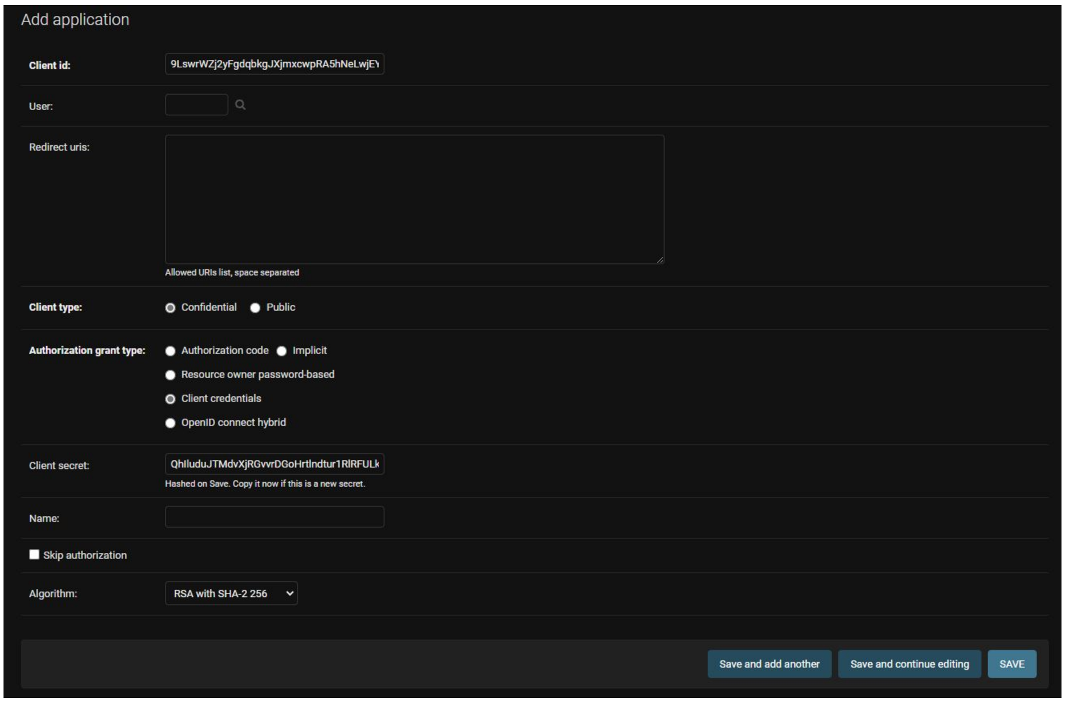
WoFA Backend Documentation

Maurice Herwig

December 2022

Contents


Add application

Client id:	9LswrWZj2yFgdqbkgJXjmxcwpRA5hNeLwjEY
User:	
Redirect uris:	Allowed URIs list, space separated
Client type:	[✓ Confidential] Public
Authorization grant type:	Authorization code Implicit Resource owner password-based [✓ Client credentials] OpenID connect hybrid
Client secret:	QhlluduJTMdvXjRGvvrDGoHrtlndtur1RlRFULk
	Hashed on Save. Copy it now if this is a new secret.
Name:	
☐ Skip authorization	
Algorithm:	RSA with SHA-2 256

Figure 1: Formula to add a new application. Choose as client type confidential and as authorization grant type Client credentials. For the encryption, choose RSA.

1 Introduction

1.1 Project

The wofa-backend[1] project provides endpoints for all the computations provided by the WoFA[2] project to use this project as a microservice in a web architecture. More information about the project and the setup can be found on the GitHub pages.

1.2 Authorization

The WoFa Backend allows only authenticated clients access to this to the services of this backend. To authenticate a client, we must register the new client as an application on the admin site. For this, visit the base_url/admin site, login with the in the setup step created superuser and add a new application. By adding a new application, we see the following formula fill this with in the fig. 1 seen data and save the client_id and client_secret. Important save the values before you save the new application, because after the save the client secret are hashed.

[1]`https://github.com/maurice-herwig/wofa-backend`
[2]`https://github.com/maurice-herwig/wofa`

1.2.1 POST o/ token/

Description:

To get access to all endpoints of this application, we need an access token. That every following request need as a header value. To get this access token, we must send a POST request to the o/token/ endpoint, with the clinet_id and client_secret. Important is that the tokens have a limited lifetime, so that we need a new one when the old one expires. The lifetime of a token can we see in the response at the expires_in field in seconds.

