Automata and Transducers in SageMath

March 31, 2015

In [1]:
```
import sage.combinat.finite_state_machine
sage.combinat.finite_state_machine.FSMOldProcessOutput = False
sage.combinat.finite_state_machine.FSMOldCodeTransducerCartesianProduct = False
```

1 Automata and Transducers in SageMath

1.1 Digit Expansions

- **decimal system**: base 10, digits 0, 1, . . . , 9
- **binary system**: base 2, digits 0, 1
- base 2, digits -1, 0, 1 –> *redundancy*

1.1.1 Non-adjacent Form (NAF)

- no consequtive non-zero digits in expansion
- examples
- 3 = 1 + 2 = (1 1)2 (standard binary) . . . not a NAF
- 3 = -1 + 4 = (-1 0 1)2 . . . a NAF

1.2 Creating a Transducer from Scatch

In [2]:
```
NAF1 = Transducer([('I', 0, 0, None), ('I', 1, 1, None),
                   (0, 0, 0, 0), (0, 1, 1, 0),
                   (1, 0, 0, 1), (1, 2, 1, -1),
                   (2, 1, 0, 0), (2, 2, 1, 0)],
                  initial_states=['I'], final_states=[0],
                  input_alphabet=[0, 1])
NAF = NAF1
NAF
```

Out[2]:
```
Transducer with 4 states
```

In [3]:
```
view(NAF)
```

In [4]:
```
14.digits(base=2)
```

Out[4]:
```
[0, 1, 1, 1]
```

In [5]:
```
NAF(14.digits(base=2))
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-5-6b73441233d1> in <module>()
----> 1 NAF(Integer(14).digits(base=Integer(2)))


/local/data/krenn/sage/6.6.rc1/local/lib/python2.7/site-packages/sage/combinat/finite_state_mach
   3646             if not 'list_of_outputs' in kwargs:
   3647                 kwargs['list_of_outputs'] = False
-> 3648             return self.process(*args, **kwargs)
   3649         raise TypeError("Do not know what to do with that arguments.")
   3650


/local/data/krenn/sage/6.6.rc1/local/lib/python2.7/site-packages/sage/combinat/finite_state_mach
  10791         if (condensed_output and not result or
  10792             not options['full_output'] and result is None):
> 10793                 raise ValueError("Invalid input sequence.")
  10794         if condensed_output and len(result) >= 2:
  10795                 raise ValueError("Found more than one accepting path.")


ValueError: Invalid input sequence.
```

In [ ]: `NAF.process(14.digits(base=2))`

In [6]: `NAF(14.digits(base=2) + [0, 0, 0])`

Out[6]: `[0, -1, 0, 0, 1, 0]`

In [7]: `NAF = NAF.with_final_word_out(0)`

In [8]: `NAF(14.digits(base=2))`

Out[8]: `[0, -1, 0, 0, 1]`

1.3 Calculating the Non-adjacent Form with Less Thinking

In [9]:
```
def NAF_transition(state_from, read):
    if state_from == 'I':
        write = None
        state_to = read
        return (state_to, write)
    current = 2*read + state_from
    if current % 2 == 0:
        write = 0
    elif current % 4 == 1:
        write = 1
    else:
        write = -1
    state_to = (current - write) / 2
    return (state_to, write)
```

```
NAF2 = Transducer(NAF_transition,
                  initial_states=['I'],
                  final_states=[0],
                  input_alphabet=[0, 1]).with_final_word_out(0)

NAF == NAF2
```

Out[9]: True

In [10]: NAF2(14.digits(base=2))

Out[10]: [0, -1, 0, 0, 1]

1.4 A 3rd Construction of the Same Transducer

- (NAF of 2n) = (binary of 3n) – (binary of n)

In [11]:
```
def f(state_from, read):
    current = 3*read + state_from
    write = current % 2
    state_to = (current - write) / 2
    return (state_to, write)

Triple = Transducer(f, input_alphabet=[0, 1],
                    initial_states=[0],
                    final_states=[0]).with_final_word_out(0)
Triple
```

Out[11]: Transducer with 3 states

In [12]: Triple(4.digits(base=2))

Out[12]: [0, 0, 1, 1]

In [13]:
```
Id = Transducer([(0, 0, 0, 0), (0, 0, 1, 1)],
                initial_states=[0], final_states=[0],
                input_alphabet=[0, 1])
```

In [14]: prebuiltId = transducers.Identity([0, 1])

In [15]:
```
Combined_3n_n = Triple.cartesian_product(Id).relabeled()
Combined_3n_n
```

Out[15]: Transducer with 3 states

In [16]: Combined_3n_n(4.digits(base=2))

Out[16]: [(0, 0), (0, 0), (1, 1), (1, None)]

In [17]:
```
def g(read0, read1):
    return ZZ(read0) - ZZ(read1)

Minus = transducers.operator(g, input_alphabet=[None, -1, 0, 1])
```

In [18]: prebuiltMinus = transducers.sub([-1, 0, 1])

