maquinas

Release 0.1.5.14

Ivan Vladimir Meza Ruiz

Jan 24, 2023

CONTENTS

maquinas is a library to build computing machines and grammars. It can be used as a support for the teaching Formal Language Theory.

CHAPTER

ONE

CONTENTS

1.1 Installation

1.1.1 Dependencies

These dependencies will be installed automatically:

- Matplotlib
- graphviz
- IPython
- ordered_set
- Pillow
- TatSu

1.1.2 Create virtual enviroment

To better manage your code create a virtual environment

```
$ mkdir project_mame
$ cd project_name
$ python3 -m venv venv
```

1.1.3 Activate enviroment

Before working on your machines or grammars activate the corresponding enviroment

```
$ . venv/bin/activate
```

1.1.4 Install maquinas

Within the activated environment use pip to install maquinas:

```
$ pip install maquinas
```

For examples of its use, check out the *Quickstart*.

1.2 Quickstart

1.2.1 Creating machines

Regular machines

Deterministic Finete Automaton (DFA)

Create a *DeterministicFiniteAutomaton* from scratch:

```
from maquinas.regular.dfa import DeterministicFiniteAutomaton as DFA

m=DFA(Q=['q_0','q_1'],
                      sigma=['a','b'],
                      q_0='q_0',
                      A=['q_0'],
                      delta=[
                         (('q_0','a'),'q_0'),
                         (('q_1','b'),'q_0'),
                         (('q_1','a'),'q_1'),
                         (('q_0','b'),'q_1'),
                      ])
m.print_summary()
```

Non-Deterministic Finete Automaton (NDFA)

Create a *NonDeterministicFiniteAutomaton* from scratch:

```
from maquinas.regular.ndfa import NonDeterministicFiniteAutomaton as NDFA

m=NDFA(Q=['q_0','q_1'],
                      sigma=['a','b'],
                      q_0='q_0',
                      A=['q_0'],
                      delta=[
                         (('q_0','a'),['q_0']),
                         (('q_1','b'),['q_0']),
                         (('q_1','a'),['q_1']),
                         (('q_0','b'),['q_1']),
                      ])
m.print_summary()
```

Non-Deterministic Finete Automaton with -moves (NDFA-)

Create a *NonDeterministicFiniteAutomaton_epsilon* from scratch:

```
from maquinas.regular.ndfa_e import NonDeterministicFiniteAutomaton_epsilon as NDFA_e

m=NDFA_e(Q=['q_0','q_1'],
                     sigma=['a','b'],
                     q_0='q_0',
                     A=['q_0'],
                     delta=[
                        (('q_0',''),['q_0']),
                        (('q_0','a'),['q_0']),
                        (('q_1','b'),['q_0']),
                        (('q_1','a'),['q_1']),
                        (('q_0','b'),['q_1']),
                     ])
m.print_summary()
```

Context free machines

Pushdown Automaton (PDA)

Create a PushDownAutomatoAutomaton from scratch:

```
from maquinas.contextfree.pda import PushDownAutomaton as PDA

m=PDA(Q=['q_0','q_1','q_2'],
     sigma=['a','b'],
     gamma=['A'],
     q_0='q_0',
     A=['q_2'],
     delta=[
        (('q_0','a','Z0'),[('q_0','AZ0')]),
        (('q_0','a','A'),[('q_0','AA')]),
        (('q_0','b','A'),[('q_1','epsilon')]),
        (('q_1','b','A'),[('q_1','epsilon')]),
        (('q_1','epsilon','Z0'),[('q_2','Z0')]),
     ]
)
m.print_summary()
```

Recursively enumerable machines

Two-Stack Pushdown Automaton (TSPDA)

Create a PushDownAutomatoAutomaton from scratch:

```
from maquinas.recursivelyenumerable.tspda import TwoStackPushDownAutomaton as PDA2

m=PDA2(Q=['q_0','q_1','q_2','q_3'],
     sigma=['a','b'],
     gamma=['A'],
     q_0='q_0',
     A=['q_3'],
     delta=[
        (('q_0','a','Z0','Z0'),[('q_0','AZ0','AZ0')]),
        (('q_0','a','A','A'),[('q_0','AA','AA')]),
        (('q_0','b','A','A'),[('q_1','epsilon','A')]),
        (('q_1','b','A','A'),[('q_1','epsilon','A')]),
        (('q_1','c','Z0','A'),[('q_2','Z0','epsilon')]),
        (('q_2','c','Z0','A'),[('q_2','Z0','epsilon')]),
        (('q_2','epsilon','Z0','Z0'),[('q_3','Z0','Z0')]),
     ]
)
m.print_summary()
```

Turing Machine (TM)

