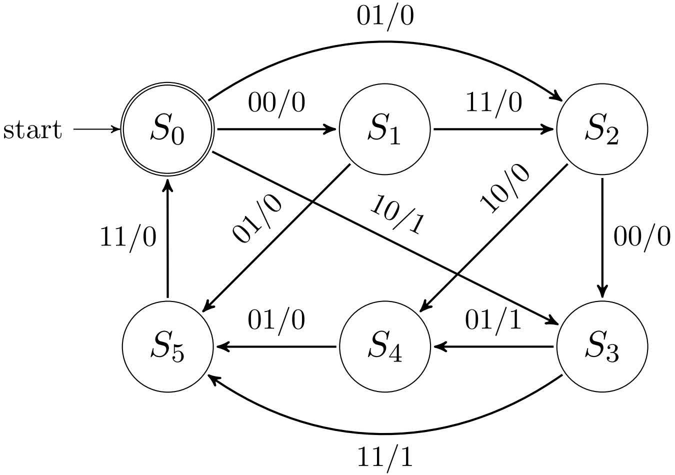
Watermarking-based Intellectual Property Core Protection Scheme

Li-Wei Chen, Shan-Yuan Zheng, Guan-Yu Chen
Supervised by C.M. Li

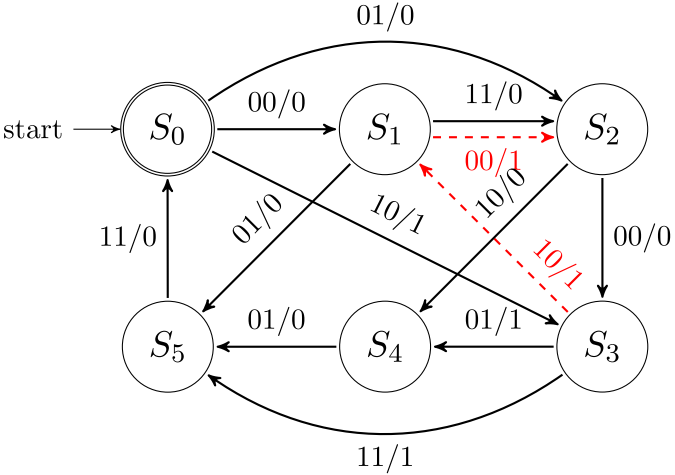
National Taiwan University

June 05, 2018

Project Description



Project Description



Algorithm

Given the input/output bit string $b_1 b_2 \cdots b_n$, where b_i is the i^{th} input/output pair and a the FSM $(\Sigma, S, s_0, \delta, F)$, we first compute the maximum length one may go from each $s \in S$ with the input and output relation specified by $b_i b_{i+1} \cdots b_n$.

The unspecified transition will not be taken into consideration, and the maximum length is recorded if there is no path to satisfy the input bit string. We also add a constraint, if the one candidate stops at b_j, the terminate state for the it must have a unspecified transition for b_{j+1} to be the maximum length path, expect for when the last input is b_n.

Algorithm

If their are multiple states that holds the same length, we choose one randomly.

Then we use a greedy strategy, starting from b_0, we first choose the maximum length state as the first state, then if the maximum path stops at b_i, we then choose the maximum length state for b_{i+2} as the second state. Using b_j as the augmented trasition from the first state to the second state.

Algorithm

A new state is inserted if all the states has zero maximum length and the required input is already specified for all states. Suppose the next input/output pair is b_i, then we use b_i as transition to the new state, and b_{i+1} as the new input/output pair(transition) and continue the algorithm.

Once a new transition or state is add to the graph, they have no difference from the predefined ones. That is, the algorithm will also take them into consideration.

Algorithm

If the algorithm leads to a dead end, which is, all transitions of the give input b_j^i is occupied, we push a new state into the graph and restart the algorithm.

