WALNUT PROVER

- Written by Hamoon Mousavi
- Available on webside of Jeffrey O. Shallit *https://cs.uwaterloo.ca/~shallit/papers.html*
- Most of examples in this representation come from this webside and the Walnut documentation.

How to run Walnut?

- Walnut is used by typing command in command line.
- Java is needed to run the Walnut.
- We run Walnut by write:

```
cd C:/.../Walnut/bin
>java Main.prover
.
.
.
>exit
```

Commands

List of all commands and their parameters:

```
eval <name> <predicate>
def <name> <predicate>
reg <name> <number system> <regular expression>
reg <name> <alphabet> <regular expression>
load <file name>
exit
```

Command eval

- Eval is basic command.
- It has two parameters: name and predicade.
- Example:

```
eval example "b=a+1";
```

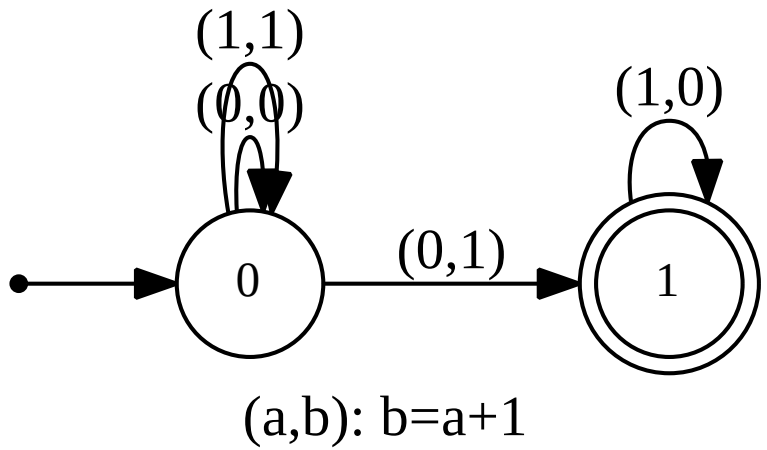
- As an output we get three files in the directory .../Walnut/Result:
  - *example.txt*
  - *example.gv*
  - *example_ log.txt*

Output

table-form description
(example.txt)

msd_2	msd_2		
0	0		
0	0	->	0
0	1	->	1
1	1	->	0
1	1		
1	0	->	1

graph-form description
(example.gv)



(a,b): b=a+1

inputed words:

a = 101101111
b = 101110000

Overlaps in the Thue-Morse infinite word

Theorem

The Thue-Morse word is overlap-free.

Overlaps in the Thue-Morse infinite word

Theorem

The Thue-Morse word is overlap-free.

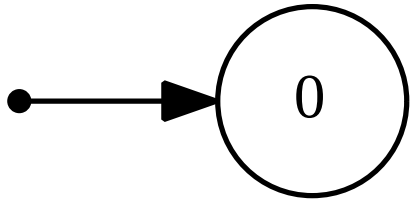
- The following predicade sais "In $\mathbf{t}$ is overelap of order n, starting at position i":
  $(n > 0) \wedge \forall k : k \leq n \Rightarrow T[i + k] = T[i + n + k]$

Overlaps in the Thue-Morse infinite word

Theorem

The Thue-Morse word is overlap-free.

- The following predicade sais "In **t** is overelap of order n, starting at position i":
  $(n > 0) \wedge \forall k : k \leq n \Rightarrow T[i + k] = T[i + n + k]$
- The resultant automaton:



(i,n): n>0 & Ak k<= n => T[i+k]=T[i+k+n]

- The automaton defines an empty language, so there are no overlaps in the Thue-Morse infinite word.

Overlaps in the Thue-Morse infinite word

- In the file *log* we can find some information about the proces of computing the automaton.

```
n>0 has 2 states: 453ms
 k<=n has 2 states: 0ms
  T[(i+k)]=T[((i+k)+n)] has 12 states: 56ms
   (k<=n=>T[(i+k)]=T[((i+k)+n)]) has 25 states: 2ms
    (A k (k<=n=>T[(i+k)]=T[((i+k)+n)])) has 1 states: 9ms
     (n>0\&(A k (k<=n=>T[(i+k)]=T[((i+k)+n)]))) has 1 states: 0ms
total computation time: 583ms
```

- Computation is devided into several steps following the structure of the predicade.