Body:

- client_id: ⟨client_id⟩
- client_secret: ⟨client_secret⟩
- grant_type: client_credentials

Example:

```
curl -X POST 'http://localhost:8000/o/token/'
-d { "client_id": "PWR2JsaskdvpbcGieDZfNjKfhgw9CW2UoRV6zoH5"
     "client_secret": e19e3yxbNejQ7bg7ND6vBic4XEVRxXV41oMrEkDqAYGa...
     "grant_type": "client_credentials"}
```

Response:

```
{
     "access_token": "sTAovipN0p1mpJ0bRKpQ2ncxD6pUSd",
     "expires_in": 3600,
     "token_type": "Bearer",
     "scope": "all"
}
```

2 WoFA

The wofa part of this backend provides endpoints to use the Weight of finite automata (WoFA) python package, to use the functions of the package as a backend service. For more informations about the WoFA project visit the GitHub page[3] or read the paper with the theoretical background[4].

2.1 POST wofa/ weight/

At this endpoint we use the weight function of the WoFA project to compute the weight of one finite automaton over a given alphabet. Is the alphabet not explicit given, we use as alphabet the set of all used transition letter. Furthermore, can we give optional the settings of the parameters η and λ, in the default case the backend calculates avoidable good parameter values for the given automaton. At least we can set variant in this the words in the constant part up to η are weighted, 'words' set all words to the same weight and 'wordLengths' set all word length to the same weight.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)
- eta: int (optional, greater or equal 0)
- lambda: float (optional, in range 0 to 1)
- variant: words | wordLengths

Example:

```
curl -X POST 'http://localhost:8000/wofa/weight/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
          "start_states": [0],
          "transitions": [
               {
                    "start": 0,
                    "letter": "a",
                    "end": 1
               }
          ],
          "acc_states" : [1] }
     "alphabet": ["a", "b"] }
```

[3]https://github.com/maurice-herwig/wofa.git
[4]https://link.springer.com/chapter/10.1007/978-3-031-07469-1_6

Response:

```
{
    "weight": 0.040209982467196б65
}
```

2.2 POST wofa/ weight_dif/

At the endpoint weight_diff we can compute the difference between the languages of the two given automatons. To do this, the backend calculates the weight of the symmetric difference between the two languages. Furthermore, we can add parameters for the calculation of the weight as in the previous endpoint.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- eta: int (optional, greater or equal 0)
- lambda: float (optional, in range 0 to 1)
- variant: words | wordLengths

Example:

```
curl -X POST 'http://localhost:8000/wofa/weight_dif/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton_1": {
            "start_states": ["1","2"],
            "transitions": [
                {
                    "start": "1",
                    "letter": "a",
                    "end": "2"
                },{
                    "start": "2",
                    "letter": "b",
                    "end": "1"
                }
            ],
            "acc_states": ["2"] },
          "automaton_2": {
            "start_states": ["1","2"],
            "transitions": [
                {
                    "start": "1",
                    "letter": "a",
                    "end": "2"
                }

            ],
            "acc_states": []
        }
}
```

Response:

```
{
    "weight": 0.16932351979419896,
    "weight_only_solution": 0.16932351979419896,
    "weight_only_test_object": 0
}
```

2.3 POST wofa/ grading/ weight/

We use the weight of the symmetric difference to calculate a score. So this endpoint gives us a score proposal for a student submission (automaton_2) compared to a solution (automaton_1). The parameter may_points defines the maximum number of points normally given to graduate equivalent submissions to the solution with this point proposal. With the linear_displacement parameter, we provide a parameter to set the rigour of the assessment, taking into account the level of difficulty or experience of the students.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- eta: int (optional, greater or equal 0)
- lambda: float (optional, in range 0 to 1)
- variant: words | wordLengths
- max_points: number
- linear_displacement: number

Example:

```
curl -X POST 'http://localhost:8000/wofa/grading/weight/'
-H Authorization: Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton_1": {
        "start_states": ["1","2"],
        "transitions": [
            {
                "start": "1",
                "letter": "a",
                "end": "2"
            },{
                "start": "2",
                "letter": "b",
                "end": "1"
            }
        ],
        "acc_states": ["2"] },
      "automaton_2": {
        "start_states": ["1","2"],
        "transitions": [
            {
                "start": "1",
                "letter": "a",
                "end": "2"
            }

        ],
        "acc_states": []
    } max_points: 10
    linear_displacement: 5
}
```

Response:

```
{
    "points": 0
}
```

2.4 POST wofa/ grading/ subsets/

A very simple way to calculate a score proposal for student submissions is to determine the subset relationship of the student submission language and the solution language. Here we get half the points for each correct subset relationship.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- max_points: number

