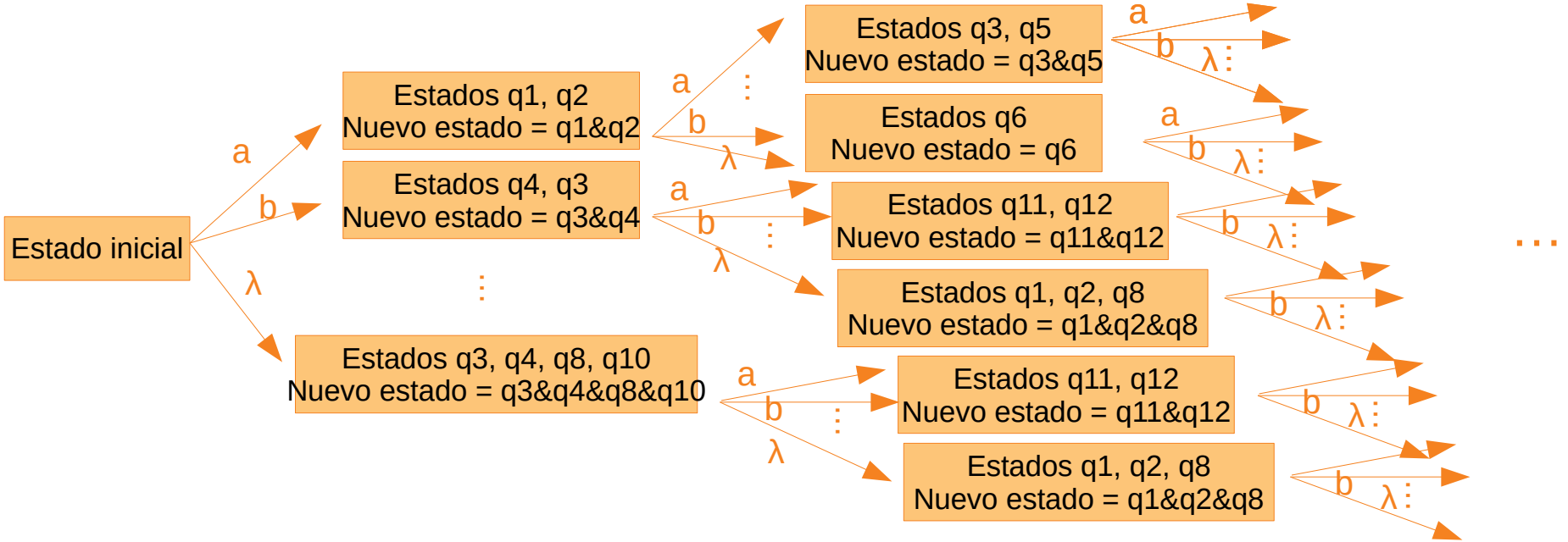
Autómatas y lenguajes
Práctica 1

Semanas 3y4 - Ejercicio 3

Repasar transparencias Lenguajes Regulares (pg. 46-186)

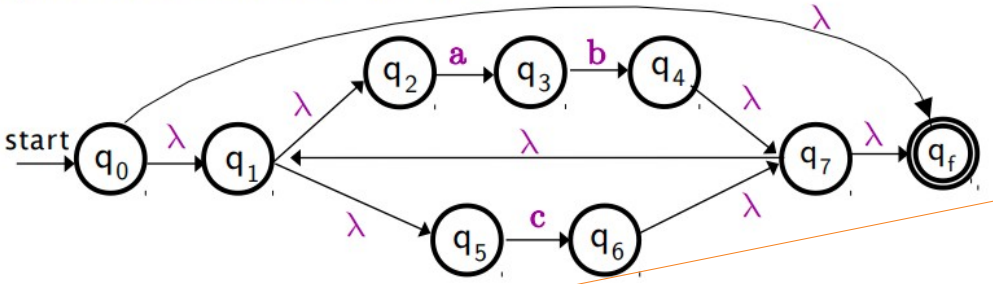
AF no deterministas → deterministas

empezando con el estado inicial, consideramos conjuntos de estados del AFN-λ y computamos a qué estados podemos llegar con cada símbolo desde ellos.



