delta_star M p (x ++ y) = delta_star M (delta_star M p x) y.
  Proof.
    intros p x.
    gendep p.
```

```
Q, Σ: Type    M: dfa Q Σ    x: list Σ
----------------------------------------
forall (p : Q) (y : list Σ),
delta_star M p (x ++ y) =
delta_star M (delta_star M p x) y
```

```
    induct x.
```

```
Q, Σ: Type    M: dfa Q Σ    a: Σ    x: list Σ
IHx:
 forall (p : Q) (y : list Σ),
 delta_star M p (x ++ y) = delta_star M (delta_star M p x) y
p: Q    y: list Σ
----------------------------------------
delta_star M (δ M p a) (x ++ y) =
delta_star M (delta_star M (δ M p a) x) y
```

The `induct` tactic is a custom tactic which tries to discharge the base case, because most base cases in induction proofs are easily solvablw with basic tactics. As you can see here, `induct x` skips the base case, and moves on to the inductive case.

```
    - rewrite IHx.
```

```
Q, Σ: Type    M: dfa Q Σ    a: Σ    x: list Σ
IHx:
 forall (p : Q) (y : list Σ),
 delta_star M p (x ++ y) = delta_star M (delta_star M p x) y
p: Q    y: list Σ
----------------------------------------
delta_star M (delta_star M (δ M p a) x) y =
delta_star M (delta_star M (δ M p a) x) y
```

```
reflexivity.
  Qed.

  Lemma delta_single: forall p a,
      M.(δ) p a = delta_star M p [a].
  Proof. trivial. Qed.
```

The following theorem, `delta_step` is important for future proofs. It is also another way to look at the δ^* function.

```
  Theorem delta_step: forall w p x,
    delta_star M p (w ++ [x]) = M.(δ) (delta_star M p w) x.
  Proof.
    induct w.
```

```
Q, Σ: Type    M: dfa Q Σ    a: Σ    w: list Σ
IHw:
 forall (p : Q) (x : Σ), delta_star M p (w ++ [x]) = δ M
 (delta_star M p w) x
p: Q    x: Σ
________________________________________________
delta_star M (δ M p a) (w ++ [x]) =
δ M (delta_star M (δ M p a) w) x
```

```
    - rewrite IHw.
```

```
Q, Σ: Type    M: dfa Q Σ    a: Σ    w: list Σ
IHw:
 forall (p : Q) (x : Σ), delta_star M p (w ++ [x]) = δ M
 (delta_star M p w) x
p: Q    x: Σ
________________________________________________
δ M (delta_star M (δ M p a) w) x =
δ M (delta_star M (δ M p a) w) x
```

```
reflexivity.
  Qed.

End DeltaStar.
```

Acceptance for DFAs

Acceptance of a word w by a dfa M is as simple as checking if $\delta^*(s, w) \in F$.

```
Definition acceptb {Q Σ} (M : dfa Q Σ) word : bool :=
  M.(F) (delta_star M M.(s) word).
```

Complement of a DFA

The complement construction of a DFA is very simpl. You only need to turn final states into non-final states and vice-versa. This is achieved in Coq by performing the boolean negation of `M.(F)`

```
Section Complement.

  Definition compl_dfa {Q Σ} (M: dfa Q Σ): dfa Q Σ := {|
    states := M.(states);
    char := M.(char);
    s := M.(s);
    F := fun x ⇒ negb (M.(F) x);
    δ := M.(δ);
  |}.

  Variables Q Σ : Type.
  Variable M : dfa Q Σ.
```

The lemma that follows is a **mirorring** lemma. It shows how both the original DFA and the complement DFA *move* together or *mirror* each other i.e. for the same input string, the complement DFA **must** land on the same state as the original DFA, provided we start from the same state.

