A disjoint set data structure will be used to represent equivalence classes of states. Specifically, for n disjoint sets, the following operations will be defined:

1. `Make(`i`)`, a set containing only i will be created
2. `Find(`i`)`, returns a (consistent) identifying element for S_i, the set containing i.
3. `Union`(i, j), creates a new set S_k such that $S_k = S_i \cup S_j$ and sets S_i, S_j are destroyed

The general incremental minimization function for complete, deterministic SFAs is given below:

```
Function IncrementalMinimize(M = (Q, A, q^0, Δ, F)):
    for q ∈ Q do
        Make(q)
    end
    neq = {Normalize(p, q) | p ∈ F, q ∈ Q \ F} // initializes set of pairs
        known not equal
    for p ∈ Q do
        for q ∈ {x ∈ Q | x > p} do
            if (p, q) ∈ neq then
                continue
            end
            if Find(p) = Find(q) then
                continue
            end
            equiv, path = ∅
            if Equiv-p (p,q) then
                for ((p′, q′) ∈ equiv) do
                    Union(p’,q’)
                end
            else
                for (p′, q′) ∈ path do
                    neq = neq ∪ {(p′, q′)}
                end
            end
        end
    end
    return JoinStates(M)
```

Note that we assume there exists an ordering on Q (i.e. $p < q$ makes sense for all $p, q \in Q$). This can be done easily by labeling each state with a unique positive integer. `Normalize` takes a pair (p, q) as input and reorders it so that the first element is less than the second. `JoinStates` merges the states that share the same equivalence class (i.e. share the same disjoint set).

`Equiv-p` is defined below and is the only major difference between the SFA and DFA case. It returns True on (p, q) iff p, q are equivalent. It uses *equiv* to track pairs of states found to be equivalent and *path* to keep track of the path through the set of pairs of states.

Function `Equiv-p`(p, q):
- **if** $(p, q) \in neq$ **then**
  - **return** False
- **end**
- **if** $(p, q) \in path$ **then**
  - `// cycle of equivalences found`
  - **return** True
- **end**
- $path = path \cup \{(p, q)\}$
- $Out_p = \{\varphi \in \Psi_A \mid \exists p', (p, \varphi, p') \in \Delta\}$ `// All outgoing predicates of` p
- $Out_q = \{\psi \in \Psi_A \mid \exists q', (q, \psi, q') \in \Delta\}$
- **while** $Out_p \cup Out_q \neq \varnothing$ **do**
  - Let $a \in [\![(\bigvee_{\varphi \in Out_p} \varphi) \wedge (\bigvee_{\psi \in Out_q} \psi)]\!]$ `// Always satisfiable while in the loop assuming` M `is complete`
  - $(p', q') =$ `Normalize`(`Find`$(\delta(p, a))$, `Find`$(\delta(q, a))$)
  - **if** $p' \neq q'$ **and** $(p', q') \notin equiv$ **then**
    - $equiv = equiv \cup \{(p', q')\}$
    - **if not** `Equiv-p`(p', q') **then**
      - **return** False
    - **else**
      - $path = path \setminus \{(p', q')\}$
    - **end**
  - **end**
  - Let $\varphi \in Out_p$ with $a \in [\![\varphi]\!]$
  - Let $\psi \in Out_q$ with $a \in [\![\psi]\!]$
  - $Out_p = Out_p \setminus \{\varphi\} \cup \{\varphi \wedge \neg\psi\}$
  - $Out_q = Out_q \setminus \{\psi\} \cup \{\psi \wedge \neg\varphi\}$
- **end**
- $equiv = equiv \cup \{(p, q)\}$
- **return** True

This algorithm was adapted for SFAs from "Incremental DFA minimisation" by Almeida, Moreira, Reis. In particular, the only content really unique to the SFA algorithm is contained in `Equiv-p`. Note that certain optimization steps have been left out for simplicity.