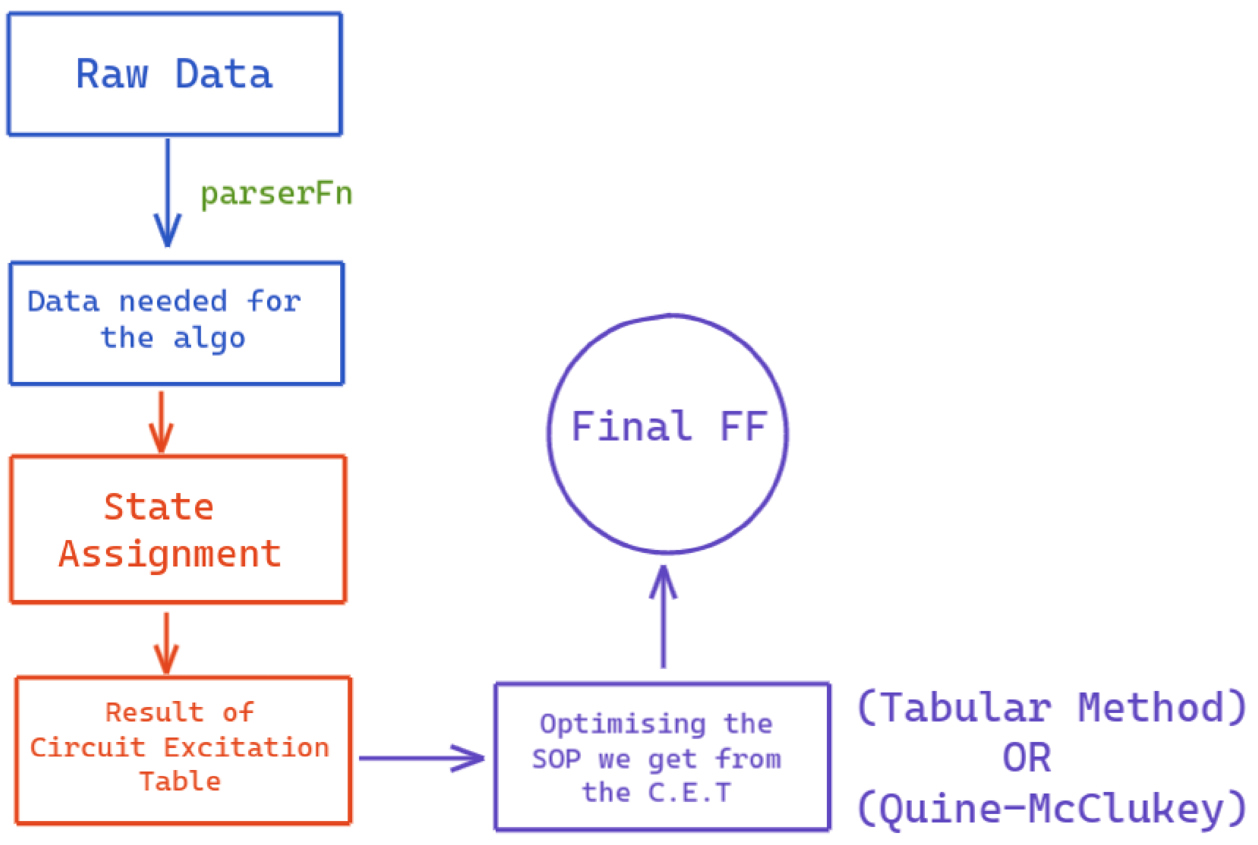
README.md

Name	Arnab Sen
Roll	510519006
Date	3-March-2021

Algorithm

Here is the basic flow-diagram of the Algorithm.



Raw Data
parserFn
Data needed for the algo
State Assignment
Result of Circuit Excitation Table
Optimising the SOP we get from the C.E.T
(Tabular Method) OR (Quine-McClukey)
Final FF

After parsing of the XML data (which is covered here and here) we need to get the final Flip Flop values. To simplify the process, the algorithm is broken down into three steps:

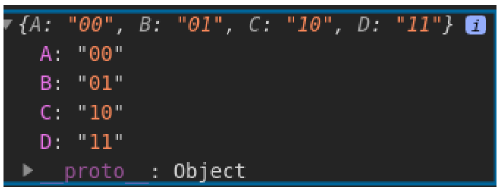
1. State Assignment

The states that the user will provide will have character sets like 'A', 'B', 'C', .etc. But to move ahead with the algorithm we need to assign binary values to each state. This is done using this function:

```
function stateAssign(data) {
    let maxBits = data['gates_count'];
    let maxValue = Math.pow(2, maxBits);
    let states = [];
    for (i = 0; i < maxValue; i += 1) {
        states.push(i.toString(2).padStart(maxBits, '0'));
    }
    let presentStates = data['states'];
    presentStates.sort();
    let stateMap = {};
    presentStates.forEach((present, index) => {
        stateMap[present] = states[index];
    });
    console.log(stateMap);

    return stateMap;
}
```

The function works this way:

1. Calculates the number of gates needed let it be `c`.
2. Generates a squence of binary numbers each having `c` bits.
3. Sorts the character state set in ascending order and assigns binary values correspondingly, creating a map like object.

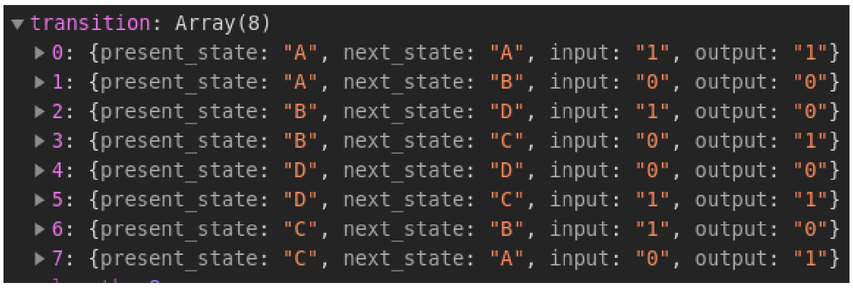


```
▼{A: "00", B: "01", C: "10", D: "11"} i
   A: "00"
   B: "01"
   C: "10"
   D: "11"
 ▶ __proto__: Object
```

After we get the state names' binary correspondence, we have to update the transition states as well. So for that we have:

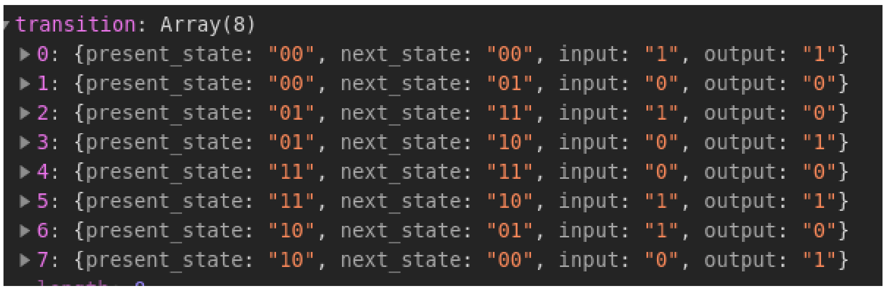
```
function updateStates(data, stateMap) {
    let newData = { ...data };
    newData['transition'] = data['transition'].map((trans) => {
        return {
            ...trans,
            present_state: stateMap[trans.present_state],
            next_state: stateMap[trans.next_state],
        };
    });

    console.log(newData);
    return newData;
}
```

PREVIOUSLY



```
▼transition: Array(8)
  ▶0: {present_state: "A", next_state: "A", input: "1", output: "1"}
  ▶1: {present_state: "A", next_state: "B", input: "0", output: "0"}
  ▶2: {present_state: "B", next_state: "D", input: "1", output: "0"}
  ▶3: {present_state: "B", next_state: "C", input: "0", output: "1"}
  ▶4: {present_state: "D", next_state: "D", input: "0", output: "0"}
  ▶5: {present_state: "D", next_state: "C", input: "1", output: "1"}
  ▶6: {present_state: "C", next_state: "B", input: "1", output: "0"}
  ▶7: {present_state: "C", next_state: "A", input: "0", output: "1"}
```

FINALLY



```
▼transition: Array(8)
  ▶0: {present_state: "00", next_state: "00", input: "1", output: "1"}
  ▶1: {present_state: "00", next_state: "01", input: "0", output: "0"}
  ▶2: {present_state: "01", next_state: "11", input: "1", output: "0"}
  ▶3: {present_state: "01", next_state: "10", input: "0", output: "1"}
  ▶4: {present_state: "11", next_state: "11", input: "0", output: "0"}
  ▶5: {present_state: "11", next_state: "10", input: "1", output: "1"}
  ▶6: {present_state: "10", next_state: "01", input: "1", output: "0"}
  ▶7: {present_state: "10", next_state: "00", input: "0", output: "1"}
```