Example:

```
curl -X POST 'http://localhost:8000/wofa/grading/subsets/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton_1": {
            "start_states": ["1","2"],
            "transitions": [
                {
                    "start": "1",
                    "letter": "a",
                    "end": "2"
                },{
                    "start": "2",
                    "letter": "b",
                    "end": "1"
                }
            ],
            "acc_states": ["2"] },
           "automaton_2": {
            "start_states": ["1","2"],
            "transitions": [
                {
                    "start": "1",
                    "letter": "a",
                    "end": "2"
                }

            ],
            "acc_states": []
    } max_points: 10
}
```

Response:

```
{
    "points": 5.0
}
```

2.5 POST wofa/ grading/ test_words/

This endpoint needs a list of words that are not in the language of the required language and a list of words that contains in the required language. For each word, the endpoint checks whether the word has been correctly classified by the given automaton. The score is the proportion of correctly classified words multiplied by the maximum score.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- max_points: number
- containing_words: list of strings
- not_included_words: list of strings

Example:

```
curl -X POST 'http://localhost:8000/wofa/grading/test_words/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
            "start_states": ["1","2"],
            "transitions": [
                  {
                        "start": "1",
                        "letter": "a",
                        "end": "2"
                  },{
                        "start": "2",
                        "letter": "b",
                        "end": "1"
                  }
            ],
            "acc_states": ["2"]
          },
      max_points: 10
      "containing_words": ["a", "aa", "abba", "aaaa", "aab"],
      "not_included_words": ["b", "bbb", "bba", "ba", "baa"]
}
```

Response:

```
{
      "points": 5.0
}
```

3 FiniteAutomata

3.1 POST fa/ create/

On this endpoint we can crate a new finite automaton. For the creation, we can choose from four types of automatons. The one_symbol NFA is an automaton, that accepts only the word of length one for a given letter. The univ_symbol automaton accepts all words of length one for the given alphabet. The empty type, create the empty automaton, and the full type the full automaton. For all types we need we the alphabet of the new automaton

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- type: one_symbol | univ_symbol | empty | full
- alphabet: list of strings
- letter: char (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/create/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "type": "one_symbol",
     "alphabet": ["a", "b"],
     "letter": "a"}
```

Response:

```
{
     "result_automaton": {
          "start_states": [0],
          "transitions": [
               {
                    "start": 0,
                    "letter": "a",
                    "end": 1
               }
          ],
          "acc_states" : [1]
     }
}
```

3.2 POST fa/ check/

With the finite automaton check endpoint we can check if the automaton accept a special word. For this we need to choose the method word and fill the word parameter with the word, that we want to check. additionally can we check with empty_word method if the automaton accept the empty word or with the is_empty method, whether the automaton accept the empty language.

Header:m

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)
- method: word | is_empty | empty_word
- word: String (required by the word method.)

Example:

```
curl -X POST 'http://localhost:8000/fa/check/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
            "start_states": [0],
            "transitions": [
                {
                    "start": 0,
                    "letter": "a",
                    "end": 1
                }
            ],
            "acc_states" : [1] }
    "alphabet": ["a", "b"],
    "type": "word"
    "word": "ab" }
```

Response:

```
{
    "result": false
}
```

3.3 POST fa/ longest_run/

At this endpoint we can compute the number of states that can we visited by one run, without that we visit one state twice.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/longest_run/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
          "start_states": [0],
          "transitions": [
               {
                    "start": 0,
                    "letter": "a",
                    "end": 1
               }
          ],
          "acc_states" : [1] }
     "alphabet": ["a", "b"] }
```

Response:

```
{
     "result": 2
}
```

3.4 POST fa/ reachable/

At the finite automaton reachable endpoint, can we for a given automaton compute all from the start states reachable states.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/reachable/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
```

```
            "start_states": [0],
            "transitions": [
                {
                    "start": 0,
                    "letter": "a",
                    "end": 1
                },
                {
                    "start": 0,
                    "letter": "b",
                    "end": 2
                },
            ],
            "acc_states" : [1] }
        "alphabet": ["a", "b"] }
```

Response:

```
{
    "result": [0, 1, 2]
}
```

3.5 POST fa/ productive/

Here can we compute for the given alphabet, the set of states so that exists a path from the state to a accepting state.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/productive/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
            "start_states": [0],
            "transitions": [
                {
                    "start": 0,
                    "letter": "a",
                    "end": 1
                },
                {
                    "start": 0,
                    "letter": "b",
                    "end": 2
                },
            ],
            "acc_states" : [1] }
        "alphabet": ["a", "b"] }
```

