Logic Programming Project: Regular Languages

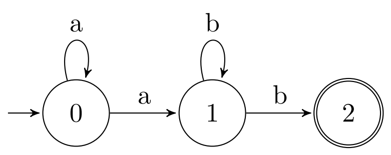
Jaco van de Pol, Formal Methods and Tools, U Twente

Deadline: June 2, 2017

Please submit your work as a **single file**, called `nfa.pl`, that can be processed by SWI-Prolog. Note that a template of `nfa.pl` is handed out as part of this assignment. Put all your answers and explanations as comments in this single file. You are supposed to work and submit solutions in groups in which you can enroll on Blackboard, as usual in this module.

1 Non-deterministic Finite State Machines

Recall that an NFA consists of states and transitions. An NFA has a unique initial state and a set of accepting states. A transition between two states is labeled with a symbol[1]. Here are two examples of NFAs in graphical form:



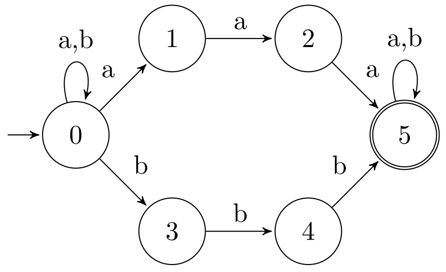


Figure 1: Graphical representation of the NFA n1 (left) and NFA n2 (right)

NFA `n1` accepts the language $\{a^m b^n \mid m > 0, n > 0\}$. NFA `n2` accepts all words that contain *aaa* or *bbb* as a subword. We represent NFAs in Prolog simply as structures of the following form:

```
nfa(Init, [trans(From,Label,To) | Transitions], Accepting)
```

where Init, From and To are states, Label is any (Prolog) symbol in the alphabet of the automaton, Transitions is a list containing the rest of the transitions, and

[1]We don't consider empty steps, labeled λ or ϵ

Accepting is a set of states. The NFAs `n1` and `n2` from Figure 1 can then be defined in Prolog as follows[2].

```
example(n1,nfa(0,[trans(0,a,0),trans(0,a,1),trans(1,b,1),trans(1,b,2)],[2])).

example(n2,nfa(0,
 [trans(0,a,0),trans(0,a,1),trans(1,a,2),trans(2,a,5),trans(5,a,5),
  trans(0,b,0),trans(0,b,3),trans(3,b,4),trans(4,b,5),trans(5,b,5)],
 [5])).
```

Question 1: computing the alphabet

Write a function `alphabet/2` that computes a list of symbols that appear on the labels of transitions in an NFA. In particular, `alphabet(N,L)` should return a list `L` for NFA `N`. Make sure that `L` does not contain duplicates. Example:

```
?- example(n1,N), alphabet(N,L).
N = nfa(0, [trans(0,a,0),trans(0,a,1),trans(1,b,1),trans(1,b,2)], [2]),
L = [a, b].
```

Hint: you should specify convenient auxiliary functions to ensure that the end result does not contain duplicate symbols.

Question 2: recognizing a word

We represent a word in Prolog as a list of symbols. For example, the word *abaabb* will be represented in Prolog as `[a,b,a,a,b,b]`. Recall that an NFA recognizes a word, if there is a path from the initial state to some accepting state, following the symbols of that word.

Write a predicate `testNFA/2`, so `testNFA(W,N)` tests if word W is in the language of NFA N. For example, the first query below should succeed, while the second fails:

```
?- example(n1,N), testNFA([a,a,b,b,b],N).  % (yes)
?- example(n2,N), testNFA([a,b,a,b,b],N).  % (no)
```

2 Regular Expressions

Next, we want to explore regular expressions. We will use the built-in facility of Prolog to define new operators, declared with `op/3`. The third argument is the name of the operator itself. The second argument shows if the operator is *infix* (`xfy`), *prefix* (`fx`) or *postfix* (`xf`). The first argument indicates the priority of the operator: operators with low priority bind strongest. For regular expressions, we introduce the following operators:

[2]These examples are included as "facts" in the skeleton file `nfa.pl` in the Assignment.

```
:- op(650,xfy,+).
:- op(640,xfy,^).
:- op(630,xf,*).
```

So this defines the infix operators $+$ (choice or union) and $\hat{}$ (concatenation or sequential composition) and postfix operator $*$ (repetition 0 or more times). The numbers indicate that $*$ binds stronger than $\hat{}$, which binds stronger than $+$.

Operators are just function symbols, but then written infix (or postfix). We can use regular unification on these operator/function symbols. So we can test our operators in Prolog:

```
?- X ^ Y = a + b ^ c.
no.
?- X + Y = a + b ^ c.
yes, X=a, Y=b^c.
```

So indeed, $\hat{}$ binds stronger than $+$. We can define more complicated regular expressions as follows:

```
(a^a + b)*         % words that repeat "aa" or "b"
(a+a^a)^(a^a^a)*   % strings of "a" that are not multiples of 3
```

Question 3: Counting Operators in a regular expression

Write a function `count(E,N)`, which counts the number N of operators ($*$,$\hat{}$,$+$) in a regular expression E.

Example:

```
?- count((a+a^a)^(a^a^a)*, N).
N=6.
```

Hint: Use `atom(X)` to test that X is a symbol (that is: has 0 operators).

Question 4: Test a word in an expression

Write a predicate `testRE(W,E)`, where W is a word and E is an expression. Examples:

```
?- testRE([a,a,b,a,a,a,a,b], (a^a+b)*).         % (yes)
?- testRE([a,a,b,a,a,a,a,a,b], (a^a+b)*).       % (no)
?- testRE([a,a,a,a,a], (a+a^a)^(a^a^a)*).       % (yes)
?- testRE([a,a,a,a,a,a], (a+a^a)^(a^a^a)*).     % (no)
?- testRE([a,b,a,b,b,c,b,c],((a^b)* ^b^c)*).    % (yes)
```

Note that the space in $*$ $\hat{}$ is required, since otherwise Prolog would consider $*\hat{}$ as a single operator by Prolog.

Hint: Recall that a word $w \in \mathcal{L}(x\hat{}y)$ if it can be split as in $w = w_1 w_2$, such that $w_1 \in \mathcal{L}(x)$ and $w_2 \in \mathcal{L}(y)$.