```
  Lemma compl_dfa_step: forall p w,
    delta_star M p w = delta_star (compl_dfa M) p w.
  Proof.
    intros.
    induct' w rev_ind.
```

```
Q, Σ: Type    M: dfa Q Σ    p: Q    x: Σ    w: list Σ
IHw: delta_star M p w = delta_star (compl_dfa M) p w
----------------------------------------------------
delta_star M p (w ++ [x]) =
delta_star (compl_dfa M) p (w ++ [x])
```

```
    - simpl in *.
```

```
Q, Σ: Type    M: dfa Q Σ    p: Q    x: Σ    w: list Σ
IHw: delta_star M p w = delta_star (compl_dfa M) p w
----------------------------------------------------
delta_star M p (w ++ [x]) =
delta_star (compl_dfa M) p (w ++ [x])
```

```
      rewrite delta_step.
```

```
Q, Σ: Type    M: dfa Q Σ    p: Q    x: Σ    w: list Σ
IHw: delta_star M p w = delta_star (compl_dfa M) p w
----------------------------------------------------
δ M (delta_star M p w) x =
delta_star (compl_dfa M) p (w ++ [x])
```

```
      rewrite delta_step.
```

```
Q, Σ: Type    M: dfa Q Σ    p: Q    x: Σ    w: list Σ
IHw: delta_star M p w = delta_star (compl_dfa M) p w
----------------------------------------------------
δ M (delta_star M p w) x =
δ (compl_dfa M) (delta_star (compl_dfa M) p w) x
```

```
      rewrite IHw.
```

```
Q, Σ: Type    M: dfa Q Σ    p: Q    x: Σ    w: list Σ
IHw: delta_star M p w = delta_star (compl_dfa M) p w
-----------------------------------------------------
δ M (delta_star (compl_dfa M) p w) x =
δ (compl_dfa M) (delta_star (compl_dfa M) p w) x
```

```
      reflexivity.
  Qed.
```

We can then use this lemma to prove that our complement DFA constructions is actually correct i.e. $w \in L(M) \longleftrightarrow w \notin L(\overline{M})$.

```
  Theorem compl_dfa_correct: forall w,
    acceptb M w = true <-> acceptb (compl_dfa M) w = false.
  Proof.
    intros.
```

```
Q, Σ: Type    M: dfa Q Σ    w: list Σ
-----------------------------------------------------
acceptb M w = true <-> acceptb (compl_dfa M) w = false
```

```
    unfold acceptb.
    split.
    all: rewrite compl_dfa_step;
         simpl;
         apply Bool.negb_false_iff.
  Qed.

End Complement.
```

Product Construction: Intersection and Union

The intersection of two DFAs is now defined. Given DFAs M_1 and M_2 with the same Σ we can define the intersection DFA $M_\cap$ as:

- $Q_\cap = Q_1 \times Q_2$
- $s_\cap = (s_1, s_2)$
- $F_\cap = F_1 \cap F_2$ i.e. $(q_1, q_2) \in F_\cap \longleftrightarrow q_1 \in F_1 \land q_2 \in F_2$
- $\delta_\cap(q_1, q_2) = (\delta_1(q_1), \delta_2(q_2))$

```
Section Product.

Definition inters_dfa {Q_1 Q_2 Σ} (M_1: dfa Q_1 Σ) (M_2: dfa Q_2 Σ) :
dfa (Q_1 * Q_2) Σ := {|
  states := cross_product (states M_1) (states M_2);
  char := (char M_1);
  s := (s M_1, s M_2);
  F := fun p => match p with (a, c) => (F M_1 a) && (F M_2 c) end;
  δ := fun p x => match p with (a, c) => (δ M_1 a x, δ M_2 c x) end;
|}.
```

What follows is the mirroring lemma for the intersection DFA...