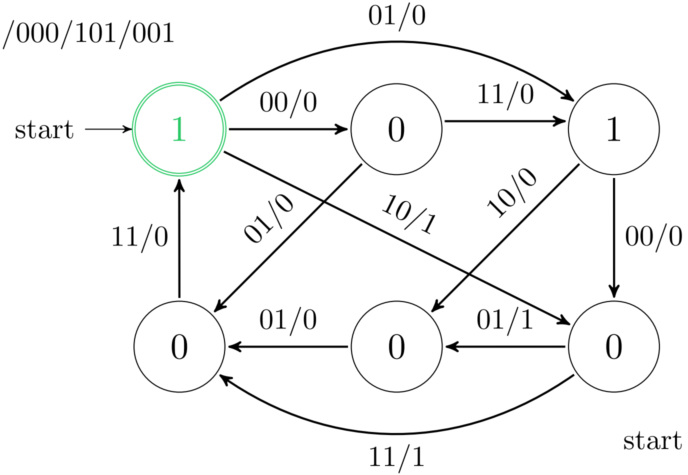
Although this will slow our algorithm a lot for the worst case, the contest input is constraint to be 128 bits, thus it has no influence to the time complexity.

CSFSM Detection

The detection of CSFSM is after the data is loaded and the graph is constructed, we run over each state the check if the out transitions span the whole possible inputs. If yes, the program will report that a CSFSM is detected.

