1 Toolkit Documentation

1.1 `qfa_toolkit.quantum_finite_state_automaton`

`Superposition`

A type-alias of `ndarray[Any, cdouble]` for superposition.
Implicitly, it represents an n-dimensional vector whose norm is 1, where n is the number of states.

`States`

A type-alias of `ndarray[Any, bool]` for a set of states.
Implicitly, it represents a set of states with one-hot encoding.

`Transitions`

A type-alias of `ndarray[Any, cdouble]` for a set of transition matrices.
Implicitly, it is an (m, n, n)-shaped array and each (n, n)-shaped subarray is `Transition`.

`Transition`

A type-alias of `ndarray[Any, cdouble]` for a transition matrix.
Implicitly, its shape is (n, n) and it denotes a unitary transition matrix.

`Observable`

A type-alias of `ndarray[Any, bool]` for an observable.
Implicitly, it is a $(3, n)$-shaped array and each $(n,)$-shaped subarray is `States` which denotes accepting, rejecting or non-halting states, respectively.

`TotalState`

Class for total state in quantum finite-state automaton

- Constructor
  - ```
    TotalState(
        superposition_or_list: Superposition | list[complex],
        acceptance: float = 0,
        rejection: float = 0
    )
    ```
- Properties
  - `superposition: Superposition`
- Methods
  - `initial(states: int) -> TotalState`
    ```

  - `measure_by(observable: Observable) -> TotalState`
  - `apply(unitary: Transition) -> TotalState`
  - `to_tuple() -> tuple[Superposition, float, float]`
  - `normalized() -> TotalState`
    * Normalize the total state so that its norm is 1.

`QuantumFiniteStateAutomatonBase`

Abstract class for quantum finite-state automaton.

- Properties
  - `alphabet: int`
  - `states: int`
  - `start_of_string: int`
  - `end_of_string: int`
- Methods
  - `initial_transition: Transition`
  - `final_transition: Transition`
  - `observable: Observable`
  - ```
    process(
        w: list[int],
        total_state: TotalState | None
    ) -> TotalState
    ```
  - `step(total_state: TotalState, c: int) -> TotalState`
  - `string_to_tape(string: list[int]) -> list[int]`

`MeasureOnceQuantumFiniteStateAutomaton`

Class for Measure-once quantum finite-state automaton.
Subclass of `QuantumFiniteStateAutomatonBase`.
Abbreviation: `MOQFA`

- Constructor
  - ```
    MeasureOnceQuantumFiniteStateAutomaton(
        transitions: Transitions,
        accepting_states: States
    )
    ```
- Properties
  - `accepting_states: States`
  - `rejecting_states: States`

- `observable: Observable`

- Methods
  - `word_transition(w: list[int]) -> Transition`
  - `union(other: MOQFA) -> MOQFA`
    - Complement of Hadamard product of complements.
  - `intersection(other: MOQFA) -> MOQFA`
    - Hadamard product.
  - `complement(other: MOQFA) -> MOQFA`
  - ```
    linear_combination(
        *moqfas: MOQFA,
        coefficients: list[float] | None = None
    ) -> MOQFA
    ```
    - Class method.
    - Default coefficients are $1/N$ for N MOQFAs.
  - `word_quotient(w: list[int]) -> MOQFA`
  - `inverse_homomorphism(phi: list[list[int]]) -> MOQFA`
  - `to_measure_many_quantum_finite_state_automaton() -> MMQFA`
  - `to_without_final_transition() -> MMQFA`
  - `to_without_initial_transition() -> MMQFA`
  - `to_real_valued() -> MMQFA`
  - `to_bilinear() -> MMQFA`
  - `to_stochastic() -> MMQFA`
  - `counter_example(other: MOQFA) -> list[int] | None`
    - Return a string w such that $f_M(w) \neq f_N(w)$.
    - If there is no such string, then return None.
  - `equivalence(other: MOQFA) -> bool`

`MeasureManyQuantumFiniteStateAutomaton`

Class for Measure-many quantum finite-state automaton.
Subclass of `QuantumFiniteStateAutomatonBase`.
Abbreviation: `MMQFA`