Squares in the Thue-Morse infinite word

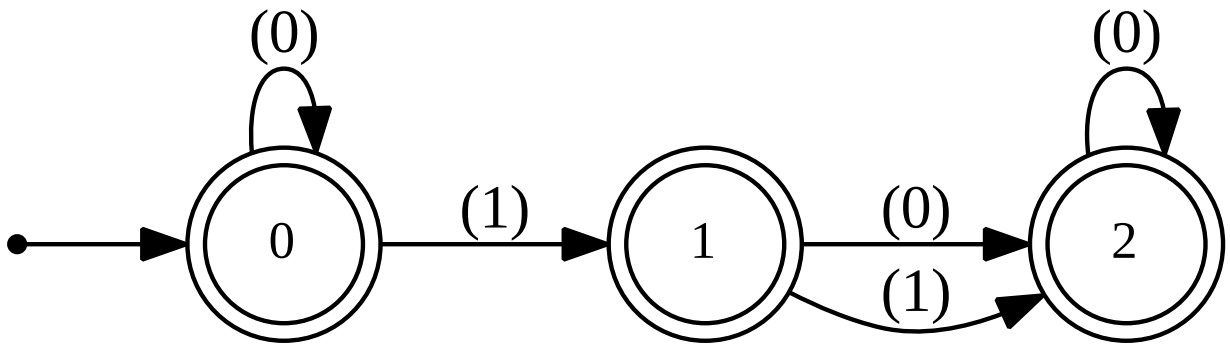
- Of what orders are squares in the Thue-Morse infinitive word?

Squares in the Thue-Morse infinite word

- Of what orders are squares in the Thue-Morse infinitive word?
- Predicade: $\exists i \forall k < n : t[i + k] = t[i + n + k]$;

Squares in the Thue-Morse infinite word

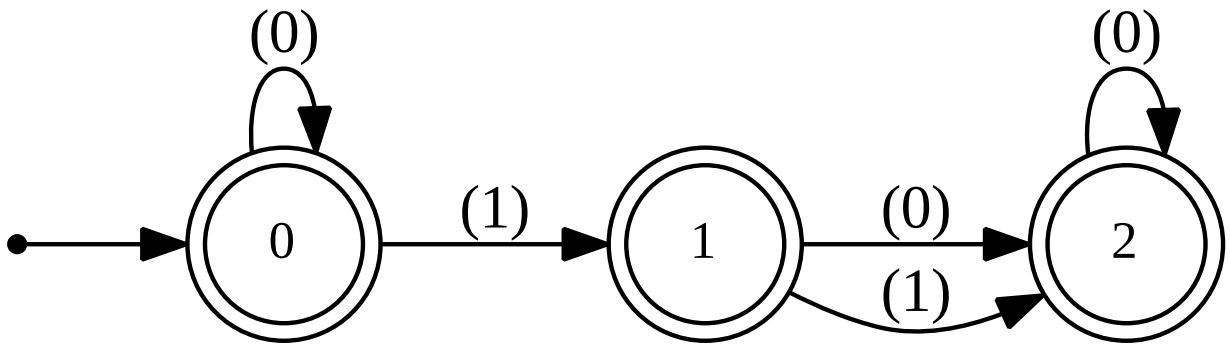
- Of what orders are squares in the Thue-Morse infinitive word?
- Predicade: $\exists i \forall k < n : t[i+k] = t[i+n+k]$;
- Automaton:



(n): Ei Ak k<n => T[i+k]=T[i+n+k]

Squares in the Thue-Morse infinite word

- Of what orders are squares in the Thue-Morse infinitive word?
- Predicade: $\exists i \forall k < n : t[i+k] = t[i+n+k]$;
- Automaton:



(n): Ei Ak k<n => T[i+k]=T[i+n+k]

- In t, there are squares of orders 2^n and 3^n for all $n \in \mathbb{N}^0$.

Comamnd def

- If predicade parameter for *eval* is to dificult, we can break it into more simpler commands, using command *def*.
- The difference between *eval* and *def* is that outcome automaton is saved in an automaton library and can be used later.