Create a TuringMachine from scratch:

```
from maquinas.recursivelyenumerable.tm import TuringMachine as TM

m=TM(Q=['q_0','q_1','q_2','q_3','q_4'],
     sigma=['a','b'],
     gamma=['X','Y'],
     q_0='q_0',
     A=['q_4'],
     delta=[
        (('q_0','a'),[('q_1','X','R')]),
        (('q_0','Y'),[('q_3','Y','R')]),
        (('q_1','a'),[('q_1','a','R')]),
        (('q_1','b'),[('q_2','Y','L')]),
        (('q_1','Y'),[('q_1','Y','R')]),
        (('q_2','a'),[('q_2','a','L')]),
        (('q_2','Y'),[('q_2','Y','L')]),
        (('q_2','X'),[('q_0','X','R')]),
        (('q_3','Y'),[('q_3','Y','R')]),
        (('q_3','[B]'),[('q_4','[B]','R')]),
     ]
)
m.print_summary()
```

1.2.2 Loading machines

Regular machines

Loading a *DeterministicFiniteAutomaton* or a *NonDeterministicFiniteAutomaton* or a *NonDeterministicFiniteAutomaton_epsilon* from a string:

```
from maquinas.io import load_fa

dfa="""
         |  a  | b  |
-> q0    | q0 | q1 |
   q1  ] | q1 | q0 |
"""

ndfa="""
         |  a    | b  |
-> q0    | q0    | q1 |
   q1  ] | q1    |    |
   q2    | q1,q2 | q1,q2  |
"""

ndfa_e="""
         |  a    | b  | epsilon
-> q0    | q0    |    | q2
   q1  ] | q1    | q0 |
   q2    | q1,q2 | q1,q2  | q2
"""

m=load_fa(dfa)
m.print_summary()
m=load_fa(ndfa)
m.print_summary()
m=load_fa(ndfa_e)
m.print_summary()
```

Converting machines into strings or saving them into a file

It is possible to create a string for the machine:

```
from maquinas.regular.dfa import DeterministicFiniteAutomaton as DFA

m=DFA(Q=['q_0','q_1'],
                     sigma=['a','b'],
                     q_0='q_0',
                     A=['q_0'],
                     delta=[
                        (('q_0','a'),'q_0'),
                        (('q_1','b'),'q_0'),
                        (('q_1','a'),'q_1'),
                        (('q_0','b'),'q_1'),
                     ])
```

(continues on next page)

(continued from previous page)

```
print(m.to_string())
m.save_file('machine.txt')
```

Loading machines from JFLAP

It is possible to load FAs using the 'jff' JFLAP format:

```
from maquinas.io import load_fa

jflap_dfa="""
<structure>
<type>fa</type>
<automaton>
<!--The list of states.-->
<state id="0" name="q0">
<x>83.0</x>
<y>139.0</y>
<initial/>
</state>
<state id="1" name="q1">
<x>235.0</x>
<y>61.0</y>
</state>
<state id="2" name="q2">
<x>545.0</x>
<y>136.0</y>
</state>
<state id="3" name="q3">
<x>391.0</x>
<y>231.0</y>
</state>
<state id="4" name="q4">
<x>287.0</x>
<y>285.0</y>
</state>
<state id="5" name="q5">
<x>504.0</x>
<y>391.0</y>
</state>
<state id="6" name="q6">
<x>161.0</x>
<y>329.0</y>
<final/>
</state>
<!--The list of transitions.-->
<transition>
<from>3</from>
<to>5</to>
<read>a</read>
</transition>
<transition>
```

(continues on next page)

(continued from previous page)

```
<from>1</from>
<to>2</to>
<read>b</read>
</transition>
<transition>
<from>4</from>
<to>5</to>
<read>b</read>
</transition>
<transition>
<from>5</from>
<to>4</to>
<read>b</read>
</transition>
<transition>
<from>3</from>
<to>4</to>
<read>b</read>
</transition>
<transition>
<from>6</from>
<to>1</to>
<read>a</read>
</transition>
<transition>
<from>4</from>
<to>6</to>
<read>a</read>
</transition>
<transition>
<from>1</from>
<to>0</to>
<read>a</read>
</transition>
<transition>
<from>0</from>
<to>2</to>
<read>b</read>
</transition>
<transition>
<from>2</from>
<to>0</to>
<read>b</read>
</transition>
<transition>
<from>6</from>
<to>5</to>
<read>b</read>
</transition>
<transition>
<from>0</from>
<to>6</to>
```

(continues on next page)

(continued from previous page)

```
<read>a</read>
</transition>
<transition>
<from>5</from>
<to>2</to>
<read>a</read>
</transition>
<transition>
<from>2</from>
<to>3</to>
<read>a</read>
</transition>
</automaton>
</structure>
"""

m=load_jflap(jflap_dfa)
m.print_summary()
```

1.2.3 Visualising machines

Saving visualization to files

To save a machine into an image file you can use *save_img()*

```
m.save_img(filename,format="jpg")
```

To create a gif animation of a machine procesing a specific word use *save_gif()*

```
m.save_gif(word,filename)
```

At this moment the filename has to include ".gif" extension.

Visualizing in Notebooks

To display a machine into an interactive notebook use *graph()*

```
m.graph()
```

To simulate a machine into an interactive notebook use `Simulation`

```
from maquinas.simulation import Simulation

s=Simulation(m,word)
s.run()
```

1.2.4 Creating grammars

Tokenization of rules

For now the tokenization of rules is done only at the character level.