```
Lemma inters_dfa_step Q_1 Q_2 Σ:
  forall (M_1: dfa Q_1 Σ) (M_2: dfa Q_2 Σ) p q w,
    delta_star (inters_dfa M_1 M_2) (p, q) w
      = (delta_star M_1 p w, delta_star M_2 q w).
Proof.
  induct' w rev_ind.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ    p: Q_1
q: Q_2    x: Σ    w: list Σ
IHw: delta_star (inters_dfa M_1 M_2) (p, q) w
     =
     (delta_star M_1 p w, delta_star M_2 q w)
-------------------------------------------------------------------
delta_star (inters_dfa M_1 M_2) (p, q) (w ++ [x]) =
(delta_star M_1 p (w ++ [x]),
delta_star M_2 q (w ++ [x]))
```

```
  - rewrite delta_step.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ    p: Q_1
q: Q_2    x: Σ    w: list Σ
IHw: delta_star (inters_dfa M_1 M_2) (p, q) w
     =
     (delta_star M_1 p w, delta_star M_2 q w)
-------------------------------------------------------------------
δ (inters_dfa M_1 M_2)
  (delta_star (inters_dfa M_1 M_2) (p, q) w) x =
(delta_star M_1 p (w ++ [x]),
delta_star M_2 q (w ++ [x]))
```

```
    rewrite IHw.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ    p: Q_1
q: Q_2    x: Σ    w: list Σ
IHw: delta_star (inters_dfa M_1 M_2) (p, q) w
     =
     (delta_star M_1 p w, delta_star M_2 q w)
-------------------------------------------------------------------
δ (inters_dfa M_1 M_2)
  (delta_star M_1 p w, delta_star M_2 q w) x =
(delta_star M_1 p (w ++ [x]),
delta_star M_2 q (w ++ [x]))
```

```
    simpl.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ    p: Q_1
q: Q_2    x: Σ    w: list Σ
IHw: delta_star (inters_dfa M_1 M_2) (p, q) w
     =
     (delta_star M_1 p w, delta_star M_2 q w)
------------------------------------------------------------
(δ M_1 (delta_star M_1 p w) x,
δ M_2 (delta_star M_2 q w) x) =
(delta_star M_1 p (w ++ [x]),
delta_star M_2 q (w ++ [x]))
```

```
    rewrite delta_step.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ    p: Q_1
q: Q_2    x: Σ    w: list Σ
IHw: delta_star (inters_dfa M_1 M_2) (p, q) w
     =
     (delta_star M_1 p w, delta_star M_2 q w)
------------------------------------------------------------
(δ M_1 (delta_star M_1 p w) x,
δ M_2 (delta_star M_2 q w) x) =
(δ M_1 (delta_star M_1 p w) x,
delta_star M_2 q (w ++ [x]))
```

```
    rewrite delta_step.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ    p: Q_1
q: Q_2    x: Σ    w: list Σ
IHw: delta_star (inters_dfa M_1 M_2) (p, q) w
     =
     (delta_star M_1 p w, delta_star M_2 q w)
------------------------------------------------------------
(δ M_1 (delta_star M_1 p w) x,
δ M_2 (delta_star M_2 q w) x) =
(δ M_1 (delta_star M_1 p w) x,
δ M_2 (delta_star M_2 q w) x)
```

```
    reflexivity.
Qed.
```

...and the correctness of the intersection DFA.

```
Theorem inters_dfa_correct Q_1 Q_2 Σ:
  forall (M_1: dfa Q_1 Σ) (M_2: dfa Q_2 Σ) w,
  acceptb (inters_dfa M_1 M_2) w = true
    <-> (acceptb M_1 w = true) /\ (acceptb M_2 w = true).
Proof.
```

```
Q_1, Q_2, Σ: Type
------------------------------------------------------------
forall (M_1 : dfa Q_1 Σ) (M_2 : dfa Q_2 Σ)
  (w : list Σ),
acceptb (inters_dfa M_1 M_2) w = true <->
acceptb M_1 w = true /\ acceptb M_2 w = true
```

```
  unfold acceptb.
```

```
  split.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ
------------------------------------------------------------
F (inters_dfa M_1 M_2)
  (delta_star (inters_dfa M_1 M_2)
     (s (inters_dfa M_1 M_2)) w) = true ->
F M_1 (delta_star M_1 (s M_1) w) = true /\
F M_2 (delta_star M_2 (s M_2) w) = true
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ
------------------------------------------------------------
F M_1 (delta_star M_1 (s M_1) w) = true /\
F M_2 (delta_star M_2 (s M_2) w) = true ->
F (inters_dfa M_1 M_2)
  (delta_star (inters_dfa M_1 M_2)
     (s (inters_dfa M_1 M_2)) w) = true
```