In [19]: NAF3 = Minus(Combined_3n_n).relabeled() # compositions of transducers

In [20]: NAF3(14.digits(base=2))

Out[20]: [0, 0, -1, 0, 0, 1]

1.5 An Automaton detecting NAFs

```
In [21]: view(NAF)

In [22]: A = NAF.output_projection()
         A

Out[22]: Automaton with 5 states

In [23]: A([0])

Out[23]: True

In [24]: A([0, -1, 1])

Out[24]: False

In [25]: A([1, 0, 1])

Out[25]: True

In [26]: view(A)

In [27]: A = A.split_transitions()
         A

Out[27]: Automaton with 6 states

In [28]: A.is_deterministic()

Out[28]: False

In [29]: A.determine_alphabets()
         A = A.minimization().relabeled()
         A

Out[29]: Automaton with 5 states

In [30]: A.is_deterministic()

Out[30]: True
```

1.6 Combining Small Transducers to a Larger One: The 3/2-1/2-NAF

```
In [31]: NAF = NAF3
         NAF3n = NAF(Triple)  # composition
         Combined_NAF_3n_n = NAF3n.cartesian_product(NAF).relabeled()
         T = Minus(Combined_NAF_3n_n).relabeled()  # composition
         T

Out[31]: Transducer with 9 states

In [32]: T(14.digits(base=2))

Out[32]: [0, 0, 2, 0, 1, -1, 1]

In [33]: def value(digits):
             return sum(d * 2^(e-2) for e, d in enumerate(digits))
         value(T(14.digits(base=2)))

Out[33]: 14
```

1.7 Again an Alternative Construction

In [34]:
```
def minus(trans1, trans2):
    if trans1.word_in == trans2.word_in:
        return (trans1.word_in,
                trans1.word_out[0] - trans2.word_out[0])
    else:
        raise LookupError

from itertools import izip_longest
def final_minus(state1, state2):
    return [x - y for x, y in
        izip_longest(state1.final_word_out, state2.final_word_out, fillvalue=0)]

Talternative = NAF3n.product_FiniteStateMachine(
                           NAF, minus,
                           final_function=final_minus).relabeled()
Talternative == T
```

Out[34]: True

1.8 Getting a Picture

In [35]:
```
sage.combinat.finite_state_machine.setup_latex_preamble()
latex.mathjax_avoid_list('tikzpicture')
T.set_coordinates({
    0: (-2, 0.75),
    1: (0, -1),
    2: (-6, -1),
    3: (6, -1),
    4: (-4, 2.5),
    5: (-6, 5),
    6: (6, 5),
    7: (4, 2.5),
    8: (2, 0.75)})
T.latex_options(format_letter=T.format_letter_negative,
                accepting_where={
                  0: 'right',
                  1: 'below',
                  2: 'below',
                  3: 'below',
                  4: 60,
                  5: 'above',
                  6: 'above',
                  7: 120,
                  8: 'left'},
                accepting_show_empty=True)

view(latex(T))
latex(T)
```

