Forlan and DFA Design/Testing

Tuesday, October 5, 2010
Reading: Stoughton Sec. 2.3 and end of Sec. 3.10

CS235 Languages and Automata

Department of Computer Science
Wellesley College

Forlan Overview

Forlan is a collection of SML modules developed by Alley Stoughton at Kansas State University that encourages experimentation with formal languages, automata, and grammars.

To use Forlan from shell, use `forlan` instead of `sml`.

To use Forlan from emacs, use C-u M-x sml with `forlan` as an argument, and then do work in `*forlan*` buffer.

Symbol and String Conventions in Forlan

No alphabets! The set of symbols is fixed to be **Sym**, which contains:

- digits 0, 1, …, 9
- upper case letters A, B, …, Z
- lower case letters a, b, …, z
- < and > delimiting any sequence of ASCII characters in which < and > are properly nested. E.g., <plus>, <<a>, <b, c>>

Symbols are ordered first by length, and then lexicographically:

0 < … < 9 < A < … < Z < a … < z < <0> < … <z> < <00> … < <zz> < …

The universe of strings is **Str** = all strings over **Sym**.
The empty string is written % .

Strings are ordered first by length, and then lexicographically. E.g.:

% < a < <ab> < ab < ac < a<bbb> < <bb><aa> < <bbb>a < aa<bb> < <bb>aa

Can't write (a + b)*c. Instead, write <LPAREN>a<PLUS>b<RPAREN><TIMES>c

The Forlan Sym Module

```
- open Sym; (* I'm using this to display the bindings in this module *)
opening Sym
  … (* I'm leaving out a bunch of details here *)
  type sym = sym (* sym is an abstract type whose representation is hidden *)
  val fromString : string -> sym
  val toString : sym -> string
  val compare : sym * sym -> order
  val equal : sym * sym -> bool
  val size : sym -> int

- val syms = map Sym.fromString ["a", "b", "<foo>", "<a<b>>"];
val syms = [-,-,-,-] : sym list

- map Sym.toString syms;
val it = ["a","b","<foo>","<a<b>>"] : string list

- Sym.compare(Sym.fromString "c", Sym.fromString "<ab>");
val it = LESS : order

- Sym.compare(Sym.fromString "<c>", Sym.fromString "<ab>");
val it = LESS : order

- Sym.compare(Sym.fromString "<cd>", Sym.fromString "<ab>");
val it = GREATER : order
```

The Forlan SymSet Module

```
- open SymSet;
opening SymSet
  val fromList : sym list -> sym set
  val compare : sym set * sym set -> order
  val memb : sym * sym set -> bool
  val subset : sym set * sym set -> bool
  val equal : sym set * sym set -> bool
  val map : ('a -> sym) -> 'a set -> sym set
  val union : sym set * sym set -> sym set
  val inter : sym set * sym set -> sym set
  val minus : sym set * sym set -> sym set
  val powSet : sym set -> sym set set
  val fromString : string -> sym set
  val input : string -> sym set
  val toString : sym set -> string
  ... (* I'm leaving out some details *)
```

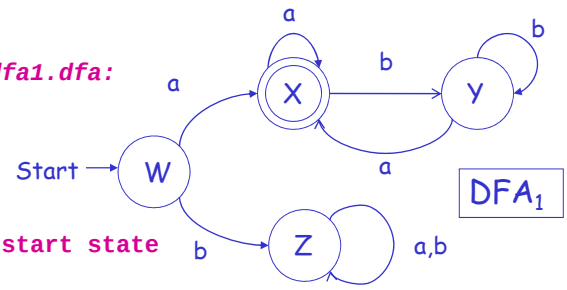

SymSet Examples

```
- val ss1 = SymSet.fromList (List.map Sym.fromString ["a", "b", "<foo>", "<a<b>>"]);
val it = - : sym set

- SymSet.toString ss1;
val it = "a, b, <foo>, <a<b>>" : string

- val ss2 = SymSet.fromString "a,c,<foo>,<bar>";
val ss2 = - : sym set

- SymSet.memb(Sym.fromString "b", ss1);
val it = true : bool

- SymSet.memb(Sym.fromString "b", ss2);
val it = false : bool

- SymSet.toString (SymSet.union(ss1,ss2));
val it = "a, b, c, <bar>, <foo>, <a<b>>" : string

- SymSet.toString (SymSet.inter(ss1,ss2));
val it = "a, <foo>" : string

- SymSet.toString (SymSet.minus(ss1,ss2));
val it = "b, <a<b>>" : string

- SymSet.equal(ss1, ss2);
val it = false : bool
```