Comamnd def

- If predicade parameter for *eval* is to dificult, we can break it into more simpler commands, using command *def*.
- The difference between *eval* and *def* is that outcome automaton is saved in an automaton library and can be used later.
- *Example*
  instead of:

```
eval complicatedEq "a = 3*((b+3)/2)";
```

  we can write:

```
def simpleEq "c =(b+3)/2";
eval simpleEq2 "Ec a=3*c & $simpleEq(c,b)";
```

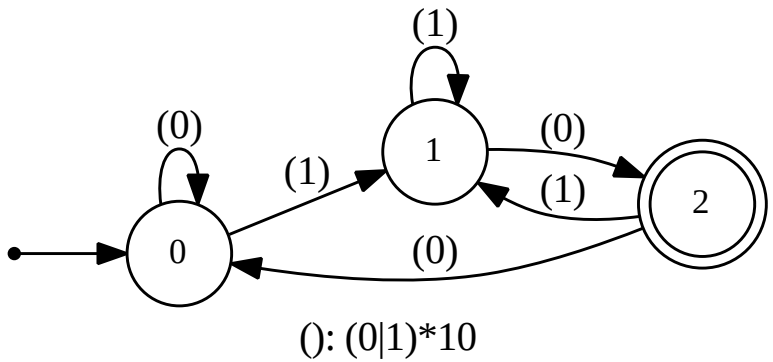
command reg

- Command *reg* provides another way to define an automaton.
- It takes as parametr an language and a regular expression.
- Some properties about numbers can be clearly seen from binary expression.

command reg

- Command *reg* provides another way to define an automaton.
- It takes as parametr an language and a regular expression.
- Some properties about numbers can be clearly seen from binary expression.
- The following automaton cheks if n equals 2 modulo 4.
- Command and automaton:

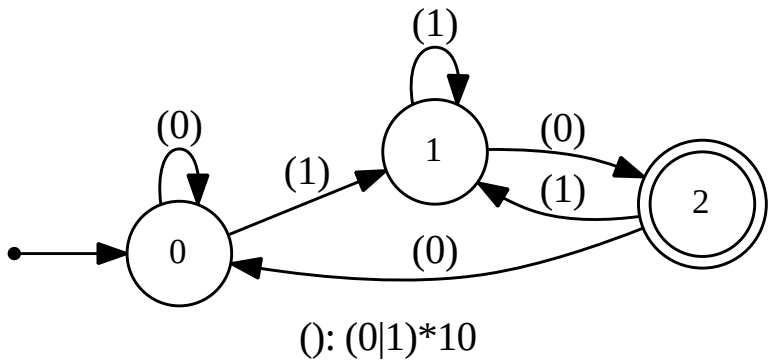
```
reg res2mod4 msd_2"(0|1)*10";
```



command reg

- Command *reg* provides another way to define an automaton.
- It takes as parametr an language and a regular expression.
- Some properties about numbers can be clearly seen from binary expression.
- The following automaton cheks if n equals 2 modulo 4.
- Command and automaton:

```
reg res2mod4 msd_2"(0|1)*10";
```



- Evaluation of reg is also saved in an automaton library.

Fibonacci base

Theorem (Zeckendorf)

Every natural number can be uniquely writen as a sum of Fibonnaci numbers, where no two consecutive Fibonacci numbers are used.

Fibonacci base

Theorem (Zeckendorf)

Every natural number can be uniquely writen as a sum of Fibonnaci numbers, where no two consecutive Fibonacci numbers are used.

- So we can uniquely express every natural number in Fibonacci base.
- Example:

$$85321$$

$$10_{dec} = 10010_{fib} = 8_{dec} + 2_{dec}$$

Example in Fibonacci base

- What orders are squares in the Fibonacci word?
- We definine Fibonacci base by typing *?msd_fib* at the begining of predicade parametr.
- Command:

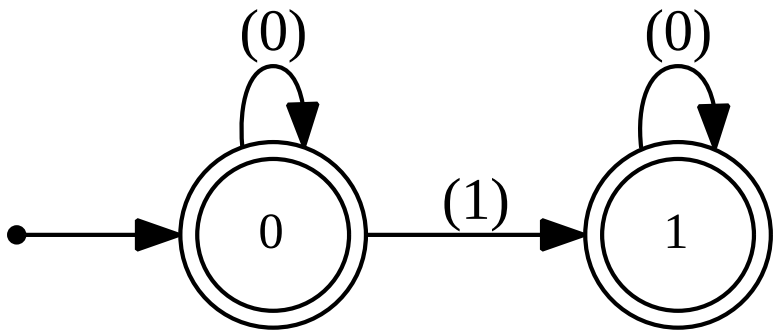
```
orderOfSquareInF "?msd_fib Ei
(Ak k<n => F[i+k]=F[i+n+k])";
```

Example in Fibonacci base

- What orders are squares in the Fibonacci word?
- We definine Fibonacci base by typing *?msd_fib* at the begining of predicade parametr.
- Command:

```
orderOfSquareInF "?msd_fib Ei
(Ak k<n => F[i+k]=F[i+n+k])";
```

- automaton:



(n): ?msd_fib Ei Ak k<n => F[i+k]=F[i+n+k]

- The automaton defines language 0^*10^*, whichs are exactly Fibonacci numbers.