Regular grammars (RG)

Create a `RegularGrammar` from a string:

```
from maquinas.regular.rg import RegularGrammar as RG

g=aes_b_ces=RG('S → aS; S → bA; A → ; A → cA') #a*bc*
g.print_summary()
```

Context free grammars (CFG)

Create a `ContextFreeGrammar` from a string:

```
from maquinas.contextfree.cfg import ContextFreeGrammar as CFG

g=CFG("S-> ACB; C-> ACB; C -> AB; A -> a; B->b")
g.print_summary()
```

1.2.5 Parsing and visualazing trees

Parsing a string

To parse a string use `parse()`

```
from maquinas.contextfree.cfg import ContextFreeGrammar as CFG
g=CFG("S-> ACB; C-> ACB; C -> AB; A -> a; B->b")
roots,chart,forest=g.parse("aabcc")
```

The function returns tree elements _roots_ symbols, the _chart_ structure and the _forest_ parse

To verify is the string was part of the language grammar check the roots

```
print(f'Accepts?', "yes" if roots else "no" )
```

To print the chart use `print_chart()`

```
g.print_chart(s,chart,pointers=True)
```

To extract the trees from the forest use `extract_trees()`

```
trees=g.extract_trees(forest)
```

Saving trees images

To save trees into an image file you can use `save_trees_img()`

```
g.save_trees_img(trees,filename=filename)
```

Visualizing in Notebooks

To display the forest grpah use `graph_forest()`

```
m.graph_forest(forest)
```

To display trees use `graph_trees()`

```
m.graph_trees(trees)
```

To display a tree use `graph_tree()`

```
m.graph_tree(trees[0])
```

CHAPTER

TWO

REFERENCE

2.1 Alphabets and Languages

2.1.1 Alphabet

2.1.2 Mapping

2.1.3 Language

2.2 Regular Languages

2.2.1 Deterministic Finite Automaton

class maquinas.regular.dfa.**DeterministicFiniteAutomaton**(*Q=[]*, *sigma=[]*, *q_0=None*, *A=[]*, *delta={}*, *force=False*)

Class for Deterministic Finite Automaton

Parameters

- **Q** – Ordered set of states (default is empty).
- **sigma** – Ordered set of symbols (default is empty).
- **q_0** – Initial state (default None).
- **A** – Set of acceptor states (default is empty).
- **delta** – List of transitions with the form tupla of tupla of q_i and a, and q_f (default is empty).
- **force** (*bool*) – If True and states or symbols do not exists create them (default is False).

acceptor(*q*)

Checks if state is an acceptor state

Parameters
q – State or states

Returns
None

`accepts`(*w*)

Checks if string is accepted

Parameters
w – String

Returns
None

`add_error_state`(*e_label='q_E'*)

Adds a error state (sink state)

Parameters
e_label – Label for the error state

Returns
None

`add_next_state`(*initial=False*)

Adds a state with a infered name based on the number of states q_max. If the name state is already defined it looks the following integer available.

Parameters

- **q** – State or states
- **initial** – Set state as a initial

Returns
Next state generated and integer

`add_state`(*q*, *initial=False*)

Adds a state

Parameters

- **q** – State or states
- **initial** – Set state as a initial

Returns
Indixes of state or states

`add_symbol`(*a*)

Adds a symbol

Parameters
a – Symbol

Returns
Indixes of symbol

`add_transition`(*q_i*, *a*, *q_f*, *force=False*, *update=False*)

Adds a transition

Parameters

- **q_i** – Source state
- **a** – Symbol
- **q_f** – Destination state

Params force
Forces the creation of new symbols or states

Params update
If transition exists, updates it

Returns
None

`autorename`(*start=0*, *avoid=[]*)

Autorenames the states with a patter of q_n, where n is a consicutive integer

Parameters

- **start** – Staring numbering (default is 0)
- **avoid** – List of labelings to avoid (default is empty)

Returns
None

`delta`(*q*, *a*)

Applies delta function

Parameters

- **q** – Source state
- **a** – Symbol

Returns
Destination state

`delta_extended`(*q*, *w*)

Applies delta extended function

Parameters

- **q** – Source state
- **w** – String

Returns
Destination state after processing the full string

`delta_stepwise`(*w*, *q=None*)

Applies a step of delta extended function

Parameters

- **w** – String
- **q** – Source state (default is initial state)

Returns
Tuple with state of precessing at step, consisting of: destination state, procesed symbol and processed string

`get_aceptors`()

Gets aceptors states

Returns
States

`get_initial_state`()

Gets an initial state

Returns
State

`get_transition`(*q*, *a*)

Gets the destintion state or states for state and symbol

Parameters

- **nq** – Source state
- **na** – Symbol

Returns
Destination state or states

`graph`(*q_c={}*, *a_c={}*, *q_prev={}*, *symbols={}*, *states={}*, *format='svg'*, *dpi='60.0'*, *string=None*, ***args*)