The Forlan DFA Module

```
- open DFA;
opening DFA
  type dfa
  val fromString : string -> dfa
  val input : string -> dfa
  val toString : dfa -> string
  val output : string * dfa -> unit
  val states : dfa -> sym set
  val startState : dfa -> sym
  val acceptingStates : dfa -> sym set
  val numStates : dfa -> int
  val numTransitions : dfa -> int
  val alphabet : dfa -> sym set
  val accepted : dfa -> str -> bool
  val renameStates : dfa * sym_rel -> dfa
  val renameStatesCanonically : dfa -> dfa
  val isomorphism : dfa * dfa * sym_rel -> bool
  val findIsomorphism : dfa * dfa -> sym_rel
  val isomorphic : dfa * dfa -> bool
 ... (* I'm leaving out a bunch of details here *)
```


A Sample Automaton in Forlan Syntax



Contents of file dfa1.dfa:

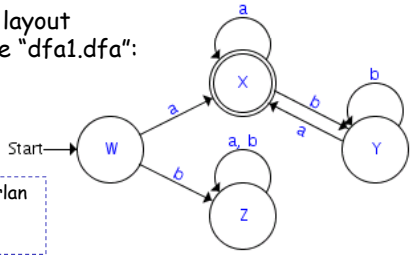
```
{states}
W,X,Y,Z
{start state}
W # always single start state
{accepting states}
X # any number of accepting states,
  # separated by commas
{transitions}
W,a -> X; W,b -> Z;
X,a -> X; X,b -> Y;
Y,a -> X; Y,b -> Y;
Z,a -> Z; Z,b -> Z # Careful! No semi-colon at end
```

JFA: Manipulating FAs in Java

JFA is a Java application in the Forlan toolset for drawing and viewing finite automata. Using JFA you can:

- Draw and label nodes and edges for a finite automaton.
- Save the diagram as a file in various formats:
  - The Forlan DFA (FA) format (can then be used in Forlan)
  - . jfa format (remembers position information)
  - .png (as a picture)
- Read a Forlan DFA file (JFA will lay it out automatically)

For example, here's JFA's layout for the DFA read from file "dfa1.dfa":



An improved application named JForlan supercedes JFA, but it doesn't yet run on most of our machines.

DFA Examples

```
- val dfa1 = DFA.input "dfa1.dfa";
val dfa1 = - : dfa

- DFA.states dfa1;
val it = - : sym set

- SymSet.toString (DFA.states dfa1);
val it = "W, X, Y, Z" : string

- Sym.toString (DFA.startState dfa1);
val it = "W" : string

- SymSet.toString (DFA.acceptingStates dfa1);
val it = "X" : string

- SymSet.toString (DFA.alphabet dfa1);
val it = "a, b" : string- print (DFA.toString dfa1);
```

DFA Examples (continued)

```
- print (DFA.toString dfa1);
{states}
W, X, Y, Z
{start state}
W
{accepting states}
X
{transitions}
W, a -> X; W, b -> Z; X, a -> X; X, b -> Y; Y, a -> X; Y, b -> Y; Z, a -> Z;
Z, b -> Zval it = () : unit
```


The Forlan Str Module

Note that DFA.accepted : dfa -> str -> bool. Q: What's str vs. string?
A: It's a *sequen of syms.

```
- open Str;
opening Str
  type str = sym list
  val fromString : string -> str
  val toString : str -> string
  val isEmpty : str -> bool
  val isNonEmpty : str -> bool
  val compare : str * str -> order
  val equal : str * str -> bool
  val alphabet : str -> sym set
  val prefix : str * str -> bool
  val suffix : str * str -> bool
  val substr : str * str -> bool
  val power : str * int -> str
  ... (* I'm leaving out some details *)
```

