

Introduction to SML

Closure and Decision Properties for DFAs

Michael R. Hansen

mrh@imm.dtu.dk

Informatics and Mathematical Modelling
Technical University of Denmark

Overview

- Representation of a DFA $M = (Q, \Sigma, \delta, q_0, F)$
 - Simple Set representation
- Closure properties for DFAs
- Emptiness problem for DFAs
 - Reachability of states in DFAs

Programs are on the homepage

Simple set library – *Signature*

- needed for set of states and set of alphabet symbols
- sets are represented by lists without repetitions: `''a list`

`remDub` : `''a list -> ''a list`

`setAdd` : `''a list * 'a -> ''a list`

`setUnion` : `''a list * ''a list -> ''a list`

`remOne` : `''a list * 'a -> ''a list`

`setDifference` : `''a list * ''a list -> ''a list`

`cartesian` : `''a list * ''b list -> (''a * ''b) list`

`setEquals` : `''a list * ''a list -> bool`

- Alternatives: use SML Basis library, e.g. the module `Splayset`

Cartesian product: $A \times B$

$$\text{cartesian}([x_1, \dots, x_m], [y_1, y_2, \dots, y_n]) = \\ [(x_1, y_1), (x_1, y_2), \dots, (x_1, y_n), \dots, (x_m, y_1), (x_m, y_2), \dots, (x_m, y_n)]$$

In SML

```
fun cartesian(xs, ys) =  
  let fun pairOne(x, []) = []  
      | pairOne(x, y::ys) = (x,y)::pairOne(x,ys)  
  
      fun cart([], ys) = []  
      | cart(x::xs, ys) = pairOne(x, ys) @ cart(xs,ys)  
  in cart(xs,ys) end  
  
- cartesian([1,2], ["a","b","c"]);  
> val it = [(1, "a"), (1, "b"), (1, "c"),  
            (2, "a"), (2, "b"), (2, "c")] : (int * string) list
```

Representation of a DFA $M = (Q, \Sigma, \delta, q_0, F)$

- states of any equality type $''q$
- alphabet symbols of any equality type $''a$
- the set of accepting states F is represented by a predicate P :

$$q \in F \text{ iff } P(q)$$

type $''q, ''a$ automaton =

$''q$ list	(* Q *)
* $''a$ list	(* Σ *)
* ($''q$ * $''a$ -> $''q$)	(* δ *)
* $''q$	(* q_0 *)
* ($''q$ -> bool)	(* F *)

Simple example

- an automaton accepting an even number of 0s –

```
val e0 : (int, string) automaton =
  let val Q = [0, 1]
      val S = ["0", "1"]
      fun d(0, "0") = 1
          | d(0, "1") = 0
          | d(1, "0") = 0
          | d(1, "1") = 1
      val q0 = 0
      fun F q = q=0
  in (Q, S, d, q0, F)
end;
```

An automaton `e1 : (int, string) automaton` accepting an even number of 1s is declared similarly.

$w \in L(A)$, where $A = (Q, \Sigma, \delta, q_0, F)$

Remember $w \in L(A) \iff \hat{\delta}(q_0, w) \in F$, where

$$\begin{aligned}\hat{\delta}(q, []) &= q \\ \hat{\delta}(q, s :: ss) &= \hat{\delta}(\delta(q, s), ss)\end{aligned}$$

Directly expressed in SML as an infix function (`infix 2 inL`)

```
fun w inL (Q, S, d, q0, F) =  
  let fun dh(q, []) = q  
        | dh(q, s::ss) = dh(d(q, s), ss)  
  in F(dh(q0, w)) end;
```

```
- ["0", "1", "1", "0"] inL e0;
```

```
> val it = true : bool
```

```
- ["0", "1", "1", "0", "0"] inL e0;
```

```
> val it = false : bool
```

Product DFA for Intersection

```
infix 6 intersection;
```

```
fun (Q1, S1, d1, q01, F1) intersection (Q2, S2, d2, q02, F2) =
```

```
  let val S = if S1 setEquals S2 then S1  
              else raise Automaton "Incompatible alphabets"
```

```
  val Q = cartesian(Q1, Q2)
```

```
  fun d((q1, q2), a) = (d1(q1, a), d2(q2, a))
```

```
  val q0 = (q01, q02)
```

```
  fun F(q1, q2) = F1 q1 andalso F2 q2;
```

```
in  (Q, S, d, q0, F) end;
```

An Example

- Constructing advanced automata from simple ones

An automaton accepting an even number of 0s and 1s:

```
- val e01 as (Q, S, d, q0, F) = e0 intersection e1;

> val Q = [(0,0), (0,1), (1,0), (1,1)]: (int*int) list
    val S = ["0", "1"] : string list
    val d = fn : (int * int) * string -> int * int
    val q0 = (0, 0) : int * int
    val F = fn : int * int -> bool

- d((1,0), "0");
> val it = (0, 0) : int * int

- F(0,0);
> val it = true : bool

- ["0", "1"] @ ["0", "1", "0", "0"] inL e01;
> val it = true : bool
```

Decision Problems for DFAs

- $L(A) = \{\}$?

Solution: reachability in graphs using a simple coloring technique

- $L(A_1) \subseteq L(A_2)$?

Solution: use closure properties and $L(A) = \{\}$?

The problem $L(A) = \{\}$?

Let $A = (Q, \Sigma, \delta, q_0, F)$

Tasks:

- Find all states reachable from the start state q_0
- Check whether F contains a reachable state

Use simple coloring to find reachable states:

- The *white set* contains states which still should be treated
- The *black set* contains the already treated reachable states

Initially: $white = \{q_0\}$ and $black = \{\}$

while $white \neq \{\}$ do

 let $q \in white$, $qs = \bigcup_{a \in \Sigma} \delta(q, a)$

 in $black' = black \cup \{q\}$ and $white' = (white \cup qs) \setminus black'$

Program for the problem $L(A) = \emptyset$

```
(* find all successor states of q, given d and S *)
fun succStates(d,q,S) = let fun f([], qs)      = remDub(qs)
                             | f(e::es, qs) = f(es, d(q,e)::qs)
                             in f(S, []) end
```

```
(* reachability analysis *)
fun reach((d, S), ([], black)) = black
  | reach((d, S), (q::white, black)) =
    let val black' = setAdd(black, q)
        val white' = setUnion(white, succStates(d, q, S))
    in reach((d,S),(setDifference(white',black'),black')) end
```

```
(* emptiness test *)
fun isEmpty (Q, S, d, q0, F) =
  let val rs = reach((d, S),([q0], []))
      fun allReject []      = true
        | allReject(q::qs) = not(F q) andalso allReject qs
  in allReject rs end;
```