Graphs DFA

Parameters

- **q_c** – Set of current states to be highlited (default is empty)
- **a_c** – Set of current symbols to be highlited (default is empty)
- **q_prev** – Set of previos states to be highlited (default is empty)
- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **format** – Format of image (default is svg)
- **dpi** – Resolution of image (default is "60.0")
- **string** – Label of string being analysed (default is None)

Returns
Returns Digraph object from graphviz

`items`()

Iterator over the transitions

Returns
Yeilds a tuple transition

`print_summary`(*symbols={}*, *states={}*, ***args*)

Print a summary of the AF

Parameters

- **symbols** – Replacements of the symbols to print
- **states** – Replacements of the states to print
- **args** – Parameters for the print

Returns
None

`print_transitions`(*w*, *symbols={}*, *states={}*, ***args*)

Print a transition for the string w

Parameters

- **w** – Replacements of the symbols to print
- **states** – Replacements of the states to print

- **args** – Parameters for the print

Returns
None

reachable_states()

Calculate the set of states which are reacheble

Returns
List of reachable states

remove_sink_states()

Remove states that do no go to any place

Returns
None

remove_states(*states*)

Remove states

Parameters
states – List of states to remove

Returns
None

remove_unreachable()

Removes the set of states which are unreacheble

Returns
List of unreachable states

replace_state(*old*, *new*)

Replace a state

Parameters

- **old** – State to be replaced
- **new** – State to replace with

Returns
None

replace_symbol(*old*, *new*)

Replace a symbol

Parameters

- **old** – Symbol to be replaced
- **new** – Symbol to replace with

Returns
None

save_file(*filename='machine.txt'*, *order_Q=None*, *order_sigma=None*)

Saves a file

Parameters

- **filename** – Name of filename (default is machine.txt)
- **order_Q** – Order to print states

- **order_sigma** – Order to print vocabulary

Returns
None

save_gif(*w*, *filename='maquina.gif'*, *symbols={}*, *states={}*, *dpi='90'*, *show=False*, *loop=0*, *duration=500*)

Saves an animation of machine

Parameters

- **w** – String to analysed during animation
- **filename** – Name of gif (default is "maquina.gif")
- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **dpi** – Resolution of image (default is "90.0")
- **show** – In interactive mode return gif
- **loop** – Number of loops in annimation, cero is forever (default is 0)
- **duration** – Duration in msegs among steps (default is 500)

Returns
None or HTML for Ipython

save_img(*filename*, *q_c={}*, *a_c={}*, *q_prev={}*, *symbols={}*, *states={}*, *format='svg'*, *dpi='60.0'*, *string=None*)

Saves machine as an image

Parameters

- **filename** – Filename of image
- **q_c** – Set of current states to be highlited (default is empty)
- **a_c** – Set of current symbols to be highlited (default is empty)
- **q_prev** – Set of previos states to be highlited (default is empty)
- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **format** – Format of image (default is svg)
- **dpi** – Resolution of image (default is "60.0")
- **string** – Label of string being analysed (default is None)

Returns
None

set_aceptors(*A*, *force=False*)

Sets aceptors states

Parameters

- **A** – States
- **force** – If not defined it creates it (default is False)

Returns
None

`set_initial_state`(*q*, *force=False*)

Sets an initial state

Parameters

- **q** – State
- **force** – If not defined it creates it (default is False)

Returns
None

`states`()

Gets states

Returns
States of machine

Return type
list

`stepStatus`(*status*)

Gives a step and calculates new status for Simulation

Parameters
Status – Status

Returns
None

`summary`(*symbols={}*, *states={}*)

Produces a summary of the AF

Parameters

- **symbols** – Replacements of the symbols to print
- **states** – Replacements of the states to print

Returns
List with summary

`symbols`()

Gets symbols

Returns
Symbols of machine

Return type
list

`table`(*symbols={}*, *states={}*, *q_order=None*, *s_order=None*, *empty_transition=''*, *color_final='#32a852'*)

Creates an HTML object for the table of the DFA

Parameters

- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **q_order** – Order to use for state
- **s_order** – Order to use for symbols
- **empty_transition** – Symbol to print for empty transitin (default is “”)

- **color_final** – RGB string for color of final state (default is "#32a852")

Returns
Display object for IPython

`to_string`(*order_Q=None*, *order_sigma=None*)

Creates a string

Parameters

- **order_Q** – Order to print states
- **order_sigma** – Order to print vocabulary

Returns
None

`unreachable_states`()

Calculate the set of states which are unreacheble

Returns
List of unreachable states

2.2.2 Non Deterministic Finite Automaton

`class` maquinas.regular.ndfa.`NonDeterministicFiniteAutomaton`(*Q=[]*, *sigma=[]*, *q_0=None*, *A=[]*, *delta={}*, *force=False*)

Class for Deterministic Finite Automaton

Parameters

- **Q** – Ordered set of states (default is empty).
- **sigma** – Ordered set of symbols (default is empty).
- **q_0** – Initial state (default None).
- **A** – Set of acceptor states (default is empty).
- **delta** – List of transitions with the form tupla of tupla of q_i and a, and q_f (default is empty).
- **force** (*bool*) – If True and states or symbols do not exists create them (default is False).