2. Output of the Circuit Excitation Table

Now, we need to find the values of the Circuit Excitation Table for each gates. For simplication the current algorithm works on **D Flip Flop**. Later on we can generalise this step for the rest of the Flip Flops. The function at this step will generate the minterms for each D flip flops. Here is the function:

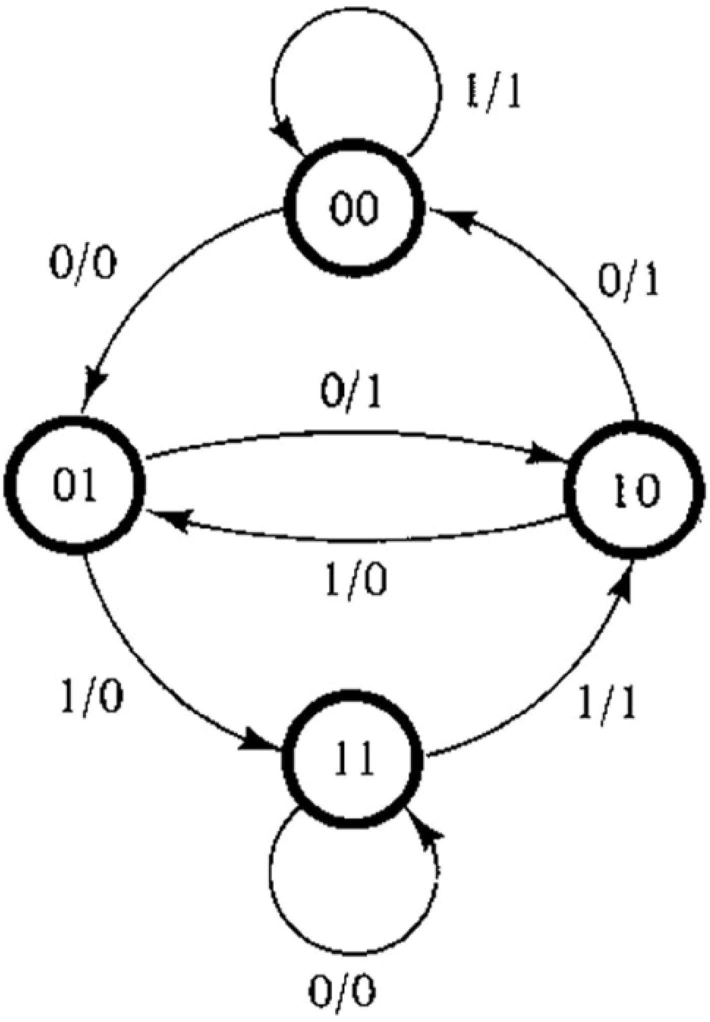
```
function getDecimalValue(output, present_state) {
    return parseInt(output + present_state, 2);
}

function getDFFValues(data) {
    let DFFValues = [];
    for (let gateIndex = 0; gateIndex < data['gates_count']; gateIndex += 1) {
        let gateValues = [];
        data.transition.forEach((trans) => {
            if (trans.next_state.charAt(gateIndex) === '1') {
                console.log(trans);
                console.log(getDecimalValue(trans.output, trans.present_state));
                gateValues.push(
                    getDecimalValue(trans.output, trans.present_state)
                );
            }
        });
        DFFValues.push(gateValues);
    }

    console.log(DFFValues);
}
```

The `getDecimalValue()` gets the decimal value equivalent of the corresponding output and present state. So, if the output for a particular state transition is `1` and the present state was `01` the function will resturn `dec(101)` i.e 5.

The `getDFFValues()` will return a 2-D array for all the values for each Flip Flop.

Results!!

The input state diagram was:



The output of the D flip fliops were:

```
[
    [1, 5, 3, 7],
    [0, 1, 3, 2]
]
```

Which gives the same results when we calculated manually: `D0 = {1, 3, 5, 7}` and `D1 = {0, 1, 2, 3}`

> The Github Repo with the source code is Mini-project-2020/Algorithms