Response:

```
{
    "result": [0, 1]
}
```

3.6 POST fa/ sim_pairs/

This endpoint compute all pairs (q, p) of states so that q is simulated by p. That means for every transition from q exists a transition from p to the same state or a simulated equivalence state and q is not a final state or p is a final state too.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/sim_pairs/'
-H Authorization:   Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
            "start_states": [0],
            "transitions": [
                {
                    "start": 0,
                    "letter": "a",
                    "end": 1
                }
            ],
            "acc_states" : [0] }
        "alphabet": ["a", "b"] }
```

Response:

```
{
    "result": [ [0, 1] ]
}
```

3.7 POST fa/ minimize/

To minimize a automaton we can use this endpoint.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/minimize/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
            "start_states": [0],
            "transitions": [
                  {
                        "start": 0,
                        "letter": "a",
                        "end": 1
                  },
                  {
                        "start": 1,
                        "letter": "a",
                        "end": 1
                  },
            ],
            "acc_states" : [0, 1] }
      "alphabet": ["a", "b"] }
```

Response:

```
{
      "result_automaton": {
            "start_states": [0],
            "transitions": [
                  {
                        "start": 0,
                        "letter": "a",
                        "end": 0
                  }
            ],
            "acc_states" : [0]
      }
}
```

3.8 POST fa/ star/

The star endpoint, Surround the set of words in the language with the star operation.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/star/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
            "start_states": [0],
            "transitions": [
                  {
                        "start": 0,
                        "letter": "a",
                        "end": 1
                  }
            ],
            "acc_states" : [1] }
      "alphabet": ["a", "b"] }
```

Response:

```
{
      "result_automaton": {
            "start_states": [0],
            "transitions": [
                  {
                        "start": 0,
                        "letter": "a",
                        "end": 0
                  }
            ],
            "acc_states" : [0]
      }
}
```

3.9 POST fa/ determine/

With the determine endpoint we can compute from a nondeterministic finite automaton the equivalent deterministic finite automaton.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/determine/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
            "start_states": [0],
            "transitions": [
                {
                    "start": 0,
                    "letter": "a",
                    "end": 1
                },
                {
                    "start": 0,
                    "letter": "a",
                    "end": 2
                },
            ],
            "acc_states" : [1, 2] }
      "alphabet": ["a", "b"] }
```

Response:

```
{
      "result_automaton": {
            "start_states": [0],
            "transitions": [
                {
                    "start": 0,
                    "letter": "a",
                    "end": 1
                }
            ],
            "acc_states" : [1]
      }
}
```

3.10 POST fa/ complement/

The complement endpoint compute the complement of the given finite automaton. The result is an automaton that accept only the words the the given one reject.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/complement/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton": {
           "start_states": [0],
           "transitions": [
                {
                     "start": 0,
                     "letter": "a",
                     "end": 1
                }
           ],
           "acc_states" : [1] }
     "alphabet": ["a", "b"] }
```

Response:

```
{
     "result_automaton": {
           "start_states": [0],
           "transitions": [
                {
                     "start": 0,
                     "letter": "a",
                     "end": 1
                },
                {
                     "start": 0,
                     "letter": "b",
                     "end": 2
                },
                {
                     "start": 1,
                     "letter": "a",
                     "end": 2
                },
                {
                     "start": 1,
                     "letter": "b",
                     "end": 2
                },
                {
                     "start": 2,
                     "letter": "a",
                     "end": 2
                },
                {
                     "start": 2,
                     "letter": "b",
                     "end": 2
                }
           ],
           "acc_states" : [0, 2]
     }
}
```

3.11 POST fa/ union/

With the union endpoint, can we compute the automaton that accept the words, that are accepted by at least one of the two given automatons. If the alphabet is not explicitly specified, the union of the used transition letters of the two automates is used here.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/union/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton_1": {
            "start_states": [0],
            "transitions": [
                {
                        "start": 0,
                        "letter": "a",
                        "end": 1
                }
        ],
        "acc_states" : [1] }
    "automaton_2": {
        "start_states": [0],
        "transitions": [
                {
                        "start": 0,
                        "letter": "b",
                        "end": 1
                }
        ],
        "acc_states" : [1] } }
```

Response:

```
{
    "result_automaton": {
        "start_states": [0],
        "transitions": [
                {
                        "start": 0,
                        "letter": "a",
                        "end": 1
                },
                {
                        "start": 0,
                        "letter": "b",
                        "end": 1
                }
        ],
        "acc_states" : [1]
    }
}
```