Periodicity of the infinity Fibonacci word

Theorem

The infinite Fibonacci word is not ultimately periodic.

Periodicity of the infinity Fibonacci word

Theorem

The infinite Fibonacci word is not ultimately periodic.

- Intuitively, periodicity would imply, that frequency of 0 is rational. However, we know it is $\frac{1}{\varphi}$, where φ is golden ratio, which is irrational number.

Periodicity of the infinity Fibonacci word

Theorem

The infinite Fibonacci word is not ultimately periodic.

- Intuitively, periodicity would imply, that frequency of 0 is rational. However, we know it is $\frac{1}{\varphi}$, where φ is golden ratio, which is irrational number.
- The predicade, coding the theorem:
  $(p > 0) \wedge \exists n \forall i \geq n : F[i] = F[i + p]$
- Log:

```
p>0 has 3 states: 0ms
p>0 has 2 states: 15ms
 i>=n has 2 states: 0ms
  F[i]=F[(i+p)] has 10 states: 63ms
   (i>=n=>F[i]=F[(i+p)]) has 22 states: 15ms
    (A i (i>=n=>F[i]=F[(i+p)])) has 1 states: 250ms
     (E n (A i (i>=n=>F[i]=F[(i+p)]))) has 1 states: 0ms
      (p>0&(E n (A i (i>=n=>F[i]=F[(i+p)])))) has 1 states: 0ms
total computation time: 32ms
```

Palindromes in the infinity Fibonacci word

Theorem

The infinite Fibonacci word $\mathbf{f}$ *has exactly one palindromic factor of length* n *for* n *even and exactly two palindromic factor of length* n *for* n *odd.*

- Theorem is proved by Walnut in two steps. Firstly we verify the even case, than the odd case.

Palindromes in the infinity Fibonacci word

Theorem

The infinite Fibonacci word **f** *has exactly one palindromic factor of length* n *for* n *even and exactly two palindromic factor of length* n *for* n *odd.*

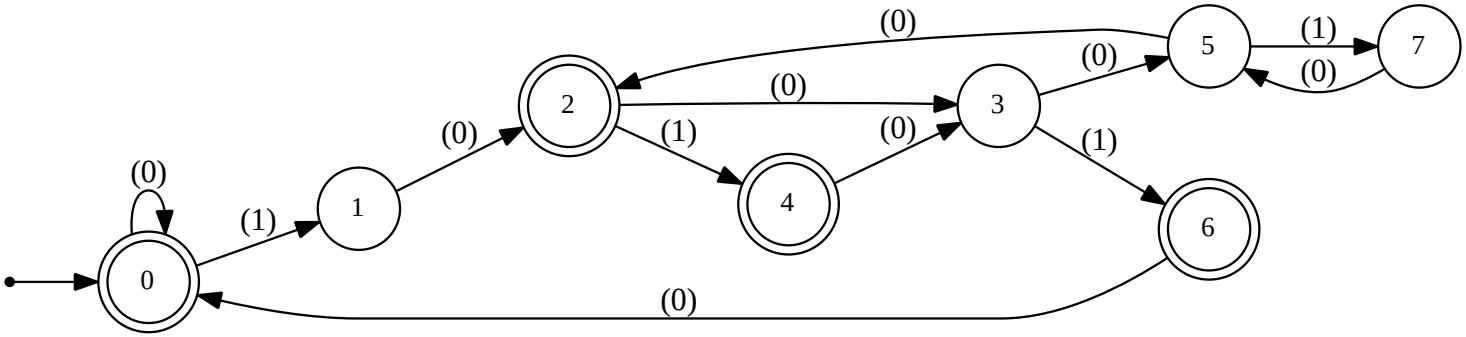
- Theorem is proved by Walnut in two steps. Firstly we verify the even case, than the odd case.
- The following predicade defines n such that f has exactly one palindromic factor of lenght n