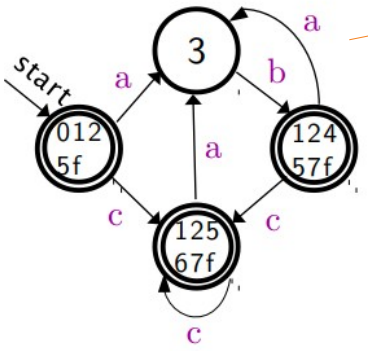
AF no deterministas → deterministas

Ejemplo 5: de AFN-λ a AFD



AFN-λ

AFD equivalente



$(ab + c)^*$

	a	b	c
*{0,1,2,5,f}	{3}	∅	{1,2,5,6,7,f}
{3}	∅	{1,2,4,5,7,f}	∅
*{1,2,5,6,7,f}	{3}	∅	{1,2,5,6,7,f}
*{1,2,4,5,7,f}	{3}	∅	{1,2,5,6,7,f}

Nuevos estados

Ejemplo páginas 119-165

AF no deterministas → deterministas

```python
class FiniteAutomaton(
    AbstractFiniteAutomaton[State, Transition],
):
    """Automaton."""

    def __init__(
        self,
        *,
        initial_state: State,
        states: Collection[State],
        symbols: Collection[str],
        transitions: Collection[Transition],
    ) -> None:
        super().__init__(
            initial_state=initial_state,
            states=states,
            symbols=symbols,
            transitions=transitions,
        )

        # Add here additional initialization code.
        # Do not change the constructor interface.

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

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

El código: hay que modificar `automaton.py` → `FiniteAutomaton`

Constructor

Implementar

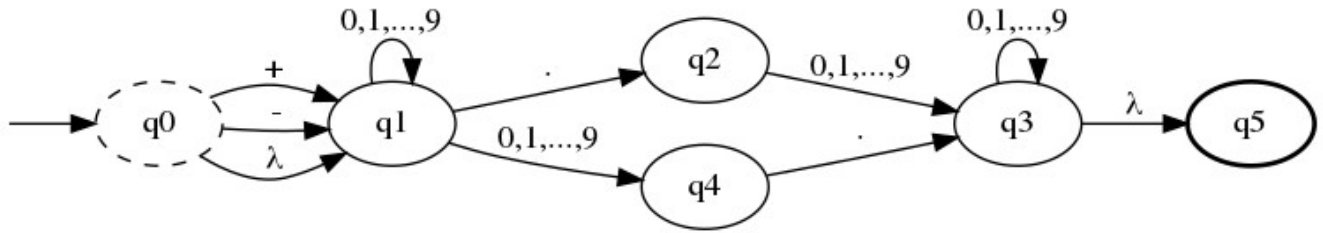
AF no deterministas → deterministas

Casos de prueba para testar (tenéis que programar la prueba):

Reconoce un número decimal con signo opcional

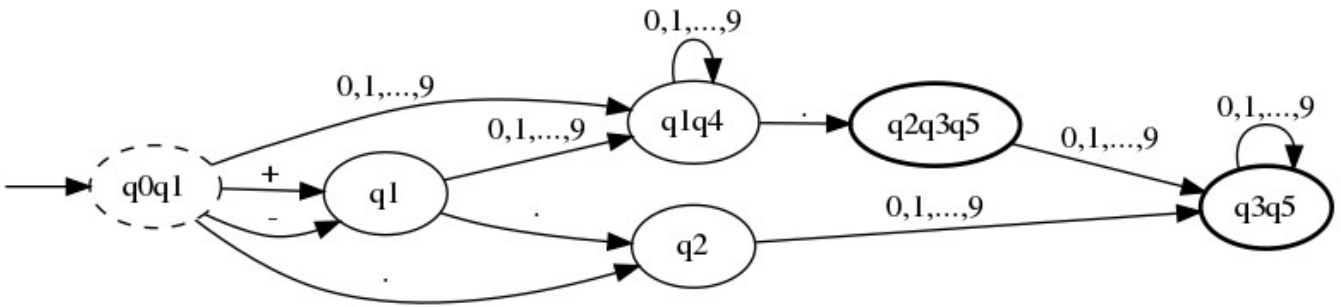
Entrada

AF no determinista



Salida

AF determinista



NOTA: Para los casos de prueba 1 y 2, para que el AFD de salida sea un AFD propiamente dicho (completo), es necesario añadir un estado “sumidero” no final (no representado en el grafo), y añadir hacia él transiciones desde los diferentes estados con los símbolos del alfabeto para los que faltan transiciones en el grafo. La comprobación de isomorfismo, por medio del método “deterministic_automata_isomorphism”, requiere que los AFD sean completos.