```
  all: simpl;
       rewrite inters_dfa_step;
       apply Bool.andb_true_iff.
Qed.
```

The union DFA is defined very easily using DeMorgan's law:

$$M_\cup = \overline{\overline{M_1} \cap \overline{M_2}}$$

, which we define as such in Coq...

```
Definition union_dfa {Q_1 Q_2 Σ} (M_1: dfa Q_1 Σ) (M_2: dfa Q_2 Σ) :=
  compl_dfa (inters_dfa (compl_dfa M_1) (compl_dfa M_2)).
```

...and then prove its correctness.

```
Theorem union_dfa_correct Q_1 Q_2 Σ:
  forall (M_1: dfa Q_1 Σ) (M_2: dfa Q_2 Σ) w,
  acceptb (union_dfa M_1 M_2) w = true
    <-> (acceptb M_1 w = true) \/ (acceptb M_2 w = true).
Proof.
```

```
Q_1, Q_2, Σ: Type
------------------------------------------------------------
forall (M_1 : dfa Q_1 Σ) (M_2 : dfa Q_2 Σ)
  (w : list Σ),
acceptb (union_dfa M_1 M_2) w = true <->
acceptb M_1 w = true \/ acceptb M_2 w = true
```

```
  split; unfold union_dfa; intros.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ
H: acceptb
   (compl_dfa
   (inters_dfa (compl_dfa M_1) (compl_dfa M_2)))
   w = true
------------------------------------------------------------
acceptb M_1 w = true \/ acceptb M_2 w = true
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ    H: acceptb M_1 w = true \/ acceptb M_2 w = true
------------------------------------------------------------
acceptb
  (compl_dfa
     (inters_dfa (compl_dfa M_1) (compl_dfa M_2))) w =
true
```

```
  -
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ
H: acceptb
   (compl_dfa
   (inters_dfa (compl_dfa M_1) (compl_dfa M_2)))
   w = true
------------------------------------------------------------
acceptb M_1 w = true \/ acceptb M_2 w = true
```

```
apply compl_dfa_correct_corr in H.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ
H: acceptb
   (inters_dfa (compl_dfa M_1) (compl_dfa M_2)) w =
   false
------------------------------------------------------------
acceptb M_1 w = true \/ acceptb M_2 w = true
```

```
    apply inters_dfa_correct_corr in H.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ    H: acceptb (compl_dfa M_1) w = false
                \/
                acceptb (compl_dfa M_2) w = false
------------------------------------------------------------
acceptb M_1 w = true \/ acceptb M_2 w = true
```

```
    destruct H as [H | H];
    apply compl_dfa_correct in H;
    [left | right];
    assumption.
  -
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ    H: acceptb M_1 w = true \/ acceptb M_2 w = true
----------------------------------------------------------------
acceptb
  (compl_dfa
     (inters_dfa (compl_dfa M_1) (compl_dfa M_2))) w =
true
```

```
apply compl_dfa_correct_corr.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ    H: acceptb M_1 w = true \/ acceptb M_2 w = true
----------------------------------------------------------------
acceptb (inters_dfa (compl_dfa M_1) (compl_dfa M_2)) w =
false
```

```
    apply inters_dfa_correct_corr.
```

```
Q_1, Q_2, Σ: Type    M_1: dfa Q_1 Σ    M_2: dfa Q_2 Σ
w: list Σ    H: acceptb M_1 w = true \/ acceptb M_2 w = true
----------------------------------------------------------------
acceptb (compl_dfa M_1) w = false \/
acceptb (compl_dfa M_2) w = false
```

```
    destruct H as [H | H];
    apply compl_dfa_correct in H;
    [left | right];
    assumption.
Qed.

End Product.
```