- Constructor
  - ```
    MeasureManyQuantumFiniteStateAutomaton(
        transitions: Transitions,
        accepting_states: States,
        rejecting_states: States
    )
    ```

- Properties
  - `accepting_states: States`
  - `rejecting_states: States`
  - `halting_states: States`
  - `non_halting_states: States`
  - `observable: Observable`
- Methods
  - `word_transition(w: list[int]) -> Transition`
  - `union(other: MMQFA) -> MMQFA`
    - Complement of Hadamard product of complements.
  - `intersection(other: MMQFA) -> MMQFA`
    - Hadamard product.
  - `complement(other: MMQFA) -> MMQFA`
  - ```
    linear_combination(
        *moqfas: MMQFA,
        coefficients: list[float] | None = None
    ) -> MMQFA
    ```
    - Class method.
    - Default coefficients are $1/N$ for N MMQFAs.
  - `is_end_decisive() -> bool`
  - `is_co_end_decisive() -> bool`
  - `to_real_valued() -> MMQFA`

1.2 qfa_toolkit.quantum_finite_state_automaton_language

`QuantumFiniteStateAutomatonLanguageBase`

Abstract class for quantum finite-state automaton language

- Constructor

```
QauntumFiniteStateAutomatonLanguageBase(
    quantum_finite_state_automaton:
        QuantumFiniteStateAutomatonBase,
    strategy: RecognitionStrategy
) -> QuantumFiniteStateAutomatonLanguageBase
```

- Properties
  - `alphabet: int`
```

- `start_of_string: int`
- `end_of_string: int`

- Methods
  - `__contains__(w: list[int]) -> bool`
  - `enumerate() -> Iterator[list[int]]`
    - Enumerate all strings in the language
  - `enumerate_length_less_than_n(n: int) -> Iterator[list[int]]`
  - `enumerate_length_n(n: int) -> Iterator[list[int]]`

`MeasureManyQuantumFiniteStateAutomatonLanguage`

Class for MOQFL.
Subclass of `QuantumFiniteStateAutomatonLanguageBase`.
Abbreviation: `MOQFL`

- Methods
  - `intersection(other: MOQFL) -> MOQFL`
  - `union(other: MOQFL) -> MOQFL`
  - `word_quotient(w: MOQFL) -> MOQFL`
  - `inverse_homomorphism(phi: list[list[int]]) -> MOQFL`
  - `from_modulo(n: int) -> MOQFL`
    - Class method.
  - ```
    from_modulo_prime(
        p: int,
        copy_num: int = 0,
        seed: int = 42,
    ) -> MOQFL
    ```
    - Class method.
    - If `copy_num` is 0, use $8 \log p$ instead.

`MeasureManyQuantumFiniteStateAutomatonLanguage`

Class for MMQFL.
Subclass of `QuantumFiniteStateAutomatonLanguageBase`.
Abbreviation: `MMQFL`

- Methods
  - `intersection(other: MMQFL) -> MMQFL`
  - `union(other: MMQFL) -> MMQFL`
    ```

  - `from_unary_finite(`
    `    ks: list[int],`
    `    params: Optional[tuple[float, float]] = None`
    `) -> MMQFL`
    - Class method.
  - `from_unary_singleton(`
    `    k: list[int],`
    `    params: Optional[tuple[float, float]] = None`
    `) -> MMQFL`
    - Class method.

1.3 qfa_toolkit.quantum_finite_state_automaton_language

`QiskitQuantumFiniteStateAutomaton`

Abstract class for Qiskit interface.

- Propeties
  - `qfa: QauntumFiniteStateAutomatonBase`
  - `size: int`
    - The number of qubits
  - `mapping: dict[int, int]`
    - Mapping from QFA states to qubit basis
  - `reverse_mapping: dict[int, int]`
    - Mapping from qubit basis to QFA states
  - `alphabet: int`
  - `states: int`
  - `defined_states: set[int]`
  - `undefined_states: set[int]`
- Methods
  - `get_circuit_for_string(w: list[int]) -> None`

`QiskitMeasureManyQuantumFiniteStateAutomaton`

Class for generating Qiskit circuits from MMQFAs.
Subclass of `QiskitQuantumFiniteStateAutomaton`.

- Constructor
  - `QiskitMeasureManyQuantumFiniteStateAutomaton(`
    `    qfa: MMQFA,`
    `    use_entropy_mapping: bool = True`
    `)`

- Properties
  - `halting_states: int`

`QiskitMeasureOnceQuantumFiniteStateAutomaton`

Class for generating Qiskit circuits from MOQFAs.
Subclass of `QiskitQuantumFiniteStateAutomaton`.

- Constructor
  - `QiskitMeasureOnceQuantumFiniteStateAutomaton(`
    `qfa: MOQFA,`
    `use_entropy_mapping: bool = True`
    `)`