Interactive Theorem Proving and Applications
Année 2020-21

Prof. Burkhart Wolff
wolff@lri.fr

Project - Extending an Automata Theory
Semaine du 8 fevrier 2021

1 Objectives

This project aims at extending an existing theory of regular expressions and automata by operations that rename states via a "renaming morphism". If this morphism is injective on the states occurring in an automata, a "morphed automata" recognizes the same language. This can be used to simplify the automata construction operations.

Modelization

The theory skeleton consists of the following components:

1. `RegExp.thy` contains a the abstract syntax of regular expression together with a *denotational semantics*, a function L that assigns to each regular expression r the "language" it denotes: $L(r)$.
2. `Automata.thy` defines non-deterministic and deterministic automata, the generalized transition functions as well as the corresponding acceptance conditions for a word in the automata.
3. `RegExp2NAe.thy` contains the compiler and its corresponding correctness nad completeness conditions.

Regular Expression

There is a **abstract syntax of regular expression** and a function L give the language of this regular expression.

```
datatype 'a rexp = Empty                          ("<>")
                 | Atom 'a                        ("⌊_⌋" 65)
                 | Alt  "('a rexp)" "('a rexp)"   (infix "|" 55)
                 | Conc "('a rexp)" "('a rexp)"   (infix ":" 60)
                 | Star "('a rexp)"
```

A function L give the language of this regular expression.

```
fun             L :: "'a rexp => 'a lang"
where L_Emp :   "L Empty       = {}"
     |L_Atom:   "L (⌊a⌋)       = {[a]}"
     |L_Un:     "L (el | er)   = (L el) ∪ (L er)"
     |L_Conc:   "L (el : er)   = {xs@ys | xs ys. xs:L el ∧ ys:L er}"
     |L_Star:   "L (Star e)    = star(L e)"
```

Nondeterministic automata

na represent nondeterministic automata. It is defined In the theory Automata using record.

```
record ('a,'s)na =
   start :: "'s"
   "next":: "'a ⇒ 's ⇒ 's set"
   fin :: "('s ⇒ bool)"
```

Delta to calcul a set of states reachable after a word in term of w from initial states.

```
primrec delta :: "('a,'s)na ⇒ 'a list ⇒ 's ⇒ 's set"
where  "delta A []    p = {p}"                    (*return the set {p}*)
     |"delta A (a#w) p = ⋃(delta A w ` next A a p)"(*next A a p return a*)
```

Accepts A Definition decide if a word ca be accepted by nondeterministic automata na which is defined here.

```
definition accepts ::   "('a,'s)na ⇒ 'a list ⇒ bool"
where     "accepts A w == ∃ q ∈ delta A w (start A). fin A q"
```

Nondeterministic automata with epsilon transitions

nae is defined for nondeterministic automata with epsilon transitions, which is defined in RegExp2DAe.thy. nae is an instance of the nondeterministic (in)finite automaton(na), which allows a transformation to a new state without consuming any input symbols.

```
type_synonym ('a,'s)nae = "('a option, 's) na"
type_synonym 'a bitsNAe = "('a ,bool list) nae"
```

From regular expressions directly to nondeterministic automata with epsilon

The transition function rexp2nae converts regular expressions into such automata and is defined in file `RegExp2DAe.thy`. It use 5 definitions for the **Atom Conc Union Star** case which are defined in the same file. The goal of this project is showing the correctness of this translation.

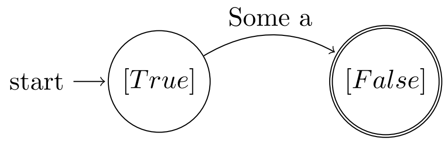
A key-problem of the construction is that the nodes from which automatas were constructed must be disjoint when "gluing" several automatas together; the present modelization realises this by a re-labelling discipline on node-names which were represented as bitstrings.

We present some diagrams to illustrate transition function of these automatas.

Atom: The Atom case. In its initial state [True] , it may consume Some a (of any type), and go over to accepting state [False] . Here is the formal definition:

```
definition atom :: "'a ⇒'a bitsNAe"
where "atom a ==⦇na.start = [True],
                na.next = (λ b s. if s =[True] ∧ b = Some a then {[False]} else {}),
                na.fin = (λ s. s = [False]) ⦈"
```

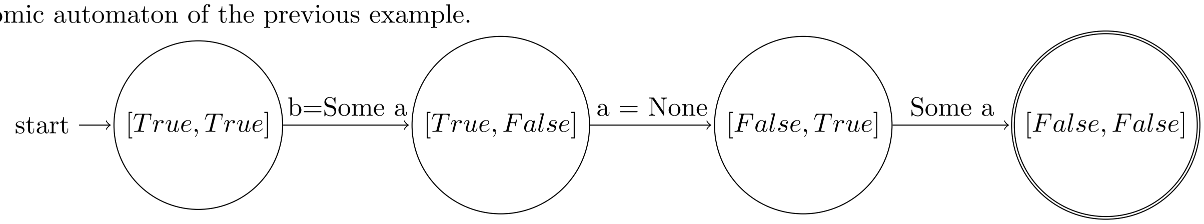
... reflecting the automaton:



Conc: The conc case. It takes two NAe l and r and returns a new NAe. What is interesting here is the injective re-labelling of the nodes; any node label in the left automaton is prefixed by `True`, any node label in the right automaton is prefixed by `False`. Since the translation of regular expressions will be done recursively over the structure of regular expressions, the construction assures that the node sets are always distinct.

```
definition conc :: "'a bitsNAe ⇒ 'a bitsNAe ⇒ 'a bitsNAe"
where  "conc l r == (let ql = na.start l;
                         dl = na.next l;
                         fl = na.fin l;
                         qr = na.start r;
                         dr = na.next r;
                         fr = na.fin r
                     in  ⦇na.start = (True#ql),
                          na.next = (λa s. case s of
                                         [] ⇒ {}
                                        |left#s ⇒ if left
                                                   then (True ## dl a s)∪
                                                        (if fl s ∧ a =None then {False#qr}
                                                                                else{})
                                                   else False ## dr a s),
                          na.fin = (λs. case s of
                                         [] ⇒ False
                                        |left#s ⇒ ¬left ∧ fr s) ⦈)  "
```

Here is the example of the concatenation of two automata l and automata r which happen to be the atomic automaton of the previous example.



Tasks:

1. Study the theory in Isabelle.
2. Define a graph morphism map_{nae} of the form : $('\sigma \Rightarrow' \sigma') \Rightarrow ('\alpha,' \sigma)nae \Rightarrow ('\alpha,' \sigma')nae$ which renames (relabels) states inside an automaton.
3. Prove: $map_{nae} f (map_{nae} g A) = map_{nae} (f\ o\ g) A$
4. Prove: for injective f, an automaton A recognises the same language than $map_{nae}\ g\ A$.
5. Try to reformulate the automata operations such as *union* via map_{nae}.

Submission

The final version of the files together with a 3-6 pages report of the theory development are due **15.3.2021** per mail at `wolff@lri.fr`.