Str Examples

```
- val s1 = Str.fromString "ab<c>d<efg>";
val s1 = [-,-,-,-,-] : str
- length s1;
val it = 5 : int
- map Sym.toString s1;
val it = ["a","b","<c>","d","<efg>"] : string list
- val s2 = Str.fromString "ab%c(def)%g";
val s2 = [-,-,-,-,-,-,-] : str
- Str.toString s2;
val it = "abcdefg" : string
- val s3 = Str.fromString "%";
val s3 = [] : str
- val s4 = Str.fromString "";
line 1 : end-of-file unexpected

uncaught exception Error
```

```
- Str.toString(Str.fromString "%");
val it = "%" : string
- val s5 = s1 @ s2;
val s5 = [-,-,-,-,-,-,-,-,-,-,-,-] : sym list
- Str.toString (s5);
val it = "ab<c>d<efg>abcdefg" : string
- Str.prefix (s1, s5);
val it = true : bool
- Str.prefix (s2, s5);
val it = false : bool
- Str.substr(Str.fromString("def"),s5);
val it = true : bool
- Str.substr(Str.fromString("edf"),s5);
val it = false : bool
```

```
- Str.toString(Str.power(s1,3));
val it = "ab<c>d<efg>ab<c>d<efg>ab<c>d<efg>" : string
```


The Forlan StrSet Module

```
- open StrSet;
opening StrSet
  val fromList : str list -> str set
  val compare : str set * str set -> order
  val memb : str * str set -> bool
  val subset : str set * str set -> bool
  val equal : str set * str set -> bool
  val map : ('a -> str) -> 'a set -> str set
  val union : str set * str set -> str set
  val inter : str set * str set -> str set
  val minus : str set * str set -> str set
  val powSet : str set -> str set set
  val fromString : string -> str set
  val toString : str set -> string
  val concat : str set * str set -> str set
  val power : str set * int -> str set
  val rev : str set -> str set
  val alphabet : str set -> sym set
 ... (* I'm leaving out a bunch of details here *)
```

StrSet Examples

```
- val ss1 = StrSet.fromString "in,out";
val ss1 = - : str set

- val ss2 = StrSet.fromString "doors,put,side";
val ss2 = - : str set

- StrSet.toString(StrSet.concat(ss1,ss2));
val it = "input, inside, output, indoors, outside, outdoors" : string

- StrSet.toString(StrSet.union(ss1,ss2));
val it = "in, out, put, side, doors" : string

- StrSet.toString(StrSet.power(ss1,3));
val it = "ininin, ininout, inoutin, outinin, inoutout, outinout, outoutin,
outoutout" : string
```


Testing a DFA on Individual Strings

```
- DFA.accepted dfa1 (Str.fromString "abbabaa");
val it = true : bool

- DFA.accepted dfa1 (Str.fromString "abbabaab");
val it = false : bool

- DFA.accepted dfa1 (Str.fromString "baa");
val it = false : bool

- DFA.accepted dfa1 (Str.fromString "");
line 1 : end-of-file unexpected

uncaught exception Error

- DFA.accepted dfa1 (Str.fromString "%");
val it = false : bool
```

Testing a DFA on Lots of Strings

```
(* testOnString handles empty string messiness *)
fun testOnString dfa string =
  DFA.accepted dfa (Str.fromString (if string = "" then "%" else string))

(* all strings of as and bs up to length 5  *)
- val ab5 = genStrings(5, [#"a", #"b"]);
val ab5 =
["","a","b","aa","ab","ba","bb","aaa","aab","aba","abb","baa","bab","bba",
  "bbb","aaaa","aaab","aaba","aabb","abaa","abab","abba","abbb","baaa",
  "baab","baba","babb","bbaa","bbab","bbba","bbbb","aaaaa","aaaab",
  "aaaba","aaabb","aabaa","aabab","aabba","aabbb","abaaa","abaab",
  "ababa","ababb","abbaa","abbab","abbba","abbbb","baaaa","baaab",
  "baaba","baabb","babaa", "babab","babba","babbb","bbaaa","bbaab",
  "bbaba","bbabb","bbbaa","bbbab", "bbbba","bbbbb"] : string list

(* test dfa1 on all strings of as and bs up to length 5 *)
- List.filter (testOnString dfa1) ab5);
val it =
  ["a","aa","aaa","aba","aaaa","aaba","abaa","abba","aaaaa","aaaba",
    "aabaa", "aabba","abaaa","ababa","abbaa","abbba"] : string list
```