Situation—New Transition

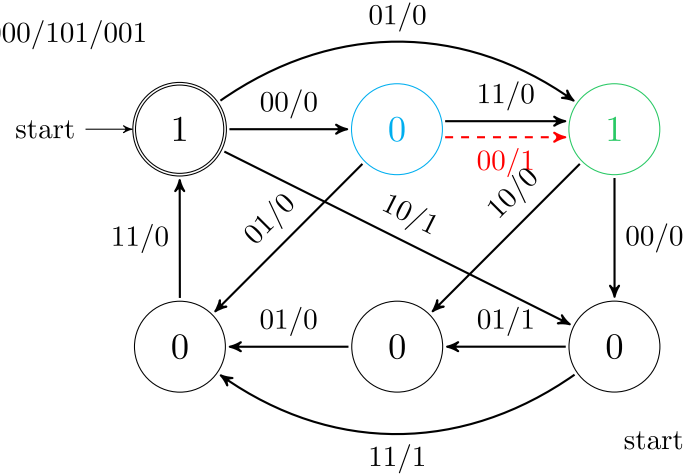
000/001/000/101/001



start at b_0 choose S_0

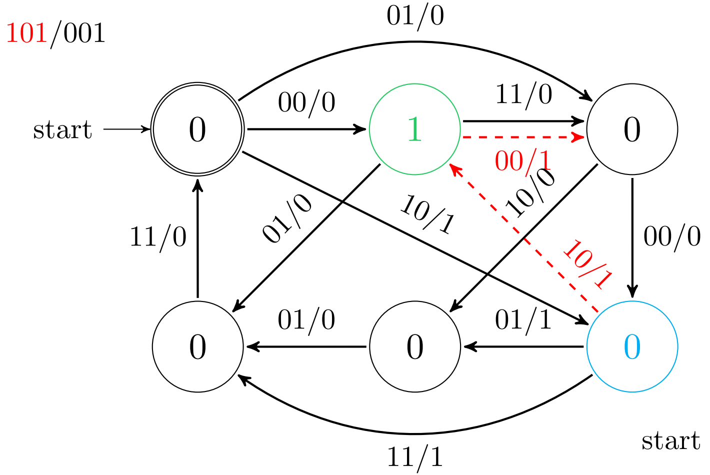
Situation—New Transition

001/000/101/001



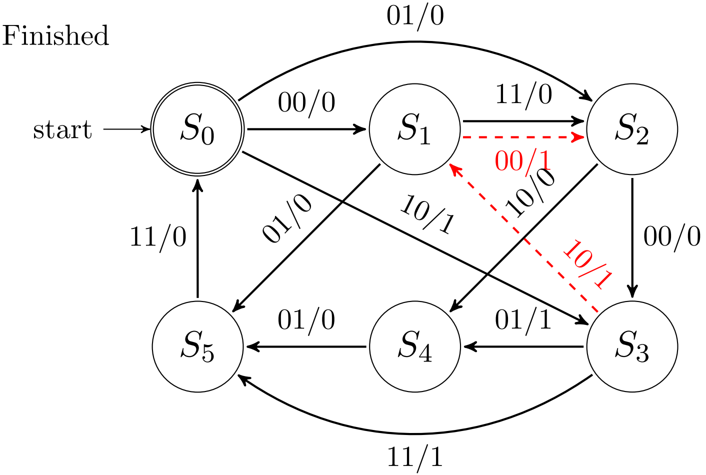
start at b_2 choose S_2

Situation—New Transition



start at b_4 choose S_1

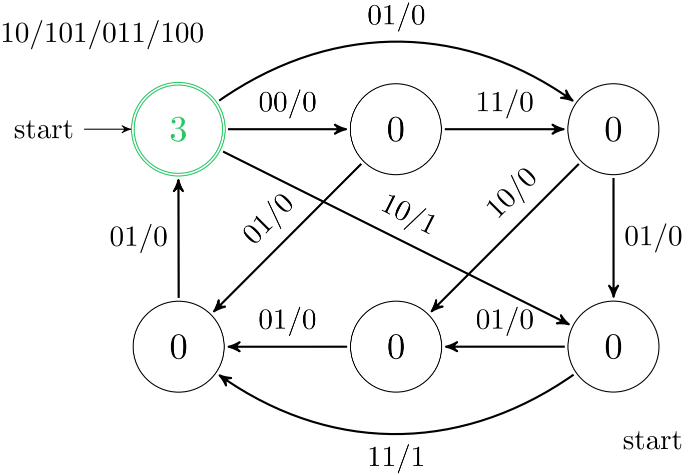
Situation—New Transition



Cost $= 2$

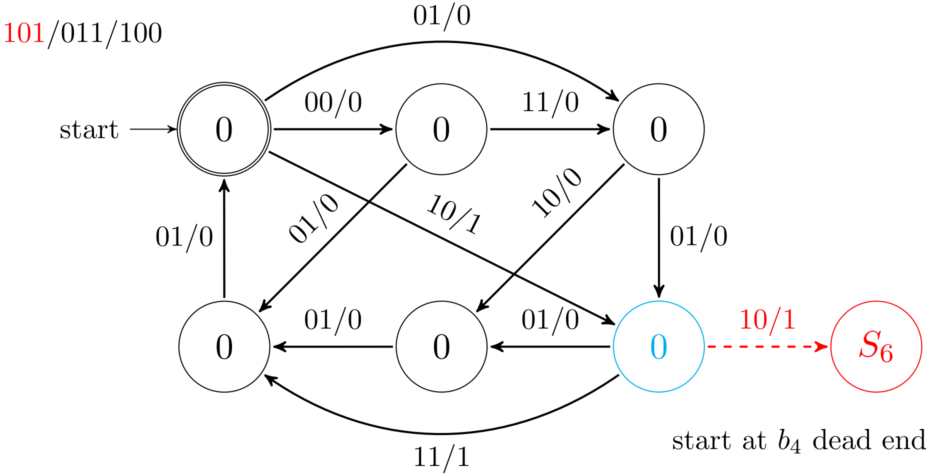
Situation—New State

000/110/010/101/011/100



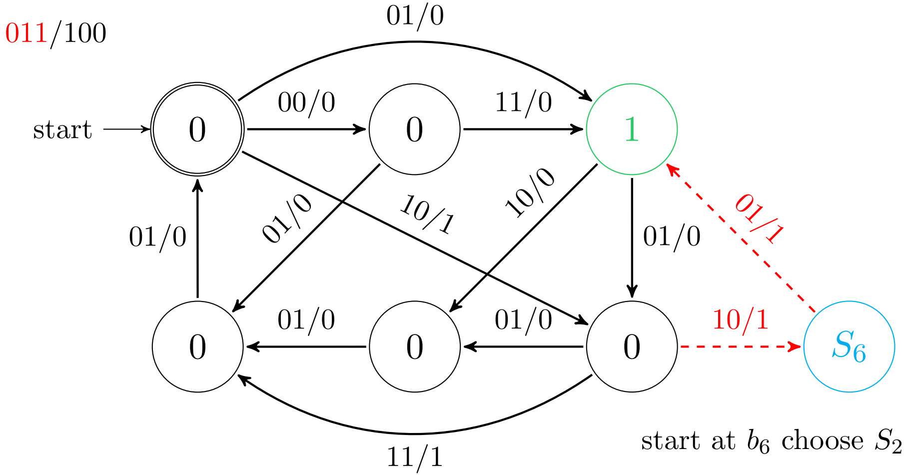
start at b_0 choose S_0

Situation—New State



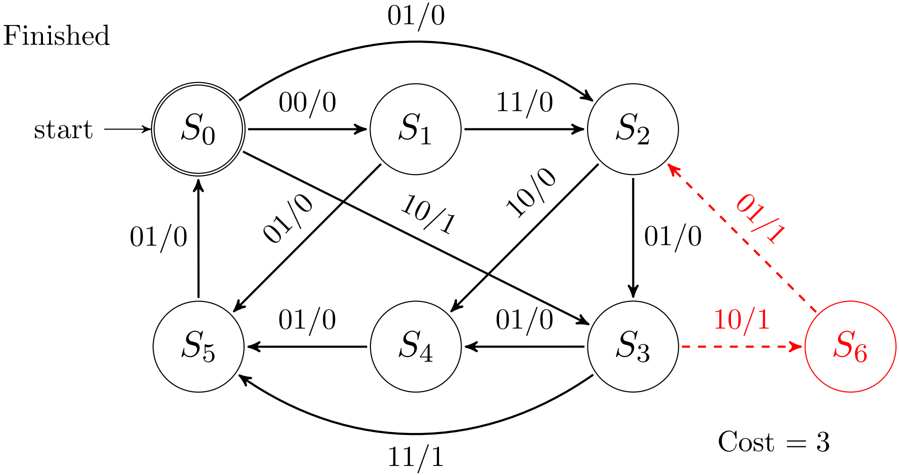
start at b_4 dead end

Situation—New State



start at b_6 choose S_2

Situation—New State



Cost $= 3$

Pseudocode

Algorithm 1 Watermark Part 1

G: Input FSM, $(\Sigma, S, s_0, \delta, F)$
b^i: Input bit string
b^o: Output bit string
$j = 0$, $maxlen = 0$, $start = null$, $dest = null$
while $j \neq b^i.length - 1$ **do**
    for each `s in G.S` **do**
        $m, d, j =$ FindMaxLength(s, b^i, b^o, j)
        if $m \geq maxlen$ and $d \neq null$ **then**
            $maxlen = m$
            $dest = d$
        end if
    end for

Algorithm 2 Watermark Part 2

    if $dest \neq null$ **then**
        AddTransition($start$, $dest$, b^i_j, b^o_j)
    else
        $dest =$ SearchFreeTransition(G, b^i_j)
        if $dest \neq null$ **then**
            AddNewState($dest$, b^i_j, G)
        else
            RandomAddState(G)
            `break`