`acceptor`(*q*)

Checks if state is an acceptor state

Parameters
q – State or states

Returns
None

`accepts`(*w*)

Checks if string is accepted

Parameters
w – String

Returns
None

`add_error_state`(*e_label='q_E'*)

Adds a error state (sink state)

Parameters
`e_label` – Label for the error state

Returns
None

`add_next_state`(*initial=False*)

Adds a state with a infered name based on the number of states q_max. If the name state is already defined it looks the following integer available.

Parameters

- `q` – State or states
- `initial` – Set state as a initial

Returns
Next state generated and integer

`add_state`(*q*, *initial=False*)

Adds a state

Parameters

- `q` – State or states
- `initial` – Set state as a initial

Returns
Indixes of state or states

`add_symbol`(*a*)

Adds a symbol

Parameters
`a` – Symbol

Returns
Indixes of symbol

`add_transition`(*q_i*, *a*, *q_f*, *force=False*, *update=False*)

Adds a transition

Parameters

- `q_i` – Source state
- `a` – Symbol
- `q_f` – Destination state

Params force
Forces the creation of new symbols or states

Params update
If transition exists, updates it

Returns
None

`autorename`(*start=0*, *avoid=[]*)

Autorenames the states with a patter of q_n, where n is a consicutive integer

Parameters

- **start** – Staring numbering (default is 0)
- **avoid** – List of labelings to avoid (default is empty)

Returns
None

`delta`(*q*, *a*)

Applies delta function

Parameters

- **q** – Source state
- **a** – Symbol

Returns
Set of destination state

`delta_extended`(*q*, *w*)

Applies delta extended function

Parameters

- **q** – Source state
- **w** – String

Returns
Set of destination states after processing the full string

`delta_stepwise`(*w*, *q=None*)

Applies a step of delta extended function

Parameters

- **w** – String
- **q** – Source state (default is initial state)

Returns
Tuple with state of precessing at step, consisting of: set of destination states, procesed symbol and processed string

`get_aceptors`()

Gets aceptors states

Returns
States

`get_initial_state`()

Gets an initial state

Returns
State

`get_transition`(*q*, *a*)

Gets the destintion state or states for state and symbol

Parameters

- **nq** – Source state
- **na** – Symbol

Returns
Destination state or states

graph(*q_c={}*, *a_c={}*, *q_prev={}*, *symbols={}*, *states={}*, *format='svg'*, *dpi='60.0'*, *string=None*, ***args*)

Graphs NDFA

Parameters

- **q_c** – Set of current states to be highlited (default is empty)
- **a_c** – Set of current symbols to be highlited (default is empty)
- **q_prev** – Set of previos states to be highlited (default is empty)
- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **format** – Format of image (default is svg)
- **dpi** – Resolution of image (default is "60.0")
- **string** – Label of string being analysed (default is None)

Returns
Returns Digraph object from graphviz

items()

Iterator over the transitions

Returns
Yeilds a tuple transition

print_summary(*symbols={}*, *states={}*, ***args*)

Print a summary of the AF

Parameters

- **symbols** – Replacements of the symbols to print
- **states** – Replacements of the states to print
- **args** – Parameters for the print

Returns
None

print_transitions(*w*, *symbols={}*, *states={}*, ***args*)

Print a transition for the string w

Parameters

- **w** – Replacements of the symbols to print
- **states** – Replacements of the states to print
- **args** – Parameters for the print

Returns
None

`reachable_states`()

Calculate the set of states which are reacheble

Returns
List of reachable states

`remove_sink_states`()

Remove states that do no go to any place

Returns
None

`remove_states`(*states*)

Remove states

Parameters
states – List of states to remove

Returns
None

`remove_unreachable`()

Removes the set of states which are unreacheble

Returns
List of unreachable states

`replace_state`(*old*, *new*)

Replace a state

Parameters

- **old** – State to be replaced
- **new** – State to replace with

Returns
None

`replace_symbol`(*old*, *new*)

Replace a symbol

Parameters

- **old** – Symbol to be replaced
- **new** – Symbol to replace with

Returns
None

`save_file`(*filename='machine.txt'*, *order_Q=None*, *order_sigma=None*)

Saves a file

Parameters

- **filename** – Name of filename (default is machine.txt)
- **order_Q** – Order to print states
- **order_sigma** – Order to print vocabulary

Returns
None

`save_gif`(*w*, *filename='maquina.gif'*, *symbols={}*, *states={}*, *dpi='90'*, *show=False*, *loop=0*, *duration=500*)

Saves an animation of machine

Parameters

- **w** – String to analysed during animation
- **filename** – Name of gif (default is "maquina.gif")
- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **dpi** – Resolution of image (default is "90.0")
- **show** – In interactive mode return gif
- **loop** – Number of loops in annimation, cero is forever (default is 0)
- **duration** – Duration in msegs among steps (default is 500)