Out[35]:
```
\begin{tikzpicture}[auto, initial text=, >=latex, accepting text=, accepting/.style=accepting b
\node[state, initial] (v0) at (-2.000000, 0.750000) {$0$};
\path[->] (v0.0.00) edge node[rotate=0.00, anchor=south] {$\$ \mid \varepsilon$} ++(0.00:7ex);
\node[state] (v1) at (0.000000, -1.000000) {$1$};
```

```
\path[->] (v1.270.00) edge node[rotate=450.00, anchor=south] {$\$ \mid \overline{2} 0 1$} ++(27
\node[state] (v2) at (-6.000000, -1.000000) {$2$};
\path[->] (v2.270.00) edge node[rotate=450.00, anchor=south] {$\$ \mid 0 1$} ++(270.00:7ex);
\node[state] (v3) at (6.000000, -1.000000) {$3$};
\path[->] (v3.270.00) edge node[rotate=450.00, anchor=south] {$\$ \mid 0 \overline{1} 1$} ++(27
\node[state] (v4) at (-4.000000, 2.500000) {$4$};
\path[->] (v4.60.00) edge node[rotate=60.00, anchor=south] {$\$ \mid 1$} ++(60.00:7ex);
\node[state] (v5) at (-6.000000, 5.000000) {$5$};
\path[->] (v5.90.00) edge node[rotate=90.00, anchor=south] {$\$ \mid \overline{1} 0 1$} ++(90.0
\node[state] (v6) at (6.000000, 5.000000) {$6$};
\path[->] (v6.90.00) edge node[rotate=90.00, anchor=south] {$\$ \mid \overline{1} 1$} ++(90.00:
\node[state] (v7) at (4.000000, 2.500000) {$7$};
\path[->] (v7.120.00) edge node[rotate=300.00, anchor=south] {$\$ \mid 1 \overline{1} 1$} ++(12
\node[state] (v8) at (2.000000, 0.750000) {$8$};
\path[->] (v8.180.00) edge node[rotate=360.00, anchor=south] {$\$ \mid 0 \overline{2} 0 1$} ++(
\path[->] (v0) edge[loop above] node {$0\mid 0$} ();
\path[->] (v0) edge node[rotate=-41.19, anchor=south] {$1\mid 0$} (v1);
\path[->] (v1) edge node[rotate=360.00, anchor=south] {$0\mid \overline{2}$} (v2);
\path[->] (v1) edge node[rotate=0.00, anchor=south] {$1\mid 2$} (v3);
\path[->] (v2) edge node[rotate=60.26, anchor=south] {$0\mid 0$} (v4);
\path[->] (v2.95.00) edge node[rotate=90.00, anchor=south] {$1\mid 0$} (v5.265.00);
\path[->] (v3.95.00) edge node[rotate=90.00, anchor=south] {$0\mid 0$} (v6.265.00);
\path[->] (v3) edge node[rotate=299.74, anchor=south] {$1\mid 0$} (v7);
\path[->] (v4) edge node[rotate=-41.19, anchor=south] {$0\mid 1$} (v0);
\path[->] (v4) edge node[rotate=308.66, anchor=south] {$1\mid \overline{1}$} (v5);
\path[->] (v5.-85.00) edge node[rotate=90.00, anchor=north] {$0\mid \overline{1}$} (v2.85.00);
\path[->] (v5) edge node[rotate=-14.04, anchor=south] {$1\mid 1$} (v7);
\path[->] (v6.-85.00) edge node[rotate=90.00, anchor=north] {$1\mid 1$} (v3.85.00);
\path[->] (v6) edge node[rotate=14.04, anchor=south] {$0\mid \overline{1}$} (v4);
\path[->] (v7) edge node[rotate=51.34, anchor=south] {$0\mid 1$} (v6);
\path[->] (v7) edge node[rotate=41.19, anchor=south] {$1\mid \overline{1}$} (v8);
\path[->] (v8) edge node[rotate=41.19, anchor=south] {$0\mid 0$} (v1);
\path[->] (v8) edge[loop above] node {$1\mid 0$} ();
\end{tikzpicture}
```

1.9 Weights

In [36]:
```python
def weight(state_from, read):
    write = ZZ(read != 0)
    return (0, write)

Weight = Transducer(weight, input_alphabet=srange(-2, 2+1),
                    initial_states=[0], final_states=[0])
Weight.add_transition((0, 0, None, None))
Weight
```

Out[36]:
```
Transducer with 1 states
```

In [37]:
```python
prebuiltWeight = transducers.weight(srange(-2, 2+1))
```

In [38]:
```python
NAF = NAF2  # reset since modified above
W_binary = Weight(Id)
W_NAF = Weight(NAF)
W_T = Weight(T)
```

In [39]:
```
expansion = 14.digits(base=2)
print "binary" , Id(expansion), W_binary(expansion), sum(W_binary(expansion))
print "NAF", NAF(expansion), W_NAF(expansion), sum(W_NAF(expansion))
print "T", T(expansion), W_T(expansion), sum(W_T(expansion))
```

```
binary [0, 1, 1, 1] [0, 1, 1, 1] 3
NAF [0, -1, 0, 0, 1] [0, 1, 0, 0, 1] 2
T [0, 0, 2, 0, 1, -1, 1] [0, 0, 1, 0, 1, 1, 1] 4
```

1.10 Also Possible: Adjacency Matrices

In [40]:
```
var('y')
def am_entry(trans):
    return y^add(trans.word_out) / 2
W_T.adjacency_matrix(entry=am_entry)
```

Out[40]:
```
[  1/2   1/2     0     0     0     0     0     0     0]
[    0     0 1/2*y 1/2*y     0     0     0     0     0]
[    0     0     0     0   1/2   1/2     0     0     0]
[    0     0     0     0     0     0   1/2   1/2     0]
[1/2*y     0     0     0     0 1/2*y     0     0     0]
[    0     0 1/2*y     0     0     0     0 1/2*y     0]
[    0     0     0 1/2*y 1/2*y     0     0     0     0]
[    0     0     0     0     0     0 1/2*y     0 1/2*y]
[    0   1/2     0     0     0     0     0     0   1/2]
```

1.11 Asymptotic Analysis

In [41]:
```
var('k')
W_binary.asymptotic_moments(k)
```

Out[41]:
```
{'covariance': 1/4*k + Order(1),
 'expectation': 1/2*k + Order(1),
 'variance': 1/4*k + Order(1)}
```

In [42]:
```
W_NAF.asymptotic_moments(k)
```

Out[42]:
```
{'covariance': Order(1),
 'expectation': 1/3*k + Order(1),
 'variance': 2/27*k + Order(1)}
```

In [43]:
```
W_T.asymptotic_moments(k)
```

Out[43]:
```
{'covariance': Order(1),
 'expectation': 5/9*k + Order(1),
 'variance': 44/243*k + Order(1)}
```