A PYTHON MODULE FOR EXPLORING REVERSED SUM-PRODUCT PAIRS

XANDER FABER AND JON GRANTHAM

Abstract. We give a brief tutorial on our Python implementation for the construction of deterministic finite automata (DFAs) for reversed sum-product pairs. The module `reverse.py` can be obtained from

https://github.com/RationalPoint/reverse,

and it is compatible with Python3 and Sage 9.x.

The Python module `reverse` was written to support the research in [1]. To begin, open a Python3 (or Sage 9.x) environment and type

```
>>> import reverse
```

We can construct the DFA for base 18 and $a = 7$ as in Figure 1:

```
>>> A = reverse.RSPAutomaton(18,7)
>>> print(A)
RSPAutomaton with base = 18 and a = 7
  States: 7
  Transitions: 7
  Accepting: 2
  Trimmed: False
```

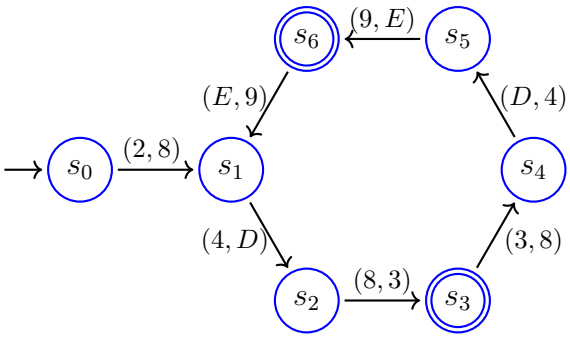

Figure 1. A deterministic finite automaton for $a = 7$ for the base 18.

For a given bound N, we can print the values b with at most N digits such that $(7, b)$ is a reversed sum-product pair for the base 18. Here we take $N = 20$:

```
>>> A.b_values(20)
['2483D8', '2483D9E483D8', '2483D9E483D9E483D8']
```

We can also obtain a regular expression that represents the DFA:

```
>>> print(A.regex())
(248, 3D8)(3D9E48, 3D9E48)* + (2483D9, E483D8)(E483D9, E483D9)*
```

Some basic manipulations have been performed, but we do not claim that this is the simplest way to write this expression.

One benefit of implementing this algorithm is that we can find large examples that would be prohibitive to compute by hand. Let's look at the base 27. The function `construct_automata` will find all DFAs $A_{27,a}$ with at least one accepting state:

```
>>> As = reverse.construct_automata(27,trim=False)
>>> for A in As: print(A.a())
...
2
6
10
26
```

(The a-values are printed in base 10.) Let's focus on the third example: $a = A$ (or 10, in base 10).

```
>>> A = As[2]
>>> print(A)
RSPAutomaton with base = 27 and a = A
  States: 12
  Transitions: 15
  Accepting: 1
  Trimmed: False
```

Support for drawing state diagrams is available if the `graphviz` module is installed:

```
>>> A.graphviz_draw('pretty.png')
```

The output is shown in Figure 2. (See also Figure 5 of [1] for a more symmetric representation.) Looking at Figure 2, we see that the simplest reversed sum-product pair with $a = A$ is $(A, 23DI18B)$. Compare with the output of `A.b_values(10)`.

After constructing a DFA, we may find that there are states s that do not participate in any accepted string — i.e., no path from the initial state to an accepting state passes through s. We can "trim" all such s from the DFA. As an extreme example, the DFA $A_{150,31}$ has 13 states accessible from the initial state, but it has no accepting state. Consequently, we can trim all states but the initial one. The function `construct_automata` automatically trims superfluous states unless the user specifies otherwise.

```
>>> A = reverse.RSPAutomaton(150,31)
>>> print(A)
RSPAutomaton with base = 150 and a = (31,)
  States: 13
  Transitions: 13
  Accepting: 0
  Trimmed: False
>>> A.trim()
>>> print(A)
RSPAutomaton with base = 150 and a = (31,)
  States: 1
  Transitions: 0
  Accepting: 0
  Trimmed: True
```

References

[1] Xander Faber and Jon Grantham. On integers whose sum is the reverse of their product. arXiv:2108.13441 [math.NT], preprint, 2021.

Institute for Defense Analyses, Center for Computing Sciences, 17100 Science Drive, Bowie, MD

E-mail address: awfaber@super.org

Institute for Defense Analyses, Center for Computing Sciences, 17100 Science Drive, Bowie, MD

E-mail address: grantham@super.org

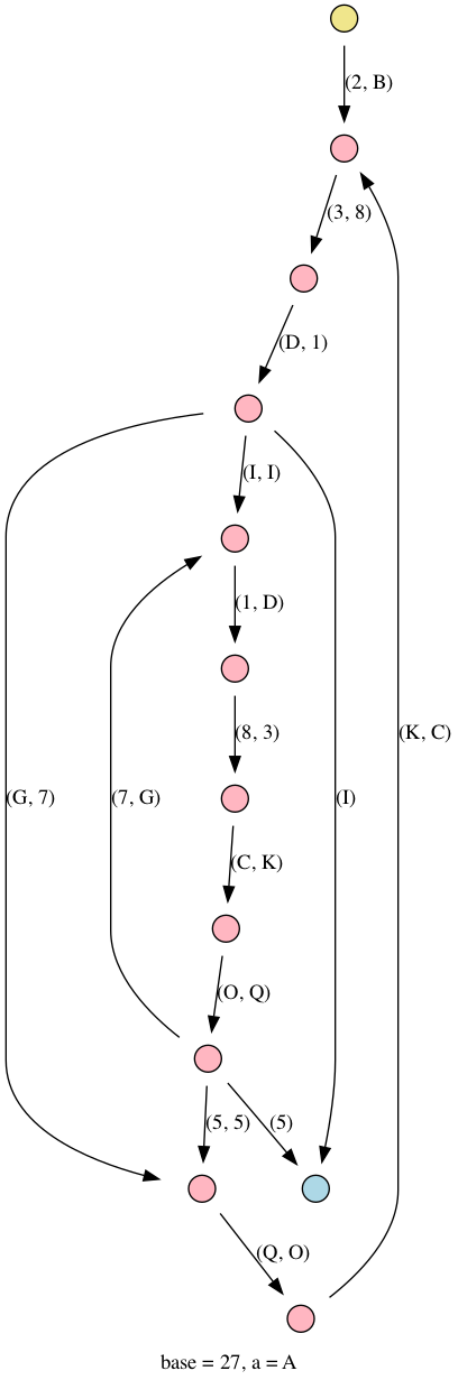

FIGURE 2. A state diagram for the DFA $A_{27,A}$ as output by the method `graphviz_draw()`. The initial state is colored yellow; the accepting state is colored blue.