$$\exists i(\forall t < n : f[i+t] = f[i+n-1-t]) \wedge$$

$$\forall j(\forall s < n : f[j+s] = f[j+n-1-s]) \Rightarrow$$

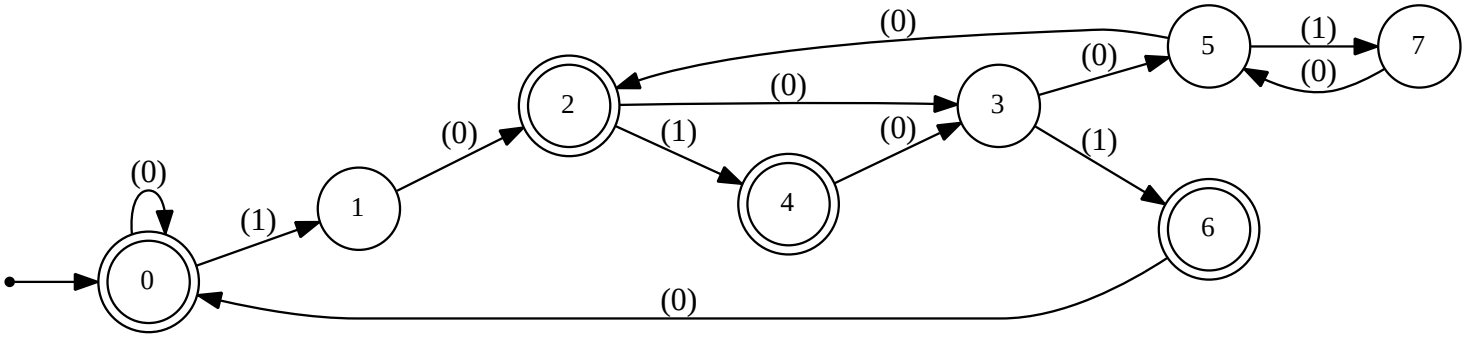
$$\Rightarrow (\forall u < n : f[i+u] = f[j+u])$$

Palindromes in the infinity Fibonacci word



(n): ?msd_fib Ei((At t<n => F[i+t]=F[i+n-1-t])& (Aj (As s<=n => F[j+s]=F[j+n-1-s])=> (Au u<n => F[i+u]=F[j+u])))

Palindromes in the infinity Fibonacci word



(n): ?msd_fib Ei((At t<n => F[i+t]=F[i+n-1-t])& (Aj (As s<=n => F[j+s]=F[j+n-1-s])=> (Au u<n => F[i+u]=F[j+u])))

- We can easily check, that this automaton defines even numbers, by compute automaton of predicade $\exists m(n = 2 * m)$ in Fibonacci base and compare them.
- The case of odd palindromes could be verify in similar way.

Antipalindromes in the infinity Fibonacci word

Definition

Word $x \in \{0,1\}^*$ is antipalindrome if $x = \bar{x}^R$. That is x is invariant under composition of reverse and negation.

Theorem

The only nonempty antipalindromes in $\mathbf{f}$ *are* 01, 10, 0101 *and* 1010.

Antipalindromes in the infinity Fibonacci word

Definition

Word $x \in \{0,1\}^*$ is antipalindrome if $x = \bar{x}^R$. That is x is invariant under composition of reverse and negation.

Theorem

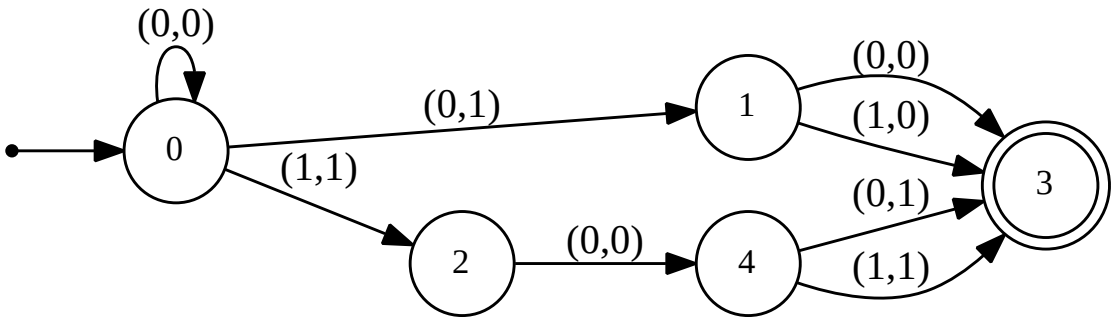
The only nonempty antipalindromes in $\mathbf{f}$ *are* 01, 10, 0101 *and* 1010.