SML String Predicates

```
(* Want to define SML predicate functions on strings that will
    help us test if our DFAs are working as expected. *)

(* Return a list of all strings on which dfa and pred differ *)
fun begin_and_end_with_a_pred str =
  let val len = String.size str
   in (len >= 1)
      andalso (String.sub(str,0) = #"a")
      andalso (String.sub(str, len - 1) = #"a")
  end

- begin_and_end_with_a_pred "abbaba";
val it = true : bool

- begin_and_end_with_a_pred "bbaba";
val it = false: bool

- begin_and_end_with_a_pred "abbab";
val it = false: bool
```

Finding Mismatches

```
(* Return a list of all strings on which dfa and pred differ *)
fun mismatches dfa pred strings =
  List.filter (fn s => (testOnString dfa s) <> (pred s)) strings

- mismatches dfa1 begin_and_end_with_a_pred ab5;
val it = [] : string list

(* dfa4 is like dfa1 except that Y is also a final state, so it accepts
    any string beginning with a *)
- val dfa4 = DFA.input "dfa4.dfa";
val dfa4 = - : dfa

- mismatches dfa4 begin_and_end_with_a_pred ab5;
val it = (* all strings that begin with a but don't end with a *)
["ab","aab","abb","aaab","aabb","abab","abbb","aaaab","aaabb","aabab",
  "aabbb","abaab","ababb","abbab","abbbb"] : string list
```


Fancier Mismatch Testing: Definition

```
(* Tests all strings with both dfa and pred. If they agree on all tests,
   print a message saying this and return true.  Otherwise, print out
   results for every mismatch and return false. *)
fun testOnStrings dfa pred strings =
  let fun printMiss miss =
        (print ("Mismatch -- \"" ^ miss
                ^ "\": DFA says " ^ (Bool.toString (testOnString dfa miss))
                ^ "; pred says " ^ (Bool.toString (pred miss))
                ^ "\n"))
      val misses = mismatches dfa pred strings
  in if misses = [] then
       (print "Passed all test cases\n"; true)
     else
       (List.app printMiss misses; false)
  end

(* Compare dfa and predicate on all strings over chars of up to length n *)
fun testOnAlphabet dfa pred chars n =
  testOnStrings dfa pred (genStrings (n,chars))
```

Fancier Mismatch Testing: Examples

```
fun test_begin_and_end_with_a n =
  testOnAlphabet (DFA.input "dfa1.dfa")
                 begin_and_end_with_a_pred
                 [#"a", #"b"]
                 n

(* Alternative definition for the above: *)
 val test_begin_and_end_with_a =
   testOnAlphabet (DFA.input "dfa1.dfa")
                  begin_and_end_with_a_pred
                  [#"a", #"b"]

- test_begin_and_end_with_a 10;
Passed all test cases
val it = true : bool

- test_begin_and_end_with_a 20;  (* This takes a minute or two – why? *)
Passed all test cases
val it = true : bool
```


The Forlan SymRel Module

```
opening SymRel
  type sym_rel = (sym,sym) rel
  val compare : sym_rel * sym_rel -> order
  val memb : (sym * sym) * sym_rel -> bool
  val subset : sym_rel * sym_rel -> bool
  val equal : sym_rel * sym_rel -> bool
  val union : sym_rel * sym_rel -> sym_rel
  val inter : sym_rel * sym_rel -> sym_rel
  val minus : sym_rel * sym_rel -> sym_rel
  val powSet : sym_rel -> sym_rel set
  val reflexive : sym_rel * sym set -> bool
  val symmetric : sym_rel -> bool
  val transitive : sym_rel -> bool
  val reflexiveClosure : sym_rel * sym set -> sym_rel
  val transitiveClosure : sym_rel -> sym_rel
  val symmetricClosure : sym_rel -> sym_rel
  val reflexiveTransitiveClosure : sym_rel * sym set -> sym_rel
  val reflexiveSymmetricClosure : sym_rel * sym set -> sym_rel
  val transitiveSymmetricClosure : sym_rel -> sym_rel
  val reflexiveTransitiveSymmetricClosure : sym_rel * sym set -> sym_rel
  … (* I'm leaving out a bunch of details here *)
```