Returns

None or HTML for Ipython

`save_img`(*filename*, *q_c={}*, *a_c={}*, *q_prev={}*, *symbols={}*, *states={}*, *format='svg'*, *dpi='60.0'*, *string=None*)

Saves machine as an image

Parameters

- **filename** – Filename of image
- **q_c** – Set of current states to be highlited (default is empty)
- **a_c** – Set of current symbols to be highlited (default is empty)
- **q_prev** – Set of previos states to be highlited (default is empty)
- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **format** – Format of image (default is svg)
- **dpi** – Resolution of image (default is "60.0")
- **string** – Label of string being analysed (default is None)

Returns

None

`set_aceptors`(*A*, *force=False*)

Sets aceptors states

Parameters

- **A** – States
- **force** – If not defined it creates it (default is False)

Returns

None

`set_initial_state`(*q*, *force=False*)

Sets an initial state

Parameters

- **q** – State
- **force** – If not defined it creates it (default is False)

Returns
None

states()

Gets states

Returns
States of machine

Return type
list

stepStatus(*status*)

Gives a step and calculates new status for Simulation

Parameters
Status – Status

Returns
None

summary(*symbols={}*, *states={}*)

Produces a summary of the AF

Parameters

- **symbols** – Replacements of the symbols to print
- **states** – Replacements of the states to print

Returns
List with summary

symbols()

Gets symbols

Returns
Symbols of machine

Return type
list

table(*symbols={}*, *states={}*, *q_order=None*, *s_order=None*, *color_final='#32a852'*, *empty_symbol=''*)

Creates an HTML object for the table of the NDFA

Parameters

- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **q_order** – Order to use for states
- **s_order** – Order to use for symbols
- **color_final** – RGB string for color of final state (default is "#32a852")
- **empy_symbol** – Symbol to be used to display in empty cells (defualt is "")

Returns
Display object for IPython

to_string(*order_Q=None*, *order_sigma=None*)

Creates a string

Parameters

- **order_Q** – Order to print states
- **order_sigma** – Order to print vocabulary

Returns
None

unreachable_states()

Calculate the set of states which are unreacheble

Returns
List of unreachable states

2.2.3 Non Deterministic Finite Automaton with epsilon

class maquinas.regular.ndfa_e.**NonDeterministicFiniteAutomaton_epsilon**(*Q=[]*, *sigma=[]*, *q_0=None*, *A=[]*, *delta=[]*, *force=False*)

Class for Deterministic Finite Automaton

Parameters

- **Q** – Ordered set of states (default is empty).
- **sigma** – Ordered set of symbols (default is empty).
- **q_0** – Initial state (default None).
- **A** – Set of acceptor states (default is empty).
- **delta** – List of transitions with the form tupla of tupla of q_i and a, and q_f (default is empty).
- **force** (*bool*) – If True and states or symbols do not exists create them (default is False).

acceptor(*q*)

Checks if state is an acceptor state

Parameters
q – State or states

Returns
None

accepts(*w*)

Checks if string is accepted

Parameters
w – String

Returns
None

add_error_state(*e_label='q_E'*)

Adds a error state (sink state)

Parameters
e_label – Label for the error state

Returns
None

`add_next_state`(*initial=False*)

Adds a state with a infered name based on the number of states q_max. If the name state is already defined it looks the following integer available.

Parameters

- **q** – State or states
- **initial** – Set state as a initial

Returns
Next state generated and integer

`add_state`(*q*, *initial=False*)

Adds a state

Parameters

- **q** – State or states
- **initial** – Set state as a initial

Returns
Indixes of state or states

`add_symbol`(*a*)

Adds a symbol

Parameters
a – Symbol

Returns
Indixes of symbol

`add_transition`(*q_i*, *a*, *q_f*, *force=False*, *update=False*)

Adds a transition

Parameters

- **q_i** – Source state
- **a** – Symbol
- **q_f** – Destination state

Params force
Forces the creation of new symbols or states

Params update
If transition exists, updates it

Returns
None

`autorename`(*start=0*, *avoid=[]*)

Autorenames the states with a patter of q_n, where n is a consicutive integer

Parameters

- **start** – Staring numbering (default is 0)
- **avoid** – List of labelings to avoid (default is empty)

Returns
None

concat(*other*, *label_a=''*, *label_b=''*)

Concatenates of this NDFA_e with other NDFA-e

Parameters

- **other** – Other NDFA-e
- **label_a** – Label to append to states of this NDFA-e (default is "")
- **label_b** – Label to append to states of other NDFA-e (default is "")

Returns
Returns NDFA resulting of the concatenation of this an another NDFA-e

delta(*q*, *a*)

Applies delta function

Parameters

- **q** – Source state
- **a** – Symbol

Returns
Destination state

delta_extended(*q*, *w*)

Applies delta extended function

Parameters

- **q** – Source state
- **w** – String

Returns
Destination state after processing the full string

delta_stepwise(*w*, *q=None*)

Applies a step of delta extended function

Parameters

- **w** – String
- **q** – Source state (default is initial state)

