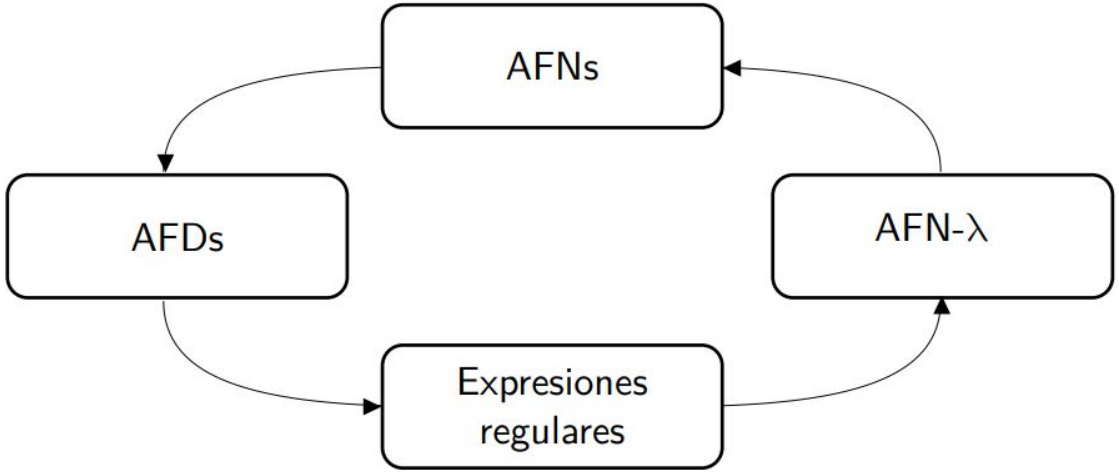
Autómatas y Lenguajes

Práctica 1 - Autómatas Finitos

Objetivos



1. Aceptación de cadenas en AFN
2. De ER a AFN
3. De AFN a AFD
4. Minimización de AFD

Material suministrado

- interfaces.py
- re_parser_interfaces.py
- utils.py
- automaton.py
- automaton_evaluator.py
- re_parser.py
- test_evaluator.py
- test_re_parser.py

No modificar

Ejercicio 1

Cómo ejecutar desde consola

```python
"""Test evaluation of automatas."""
import unittest
from abc import ABC, abstractmethod
from typing import Optional, Type

from automata.automaton import FiniteAutomaton
from automata.automaton_evaluator_sol import FiniteAutomatonEvaluator
from automata.utils import AutomataFormat
```

```
(base) luisfer@toubkal:~/Downloads/p_autlen$ ls
automata
(base) luisfer@toubkal:~/Downloads/p_autlen$
```

```
(base) luisfer@toubkal:~/Downloads/p_autlen$ export PYTHONPATH=$PYTHONPATH:.
```

```
(base) luisfer@toubkal:~/Downloads/p_autlen$ python automata/tests/test_evaluator.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Tipos y mypy

```
(base) luisfer@toubkal:~/Downloads/p_autlen$ pip install mypy
Collecting mypy
  Downloading mypy-0.910-cp38-cp38-manylinux2010_x86_64.whl (22.8 MB)
     |[redacted]| 22.8 MB 461 kB/s
Requirement already satisfied: typing-extensions>=3.7.4 in /home/luisfer/anaconda3/lib/python3.8/site-packages (from mypy) (3.7.4.3)
Requirement already satisfied: mypy-extensions<0.5.0,>=0.4.3 in /home/luisfer/anaconda3/lib/python3.8/site-packages (from mypy) (0.4.3)
Requirement already satisfied: toml in /home/luisfer/anaconda3/lib/python3.8/site-packages (from mypy) (0.10.2)
Installing collected packages: mypy
Successfully installed mypy-0.910
```

```
mypy --strict --strict-equality <ruta_del_proyecto>
```

Importante:

- Variable de entorno `MYPYPATH`
- Puede ser necesario un fichero `__init__.py` dummy (https://github.com/python/mypy/issues/1645)

Ejercicio 1

En `interfaces.py`
(no tocar)

```python
class AbstractFiniteAutomatonEvaluator(
    ABC,
    Generic[_Automaton, _State],
):
    """
    Abstract definition of an automaton evaluator.

    Args:
        automaton: Automaton to evaluate.

    Attributes:
        current_states: Set of current states of the automata.

    """

    automaton: _Automaton
    current_states: AbstractSet[_State]

    def __init__(self, automaton: _Automaton) -> None:
        self.automaton = automaton
        current_states: Set[_State] = {
            self.automaton.initial_state,  # type: ignore[arg-type]
        }
        self._complete_lambdas(current_states)
        self.current_states = current_states
```

Ejercicio 1

En `interfaces.py`
(no tocar)

```python
    @abstractmethod
    def process_symbol(self, symbol: str) -> None:
        """
        Process one symbol.

        Args:
            symbol: Symbol to consume.

        """
        raise NotImplementedError("This method must be implemented.")

    @abstractmethod
    def _complete_lambdas(self, set_to_complete: Set[_State]) -> None:
        """
        Add states reachable with lambda transitions to the set.

        Args:
            set_to_complete: Current set of states to be completed.
        """
        raise NotImplementedError("This method must be implemented.")
```

Ejercicio 1

En `interfaces.py`
(no tocar)

```python
    def process_string(self, string: str) -> None:
        """
        Process a full string of symbols.

        Args:
            string: String to process.