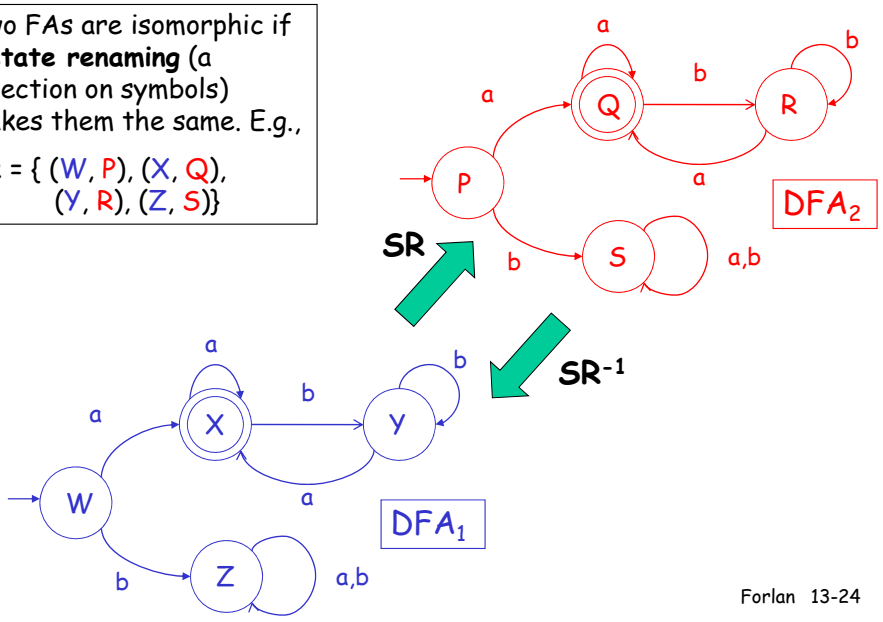
SymRel Examples

```
- val rel = SymRel.fromString "(a,b), (b,d), (d,h)";
val rel = - : sym_rel

- SymRel.symmetric rel;
val it = false : bool

- SymRel.transitive rel;
val it = false : bool

- val rel2 = SymRel.transitiveClosure rel;
val rel2 = - : sym_rel

- SymRel.toString rel2;
val it = "(a, b), (a, d), (a, h), (b, d), (b, h), (d, h)" : string

- SymRel.transitive rel2;
val it = true : bool

- SymRel.symmetric rel2;
val it = false : bool
```


Isomorphisms Between DFAs

Two FAs are isomorphic if a **state renaming** (a bijection on symbols) makes them the same. E.g.,

SR = { (W, P), (X, Q), (Y, R), (Z, S)}



Isomorphism Examples

```
- val dfa2 = DFA.renameStates  (dfa1, SymRel.fromString "(W,P),(X,Q),(Y,R),(Z,S)");
val dfa2 = - : dfa

- print (DFA.toString dfa2);
{states}
P, Q, R, S
{start state}
P
{accepting states}
Q
{transitions}
P, a -> Q; P, b -> S; Q, a -> Q; Q, b -> R; R, a -> Q; R, b -> R; S, a -> S;
S, b -> Sval it = () : unit

- DFA.isomorphic (dfa1, dfa2);
val it = true : bool

- SymRel.toString (DFA.findIsomorphism (dfa1, dfa2));
val it = "(W, P), (X, Q), (Y, R), (Z, S)" : string

- SymRel.toString (DFA.findIsomorphism (dfa2, dfa1));
val it = "(P, W), (Q, X), (R, Y), (S, Z)" : string
```


A Canonical Isomorphism

```
- val dfa3 = DFA.renameStatesCanonically dfa1;
val dfa3 = - : dfa

- print (DFA.toString dfa3);
{states}
A, B, C, D
{start state}
A
{accepting states}
B
{transitions}
A, a -> B; A, b -> D; B, a -> B; B, b -> C; C, a -> B; C, b -> C; D, a -> D;
D, b -> Dval it = () : unit

- SymRel.toString (DFA.findIsomorphism (dfa1, dfa3));
val it = "(W, A), (X, B), (Y, C), (Z, D)" : string
```