Autómatas y lenguajes
Práctica 1

Semana 2 - Ejercicio 2

Repasar transparencias Lenguajes Regulares (pg. 1-21)

Expresiones Regulares → AFN-λ

Discusión en conjunto: ¿cómo crear un autómata que acepta...?

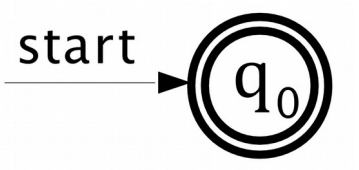
- Lenguaje vacío ({})
- Cadena vacía (“”)
- Un símbolo en particular (“α”)
- La estrella de Kleene (“r_1*”)
- Unión (“$r_1|r_2$”)
- Concatenación (“r_1r_2”)

Expresiones Regulares → AFN-λ

Discusión en conjunto: ¿cómo crear un autómata que acepta...?

- Lenguaje vacío ({})
- Cadena vacía (“”)
- Un símbolo en particular (“α”)
- La estrella de Kleene (“r_1*”)
- Unión (“$r_1|r_2$”)
- Concatenación (“r_1r_2”)

Ejemplo: ¿cuál es?



¿cómo serían los demás?

Expresiones Regulares → AFN-λ

El código: hay que modificar `re_parser.py`

`REParser`

Métodos que completar

```python
"""Conversion from regex to automata."""
from automata.automaton import FiniteAutomaton
from automata.re_parser_interfaces import AbstractREParser


class REParser(AbstractREParser):
    """Class for processing regular expressions in Kleene's syntax."""

    def _create_automaton_empty(
        self,
    ) -> FiniteAutomaton:
        raise NotImplementedError("This method must be implemented.")

    def _create_automaton_lambda(
        self,
    ) -> FiniteAutomaton:
        raise NotImplementedError("This method must be implemented.")

    def _create_automaton_symbol(
        self,
        symbol: str,
    ) -> FiniteAutomaton:
        raise NotImplementedError("This method must be implemented.")

    def _create_automaton_star(
        self,
        automaton: FiniteAutomaton,
    ) -> FiniteAutomaton:
        raise NotImplementedError("This method must be implemented.")

    def _create_automaton_union(
        self,
        automaton1: FiniteAutomaton,
        automaton2: FiniteAutomaton,
```

No necesitan inputs

Input: symbol

Input: FiniteAutomaton

Input: 2 FiniteAutomatons

Expresiones Regulares → AFN-λ

El código: hay que modificar `re_parser.py`

`REParser`

Métodos que completar

```python
"""Conversion from regex to automata."""
from automata.automaton import FiniteAutomaton
from automata.re_parser_interfaces import AbstractREParser


class REParser(AbstractREParser):
    """Class for processing regular expressions in Kleene's syntax."""

    def _create_automaton_empty(
        self,
    ) -> FiniteAutomaton:
        raise NotImplementedError("This method must be implemented.")

    def _create_automaton_lambda(
        self,
    ) -> FiniteAutomaton:
        raise NotImplementedError("This method must be implemented.")

    def _create_automaton_symbol(
        self,
        symbol: str,
    ) -> FiniteAutomaton:
        raise NotImplementedError("This method must be implemented.")

    def _create_automaton_star(
        self,
        automaton: FiniteAutomaton,
    ) -> FiniteAutomaton:
        raise NotImplementedError("This method must be implemented.")

    def _create_automaton_union(
        self,
        automaton1: FiniteAutomaton,
        automaton2: FiniteAutomaton,
```

`AbstractReParser`

```python
class AbstractREParser(ABC):
    """Abstract class for parsing regular expressions in Kleene's syntax."""

    state_counter: int

    def __init__(
        self,
    ) -> None:
        super().__init__()
        self.state_counter = 0

    @abstractmethod
    def _create_automaton_empty(
        self,
    ) -> FiniteAutomaton:
        """
        Create an automaton that accepts the empty language.

        Returns:
            Automaton that accepts the empty language.

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

Atributo `state_counter`

Descripción del método

Expresiones Regulares → AFN-λ

El código: tened en cuenta que contáis con...

`AbstractState`

```python
class AbstractState(ABC):
    """
    Abstract definition of an automaton state.

    Args:
        name: Name of the state.
        is_final: Whether the state is a final state or not.

    """

    name: str
    is_final: bool

    def __init__(self, name: str, *, is_final: bool = False) -> None:
        self.name = name
        self.is_final = is_final
```

`AbstractTransition`

```python
class AbstractTransition(ABC, Generic[_State]):
    """
    Abstract definition of an automaton transition.

    Args:
        initial_state: Initial state of the transition.
        symbol: Symbol consumed in the transition.
            ``None`` for a lambda transition.
        final_state: Final state of the transition.

    """

    initial_state: _State
    symbol: Optional[str]
    final_state: _State

    def __init__(
        self,
        initial_state: _State,
        symbol: Optional[str],
        final_state: _State,
    ) -> None:
        self.initial_state = initial_state
        self.symbol = symbol
        self.final_state = final_state
```

Atributos con los que debéis contar

Expresiones Regulares → AFN-λ

El código: tened en cuenta que contáis con...

`AbstractFiniteAutomaton`

```python
class AbstractFiniteAutomaton(
    ABC,
    Generic[_State, _Transition],
):
    """
    Abstract definition of an automaton.

    Args:
        initial_state: The initial state of the automaton.
        states: Collection of states of the automaton. It is converted to a
            tuple internally.
        symbols: Collection of symbols of the automaton. It is converted to a
            tuple internally.
        transitions: Collection of transitions of the automaton. It is
            converted to a tuple internally.

    """

    initial_state: _State
    states: Sequence[_State]
    symbols: Sequence[str]
    transitions: Sequence[_Transition]

    def __init__(
        self,
        *,
        initial_state: _State,
        states: Collection[_State],
        symbols: Collection[str],
        transitions: Collection[_Transition],
    ) -> None:
```

Hay comprobaciones ya implementadas

```python
if initial_state not in states:
    raise ValueError(
        f"Initial state {initial_state.name} "
        f"is not in the set of states",
    )

for t in transitions:
    for s in (t.initial_state, t.final_state):
        if s not in states:
            raise ValueError(
                f"State {s} from transition {t}"
                f"is not in the set of states",
            )

    if t.symbol is not None and t.symbol not in symbols:
        raise ValueError(
            f"Symbol {t.symbol} from transition {t}"
            f"is not in the set of symbols",
        )

if len(set(states)) != len(states):
    raise ValueError(
        "There are repeated states",
    )
```

Expresiones Regulares → AFN-λ

Tipo y Mypy

- Instalar mypy (usad python 3, no 2!)
  - `pip3 install mypy`
- Realizar comprobaciones
  - `mypy --strict --strict-equality <ruta_proyecto>`
- Ejecutar desde terminal
  - `export PYTHONPATH=$PYTHONPATH:.`
  - `python automata/test/test_evaluator.py`