        """
        for symbol in string:
            self.process_symbol(symbol)

    @abstractmethod
    def is_accepting(self) -> bool:
        """Check if the current state is an accepting one."""
        raise NotImplementedError("This method must be implemented.")

    def accepts(self, string: str) -> bool:
        """
        Return if a string is accepted without changing state.

        Note: This function is NOT thread-safe.

        """
        old_states = self.current_states
        try:
            self.process_string(string)
            accepted = self.is_accepting()
        finally:
            self.current_states = old_states

        return accepted
```

Ejercicio 1

En `automaton_evaluator.py`

```python
class FiniteAutomatonEvaluator(
    AbstractFiniteAutomatonEvaluator[FiniteAutomaton, State],
):
    """Evaluator of an automaton."""

    def process_symbol(self, symbol: str) -> None:

        raise NotImplementedError("This method must be implemented.")

    def _complete_lambdas(self, set_to_complete: Set[State]) -> None:

        raise NotImplementedError("This method must be implemented.")

    def is_accepting(self) -> bool:

        raise NotImplementedError("This method must be implemented.")
```

Ejercicio 1

En `automaton_evaluator.py`

1. Calcular los estados a los que se puede transitar desde `current_states` con `symbol`
2. Completar los estados con `_complete_lambdas`
3. Actualizar `current_states` con los nuevos estados
4. Lanzar excepción si `symbol` no está incluido en el alfabeto del autómata

```python
class FiniteAutomatonEvaluator(
    AbstractFiniteAutomatonEvaluator[FiniteAutomaton, State],
):
    """Evaluator of an automaton."""

    def process_symbol(self, symbol: str) -> None:

        raise NotImplementedError("This method must be implemented.")

    def _complete_lambdas(self, set_to_complete: Set[State]) -> None:

        raise NotImplementedError("This method must be implemented.")

    def is_accepting(self) -> bool:

        raise NotImplementedError("This method must be implemented.")
```

Ejercicio 1

1. Calcular el cierre por transiciones λ del conjunto de estados `set_to_complete`

En `automaton_evaluator.py`

```
class FiniteAutomatonEvaluator(
    AbstractFiniteAutomatonEvaluator[FiniteAutomaton, State],
):
    """Evaluator of an automaton."""

    def process_symbol(self, symbol: str) -> None:

        raise NotImplementedError("This method must be implemented.")

    def _complete_lambdas(self, set_to_complete: Set[State]) -> None:

        raise NotImplementedError("This method must be implemented.")

    def is_accepting(self) -> bool:

        raise NotImplementedError("This method must be implemented.")
```

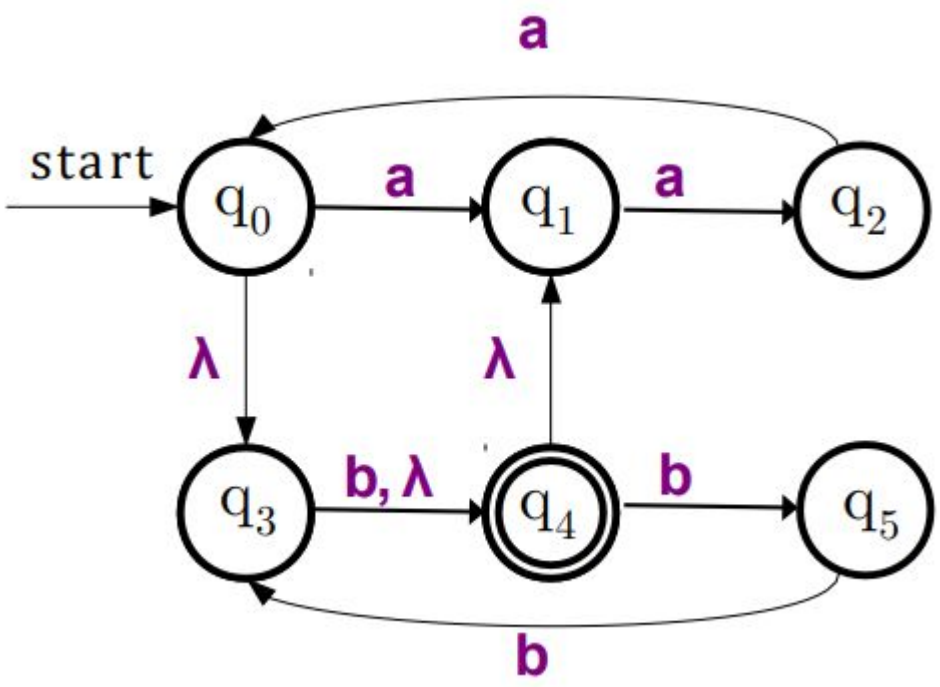
Ejercicio 1

1. Devolver `True` si el conjunto de estados `current_states` contiene algún estado final
2. Devolver `False` en caso contrario

En `automaton_evaluator.py`

```python
class FiniteAutomatonEvaluator(
    AbstractFiniteAutomatonEvaluator[FiniteAutomaton, State],
):
    """Evaluator of an automaton."""

    def process_symbol(self, symbol: str) -> None:

        raise NotImplementedError("This method must be implemented.")

    def _complete_lambdas(self, set_to_complete: Set[State]) -> None:

        raise NotImplementedError("This method must be implemented.")

    def is_accepting(self) -> bool:

        raise NotImplementedError("This method must be implemented.")
```

Ejemplo



Tests

Utils

Planificación

Ejercicio 1	Semana 1
Ejercicio 2	Semana 2
Ejercicio 3	Semanas 3 y 4
Ejercicio 4	Semanas 5 y 6