3.12 POST fa/ concatenate/

The concatenation endpoint, concatenates the two given automaton to one automaton. That means that every word of the resulting automaton can divided into two subwords, so that subword one is accepted by automaton_1 and subword two is accepted by automaton_2.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/concatenate/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton_1": {
          "start_states": [0],
          "transitions": [
               {
                    "start": 0,
                    "letter": "a",
                    "end": 1
               }
          ],
          "acc_states" : [1] }
     "automaton_2": {
          "start_states": [0],
          "transitions": [
               {
                    "start": 0,
                    "letter": "b",
                    "end": 1
               }
          ],
          "acc_states" : [1] } }
```

Response:

```
{
     "result_automaton": {
          "start_states": [0],
          "transitions": [
               {
                    "start": 0,
                    "letter": "a",
                    "end": 1
               },{
                    "start": 1,
                    "letter": "b",
                    "end": 2
               }
          ],
          "acc_states" : [2]
      }
}
```

3.13 POST fa/ intersect/

With this endpoint can we compute the intersection of the languages of the two given automaton. In the language of the resulting automaton are all words that accepted of both input automatons.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/intersect/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton_1": {
          "start_states": [0],
          "transitions": [
               {
                    "start": 0,
                    "letter": "a",
                    "end": 1
               }
          ],
          "acc_states" : [1] }
     "automaton_2": {
          "start_states": [0],
          "transitions": [
               {
                    "start": 0,
                    "letter": "b",
                    "end": 1
               }
          ],
          "acc_states" : [1] } }
```

Response:

```
{
        "result_automaton": {
                "start_states": [],
                "transitions": [],
                "acc_states" : []
        }
}
```

3.14 POST fa/ sym_diff/

This operation computes the symmetrically difference of the two given automatons and return the resulting automaton. The symmetrically difference contains all words that are exactly in one of the two given automatons.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/sym_diff/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton_1": {
                "start_states": [0],
                "transitions": [
                        {
                                "start": 0,
                                "letter": "a",
                                "end": 1
                        },
                        {
                                "start": 0,
                                "letter": "b",
                                "end": 1
                        }
                ],
                "acc_states" : [1] }
        "automaton_2": {
                "start_states": [0],
                "transitions": [
                        {
```

```
                "start": 0,
                "letter": "b",
                "end": 1
            }
        ],
        "acc_states" : [1] } }
```

Response:

```
{
    "result_automaton": {
        "start_states": [0],
        "transitions": [
            {
                "start": 0,
                "letter": "a",
                "end": 1
            }
        ],
        "acc_states" : [1]
    }
}
```

3.15 POST fa/ equivalent/

With the equivalent endpoint can we check if the languages of the two given automatons are equivalent. If the languages are not equivalent, the server return an example word, that are accepted by only one of the automatons.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/equivalent/'
-H Authorization: Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton_1": {
        "start_states": [0],
        "transitions": [
            {
                "start": 0,
                "letter": "a",
                "end": 1
            },
            {
                "start": 0,
                "letter": "b",
                "end": 1
            }
        ],
        "acc_states" : [1] }
    "automaton_2": {
        "start_states": [0],
        "transitions": [
            {
                "start": 0,
                "letter": "b",
                "end": 1
            }
        ],
        "acc_states" : [1] } }
```

Response:

```
{
    "result": false
    "false_word": "a"

}
```

3.16 POST fa/ inclusion/

At this endpoint, can we compute if the language of the automaton_1 a subset of the language of the automaton_2. If this not the case, the server returns additionally a word that are accepted by automaton_1 but not accepted by automaton_2.

Header:

- Authorization: ⟨token_type⟩ ⟨access_token⟩

Body:

- automaton_1: Automaton
- automaton_2: Automaton
- alphabet: list of strings (optional)

Example:

```
curl -X POST 'http://localhost:8000/fa/inclusion/'
-H Authorization:  Bearer 8iKqW5X9OsmrHA5G5fUOmP4LimFI7g
-d { "automaton_1": {
        "start_states": [0],
        "transitions": [
            {
                "start": 0,
                "letter": "a",
                "end": 1
            },
            {
                "start": 0,
                "letter": "b",
                "end": 1
            }
        ],
        "acc_states" : [1] }
    "automaton_2": {
        "start_states": [0],
        "transitions": [
            {
                "start": 0,
                "letter": "b",
                "end": 1
            }
        ],
        "acc_states" : [1] } }
```

Response:

```
{
    "result": false
    "false_word": "a"

}
```