Returns
Tuple with state of precessing at step, consisting of: destination state, procesed symbol and processed string

expansion_epsilon(*qs*)

Applies expansion by epsilon in a set of states

Parameters
qs – Set of states

Returns
Set of reachable states from qs

`get_aceptors`()

Gets aceptors states

Returns
States

`get_initial_state`()

Gets an initial state

Returns
State

`get_transition`(*q*, *a*)

Gets the destintion state or states for state and symbol

Parameters

- **nq** – Source state
- **na** – Symbol

Returns
Destination state or states

`graph`(*q_c={}*, *a_c={}*, *q_prev={}*, *symbols={}*, *states={}*, *format='svg'*, *dpi='60.0'*, *string=None*, ***args*)

Graphs NDFA_e

Parameters

- **q_c** – Set of current states to be highlited (default is empty)
- **a_c** – Set of current symbols to be highlited (default is empty)
- **q_prev** – Set of previos states to be highlited (default is empty)
- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **format** – Format of image (default is svg)
- **dpi** – Resolution of image (default is “60.0”)
- **string** – Label of string being analysed (default is None)

Returns
Returns Digraph object from graphviz

`items`()

Iterator over the transitions

Returns
Yeilds a tuple transition

`kleene`(*label=''*)

Applies Kleene closure to this NDFA_e

Parameters
label – Label to append to states of this NDFA-e (default is “”)

Returns
Returns NDFA resulting of the application of Kleen closure to this NDFA-e

print_summary(*symbols={}*, *states={}*, ***args*)

Print a summary of the AF

Parameters

- **symbols** – Replacements of the symbols to print
- **states** – Replacements of the states to print
- **args** – Parameters for the print

Returns
None

print_transitions(*w*, *symbols={}*, *states={}*, ***args*)

Print a transition for the string w

Parameters

- **w** – Replacements of the symbols to print
- **states** – Replacements of the states to print
- **args** – Parameters for the print

Returns
None

reachable_states()

Calculate the set of states which are reacheble

Returns
List of reachable states

remove_sink_states()

Remove states that do no go to any place

Returns
None

remove_states(*states*)

Remove states

Parameters
states – List of states to remove

Returns
None

remove_unreachable()

Removes the set of states which are unreacheble

Returns
List of unreachable states

replace_state(*old*, *new*)

Replace a state

Parameters

- **old** – State to be replaced
- **new** – State to replace with

Returns
None

`replace_symbol`(*old*, *new*)

Replace a symbol

Parameters

- **old** – Symbol to be replaced
- **new** – Symbol to replace with

Returns
None

`save_file`(*filename='machine.txt'*, *order_Q=None*, *order_sigma=None*)

Saves a file

Parameters

- **filename** – Name of filename (default is machine.txt)
- **order_Q** – Order to print states
- **order_sigma** – Order to print vocabulary

Returns
None

`save_gif`(*w*, *filename='maquina.gif'*, *symbols={}*, *states={}*, *dpi='90'*, *show=False*, *loop=0*, *duration=500*)

Saves an animation of machine

Parameters

- **w** – String to analysed during animation
- **filename** – Name of gif (default is "maquina.gif")
- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **dpi** – Resolution of image (default is "90.0")
- **show** – In interactive mode return gif
- **loop** – Number of loops in annimation, cero is forever (default is 0)
- **duration** – Duration in msegs among steps (default is 500)

Returns
None or HTML for Ipython

`save_img`(*filename*, *q_c={}*, *a_c={}*, *q_prev={}*, *symbols={}*, *states={}*, *format='svg'*, *dpi='60.0'*, *string=None*)

Saves machine as an image

Parameters

- **filename** – Filename of image
- **q_c** – Set of current states to be highlited (default is empty)
- **a_c** – Set of current symbols to be highlited (default is empty)
- **q_prev** – Set of previos states to be highlited (default is empty)

- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **format** – Format of image (default is svg)
- **dpi** – Resolution of image (default is "60.0")
- **string** – Label of string being analysed (default is None)

Returns
None

set_aceptors(*A*, *force=False*)

Sets aceptors states

Parameters

- **A** – States
- **force** – If not defined it creates it (default is False)

Returns
None

set_initial_state(*q*, *force=False*)

Sets an initial state

Parameters

- **q** – State
- **force** – If not defined it creates it (default is False)

Returns
None

states()

Gets states

Returns
States of machine

Return type
list

stepStatus(*status*)

Gives a step and calculates new status for Simulation

Parameters
Status – Status

Returns
None

summary(*symbols={}*, *states={}*)

Produces a summary of the AF

Parameters

- **symbols** – Replacements of the symbols to print
- **states** – Replacements of the states to print

Returns
List with summary

`symbols`()

Gets symbols

Returns
Symbols of machine

Return type
list

`table`(*symbols={}*, *states={}*, *q_order=None*, *s_order=None*, *color_final='#32a852'*, *empty_symbol=''*)

