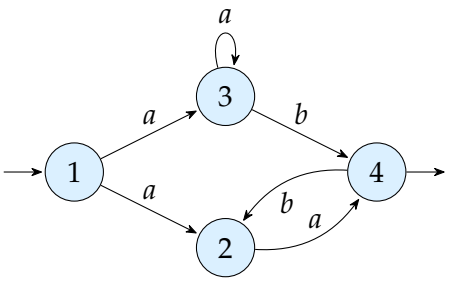
TP 3
Vcsn– 2

Version du 21 septembre 2020

In this practical, we will again use Vcsn. If needed, reread instructions from the previous one.

Exercice 1 – Evaluating deterministic automata

1. Every automaton has an equivalent deterministic automaton. The method '`automaton.determinize`' computes such a deterministic automaton.



Automate 1: Automaton to determinize

   Determinize the automate 1 and print the result. You can also check the automata of the TDs.

2. '`automaton(word)`' (and '`automaton.evaluate`') evaluates a given word on an automaton.
   Reuse yesterday's automata computed from rational expressions to evaluates words that, according to you, are recognized or not by these automata.

3. The dot notation ('`obj.fun()`') allows to easily chain several operations, in the order of their evaluation, from left to right. For instance :

```
vcsn.context('lal_char(abc), b').expression('ab*').thompson().proper()
```

   builds the Thompson automaton of the expression 'ab*', then removes its ε-transitions (and the states that have become useless).
   Add to the previous example a determinization step, and a minimization step (this concept will be seen later in class. It consists in finding the *unique* deterministic automaton with the smallest number of states).
   Then, chain commands that you already know in order to compute the automaton that corresponds to the rational expression of your choice (write a function if are familiar enough with Python).
   Apply the following steps to this automaton :
   — removal of ε-transitions
   — determinization
   — evaluation of a word

Exercice 2 – Intersection of rational languages

We will see tomorrow the *synchronized product*, which produces an automaton that recognizes the intersection of the languages of two input automata.

1. Build an automaton accepting the words on the alphabet $\{a, b\}$ that have an odd number of b. You can use the method '*expression*.automaton' to convert an expression into an NFA.
   Check your automaton with '*automaton*.shortest'.
2. The operation '*automaton* & *automaton*' (or '*automaton*.conjunction') computes the synchronized product of two automata.
   Compute an automaton that accepts the words that have an odd number of a and an odd number of b. Check it with 'shortest'.
3. From the above automaton, deduce a rational expression denoting the set of words on $\{a, b\}$ with an odd number of a and an odd number of b.
4. Vcsn supports *extended* rational expression, which feature two additional operators : 'e&f' for the intersection and 'e{c}' for the complementation.
   Go back to the previous question, but treat it directly with extended rational expressions.

Exercice 3 – Mans Hulden's problem

At the FSMNLP'08 conference, Mans Hulden mentioned a language problem whose solution is based on a complex composition of rational languages : given a rational language L, build the language L' of the words having *exactly* one infix in L. L' happens to be a rational language.

The goal of this exercise is to solve this problem. We could represent languages as automata and transform them. Since Vcsn supports extended rational expressions, we will rather use expressions. The results will be equivalent, but much easier to read.

1. Define the following function, analyze it, and try it on the expression $ab + ba$.
   Use '*expression*.shortest'.

```
ctx = vcsn.context('lal_char(abc), b')
theme
# A helper function to define expressions on 'lal_char(abc), b'.
def exp(e):
# If 'e' is not a real expression object, convert it.
return e if isinstance(e, vcsn.expression) else ctx.expression(e)

def t1(re):
re = exp(re)
all = exp('[^]*')
# * denotes the concatenation.
return all * re * all
t1('a')
```

2. Let e be a rational expression. Propose a rational expression that denotes all words having at least two disjoint (no overlap) infixes recognized by e.
3. Define a Python function to build it.
4. With 'shortest', check that your answer is (probably) correct for $e = ab + ba$.
5. Propose a rational expression denoting the set of all words that have a prefix in e and a proper suffix in e. Consider using the operator '&'.
6. Adapt your function so that it builds the latter expression, and verify it on $e = ab + ba$.
7. Adapt your expression (and your function) so that it recognizes every word having at least two infixes from e, that may overlap.

8. The operation '*expression* `%` *expression*' (or '`expression.difference`') gives a rational expression denoting the words accepted by the first expression but not by the second one.
   Modify your function so that it recognizes every word having exactly one infix in e and try it.
9. Check that your answer is correct by considering $L = \{ab, aba\}$. What do you observe? How would you explain it?
10. Given a rational language L, describe the language of words of L that have one (or more) proper infix in L.
    Implement a function '`composite`' that computes this language from rational expressions, and test it on $L = \{ab, aba\}$.
11. Finally, propose a function that takes a rational expression e as argument and returns a rational expression that recognizes all words having exactly one infix in $L(e)$.