- Following predicade sais: There is antipalindrom $f[i, \ldots, i-1+n]$ and it is the first occurence of such antipalindrom in f.

$$(n > 0) \wedge (\forall j < n : f[i+j] \neq f[i+n-1-j]) \wedge$$

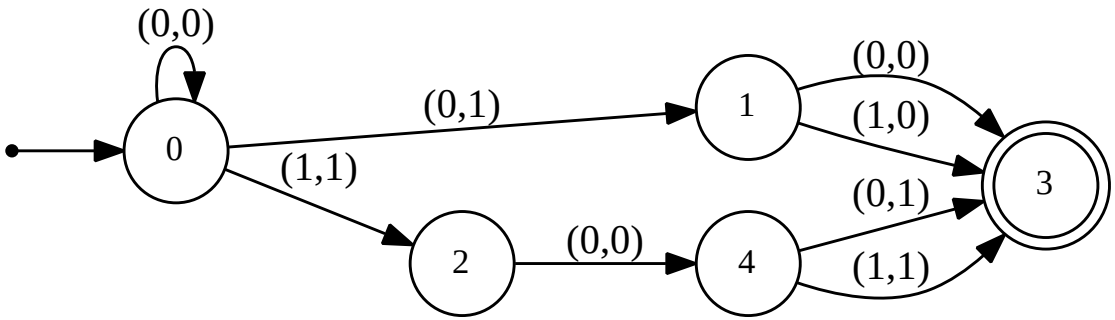
$$\wedge (\forall k < i, \exists j < n : f[k+j] \neq f[i+j])$$

One more example



(i,n): ?msd_fib (n>0) & (Aj j<n => F[i+j]!=F[i+n-1-j]) & (Ak (k<i => (Ej j<n & F[k+j]!= F[i+j])))

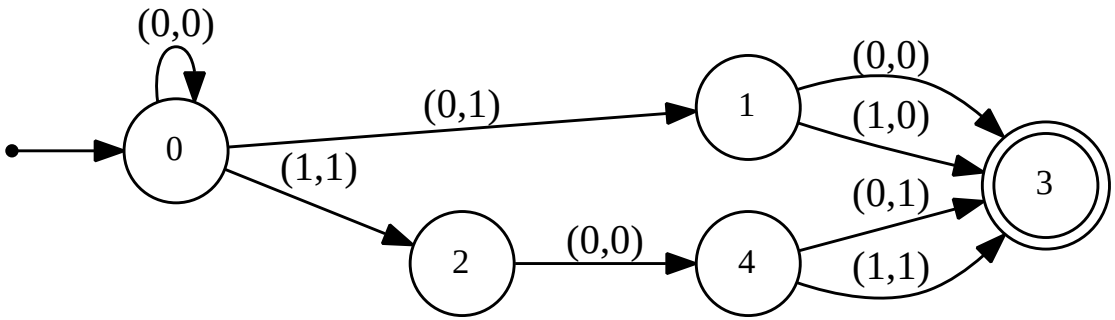
One more example



(i,n): ?msd_fib (n>0) & (Aj j<n => F[i+j]!=F[i+n-1-j]) & (Ak (k<i => (Ej j<n & F[k+j]!= F[i+j])))

- It can be easily seen that the automaton accepts pairs: $(0^*101, 0^*100)$, $(0^*101, 0^*101)$, $(0^*10, 0^*00)$ and $(0^*10, 0^*01)$ in Fibonacci base.

One more example



(i,n): ?msd_fib (n>0) & (Aj j<n => F[i+j]!=F[i+n-1-j]) & (Ak (k<i => (Ej j<n & F[k+j]!= F[i+j])))

- It can be easily seen that the automaton accepts pairs: $(0^*101, 0^*100)$, $(0^*101, 0^*101)$, $(0^*10, 0^*00)$ and $(0^*10, 0^*01)$ in Fibonacci base.
- In decadic base: $(4, 3)$, $(4, 4)$, $(2, 0)$ and $(2, 1)$.

$$0100101001001010010 1\ldots$$
$$0100101001001010010 1\ldots$$