        end if
    end if
    $j = j + 1$
end while
if $j \neq b^i.length - 1$ **then**
    Watermark(G, b^i, b^o)
end if
return G

Pseudocode

Algorithm 3 SearchFreeTransition

G: Input FSM
b^i_j: Input bit string
$dest = null$
for each `s in G.S` **do**
    if $b^i_j \notin s.outedges$ **then**
        $dest = s$
        `break`
    end if
end for
return $dest$

Algorithm 4 FindMaxLength

s: Input start state
b^i: Input bit string
b^o: Output bit string
j: Current position of the bit string
$tmp = j, d = null$
while $s.next(b^i_j) \neq null$ and $j \neq b^i.length - 1$ **do**
    $s', o = s.next(b^i_j)$
    if $o = b^o_j$ **then**
        $s = s'$
        $d = s'$
        $j = j + 1$
    end if
end while
if $b^i_{j+1} \notin s'.freetransitions$ **then**
    return 0, $null$, tmp
end if
return $j - tmp$, d, tmp

Progress and Difficulties

We are still building the parser, dealing with the kiss format and loading into the C++ class to construct the graph. Some teamates start to implement the algorithm. The difficuties are that it's hard to work parallelly, one working on the algorithm needs to tell what data structure hw needs to the one works with the parser, and the situation changes dynamically.

We think we need a universal coding structure first, we should define all the functions and data structures we need to implement the algorithm, including the return and input types. Then, filling up the whole will be much more efficient for multiple workers.