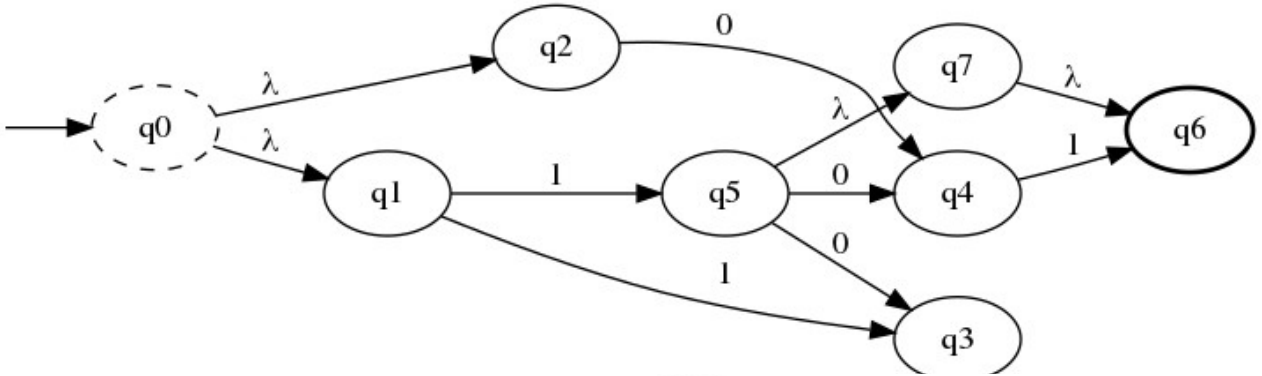
AF no deterministas → deterministas

Casos de prueba para testar (tenéis que programar la prueba):

Reconoce 1, 01, 101, artificiosamente ampliado con lambdas

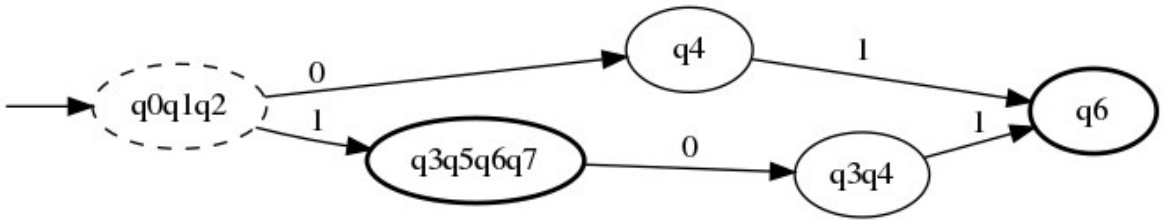
Entrada

AF no determinista



Salida

AF determinista



NOTA: Para los casos de prueba 1 y 2, para que el AFD de salida sea un AFD propiamente dicho (completo), es necesario añadir un estado “sumidero” no final (no representado en el grafo), y añadir hacia él transiciones desde los diferentes estados con los símbolos del alfabeto para los que faltan transiciones en el grafo. La comprobación de isomorfismo, por medio del método “deterministic_automata_isomorphism”, requiere que los AFD sean completos.

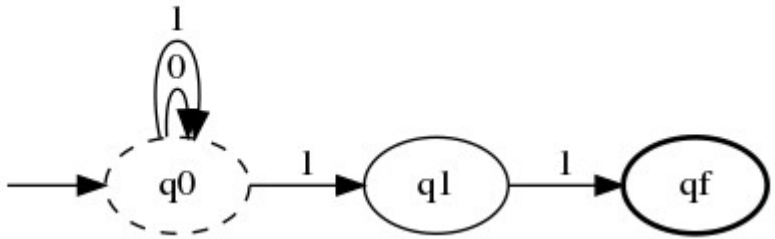
AF no deterministas → deterministas

Casos de prueba para testar (tenéis que programar la prueba):

Reconoce cadena

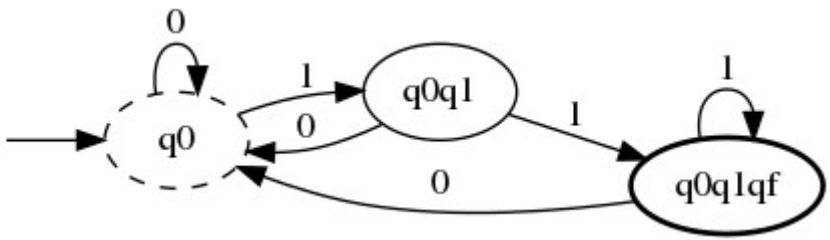
Entrada

AF no determinista



Salida

AF determinista



NOTA: Para los casos de prueba 1 y 2, para que el AFD de salida sea un AFD propiamente dicho (completo), es necesario añadir un estado “sumidero” no final (no representado en el grafo), y añadir hacia él transiciones desde los diferentes estados con los símbolos del alfabeto para los que faltan transiciones en el grafo. La comprobación de isomorfismo, por medio del método “deterministic_automata_isomorphism”, requiere que los AFD sean completos.

AF no deterministas → deterministas

Antes de empezar recordad:

- Podéis **crear más métodos** en `FiniteAutomaton` si os ayuda, o necesitáis repetir alguna operación muchas veces
- Mirad si os puede **ayudar** algo de lo que habéis programado en **apartados anteriores**.
- **Haced tantos unittest como necesitéis** para aseguraros de que vuestro algoritmo funciona correctamente en todas las situaciones posibles