Creates an HTML object for the table of the DFA

Parameters

- **symbols** – Replacements of the symbols to show (default is empty)
- **states** – Replacements of the states to show (default is empty)
- **q_order** – Order to use for states
- **s_order** – Order to use for symbols
- **color_final** – RGB string for color of final state (default is "#32a852")

Returns
Display object for IPython

`to_string`(*order_Q=None*, *order_sigma=None*)

Creates a string

Parameters

- **order_Q** – Order to print states
- **order_sigma** – Order to print vocabulary

Returns
None

`union`(*other*, *label_a=''*, *label_b=''*)

Unites this NDFA_e with other NDFA-e

Parameters

- **other** – Other NDFA-e
- **label_a** – Label to append to states of this NDFA-e (default is "")
- **label_b** – Label to append to states of other NDFA-e (default is "")

Returns
Returns NDFA resultinf of the union of this an another NDFA-e

`unreachable_states`()

Calculate the set of states which are unreacheble

Returns
List of unreachable states

2.2.4 Regular Grammar

2.2.5 Reductions

maquinas.regular.reductions.**dfa2ndfa**(*dfa*, *rename=True*, *remove_sink=True*)

Reduces a DFA to a NDFA

Parameters

- **dfa** – DFA to reduce
- **raname** – Rename states after reduction
- **remove_sink** – Remove sink states after reduction

maquinas.regular.reductions.**dfa2ndfa_e**(*dfa*, *rename=True*, *remove_sink=True*)

Reduces a DFA to a NDFA-e

Parameters

- **dfa** – DFA to reduce
- **raname** – Rename states after reduction
- **remove_sink** – Remove sink states after reduction

maquinas.regular.reductions.**ndfa2dfa**(*ndfa*, *order=None*, *rename=True*, *remove_sink=True*)

Reduces a NDFA to a DFA

Parameters

- **ndfa** – NDFA to reduce
- **order** – Order for states
- **raname** – Rename states after reduction
- **remove_sink** – Remove sink states after reduction

maquinas.regular.reductions.**ndfa2ndfa_e**(*ndfa*, *rename=True*, *remove_sink=True*)

Reduces a NDFA to a NDFA-e

Parameters

- **dfa** – NDFA to reduce
- **raname** – Rename states after reduction
- **remove_sink** – Remove sink states after reduction

maquinas.regular.reductions.**ndfa_e2dfa**(*ndfa_e*, *rename=True*, *remove_sink=True*)

Reduces a NDFA-e to a DFA

Parameters

- **dfa** – NDFA-e to reduce
- **raname** – Rename states after reduction
- **remove_sink** – Remove sink states after reduction

maquinas.regular.reductions.**ndfa_e2ndfa**(*ndfa_e*, *rename=True*, *remove_sink=True*)

Reduces a NDFA-e to a NDFA

Parameters

- **ndfa_e** – NDFA-e to reduce
- **raname** – Rename states after reduction
- **remove_sink** – Remove sink states after reduction

2.2.6 Minimization

maquinas.regular.minimization.**minimization_hopcroft**(*dfa*, *rename=True*, *remove_sink=True*)

Minimization of a DFA through the Hofproft method, for more info checK:

From http://www-igm.univ-mlv.fr/~berstel/Exposes/2009-06-08MinimisationLiege.pdf

Parameters

- **dfa** – DFA to minimize
- **rename** – If true it will rename states after minimization process
- **remove_sink** – If true it will remove sink states

2.3 Context Free Languages

2.3.1 Push Down Automaton

2.3.2 Context Free Grammar

2.4 Recursively Enumerable Languages

2.4.1 Turing Machine

2.4.2 Two Stack Push Down Automaton

2.5 Simulation

2.5.1 Simulation control

2.6 Collections

2.6.1 Exmaples

2.6.2 Collections

2.7 Input/output

The io module contains...

2.7.1 load_fa

2.7.2 load_pda

2.7.3 load_tspda

2.7.4 load_mt

2.7.5 load_jflap

CHAPTER

THREE

CHANGES

Changelog

3.1 Version 0.1.5.22

- Adds to_string for all
- Minor bugs

3.2 Version 0.1.5.22

- Fixes collections
- Adds to_string
- Documents io
- Documents collections

3.3 Version 0.1.5.21

- Adds collections
- Adds jflap collections
- Adds load_jflap
- Fixes AST tuple instead of list

3.4 Version 0.1.5.18

- Fixes minimization
- Adds save_tree_img

3.5 Version 0.1.5.17

- Adding minimization

3.6 Version 0.1.5.14

- Fixing documentation
- FIxing minor errors with reductions

3.7 Version 0.1.5.12

- Fixes save_file
- Fixes load_file with empty cells
- Add test_io case for save_file

3.8 Version 0.1.5.9

- Adiing Alphabet and Language
- Fixing saving gif

3.9 Version 0.1.5.7

- Visualization for TM

3.10 Version 0.1.5.6

- Full documentation
- Testing
- Loading FAs

3.11 Version 0.1.5.5

- Available through pip

PYTHON MODULE INDEX

m

INDEX

G

I

K

M

N

P

R

S

T

U