EE2703 : Logical Simulation

Annangi Shashank Babu

March 2, 2023

First we read the netlist file, and store the given data in a *Directed Acyclic Graph*

[ ]:
```
import networkx as nx
from collections import deque

g = nx.DiGraph()

#taking input from the user
inp = input("Please enter the netlist file: ")
inp2 = input("Please enter the primary input file: ")

#error flag to keep track of any error breaches
eflag = 0

#initilizing a queue to solve in event driven manner
q = deque()

#read from the primary input file and storing it in a list
with open (inp2) as f:
    a = []
    for line in f :
        a.append(line.split())

#reading from the netlist and building the graph
with open(inp) as f:
    for line in f:
        #for 2 input gates (eg and gate)
        if line.split()[1][-1] == '2' :
            (input1,input2,output) = line.split()[2:]
            g.add_edge(input1,output)
            g.add_edge(input2,output)
            g.nodes[output]['gate_type'] = line.split()[1]
            g.nodes[output]['gate_id'] = line.split()[0]
            g.nodes[output]['logic'] = 0
            if( 'gate_type' not in g.nodes[input1].keys()) :
                g.nodes[input1]['gate_type'] = 'PI'
            if( 'gate_type' not in g.nodes[input2].keys()) :
                g.nodes[input2]['gate_type'] = 'PI'
```

```
        #for single input gate (eg not gate)
        else :
            (input1,output) = line.split()[2:]
            g.add_edge(input1,output)
            g.nodes[output]['gate_type'] = line.split()[1]
            g.nodes[output]['gate_id'] = line.split()[0]
            g.nodes[output]['logic'] = 0
            if( 'gate_type' not in g.nodes[input1].keys()) :
                g.nodes[input1]['gate_type'] = 'PI'

#topologically sorting the network
try :
    o = list(nx.topological_sort(g))
except :
    print('error : The netlist is sequencial')
    eflag = 1
sort_node = sorted(o)
gtem3 = g
gtem4 = g
```

[53]:
```
#this function returns the output logic for 2 input gates using logic1 and␣
 ↪logic2 which are the boolean logic
#of the input nodes and the operator is the gate type
def calculate(logic1, logic2, operator) :
    ans = 0
    if (operator == 'and2') :
        ans = int(logic1 and logic2)
        return ans
    if (operator == 'nand2') :
        ans = int(not(logic1 and logic2))
        return ans
    if (operator == 'or2'):
        ans = int(logic1 or logic2)
        return ans
    if (operator == 'nor2'):
        ans = int(not(logic1 or logic2))
        return ans
    if (operator == 'xor2'):
        ans = int((((not logic1) and (logic2)) or ((not logic2) and (logic1))))
        return ans
    if (operator == 'xnor2'):
        ans = int((((not logic1) and (not logic2)) or ((logic2) and (logic1))))
        return ans
    eflag = 1
    print('error gate type not identified')
    return ans
```

1 Topologically ordered evaluation

We topologically sort the graph and iterate through it to find the boolean values of the nodes. The boolean value of a node in a Topologiacally sorted array will only depends on the boolean value of the nodes that preceed it.

[54]:
```
def topo(g,a,o,sort_node) :

    # g : the input netlist stored in form of a graph
    # a : List containing primary input file data
    # o : list of nodes that are topologicaly sorted from graph


    for i in range (1,len(a)) :
        # now we have to soulve for one input vector inside this for loop

        #first read the input vector
        for j in range (0,len(a[0])) :
            if g.nodes[a[0][j]]['gate_type'] != 'PI' :
                eflag = 1
                print('error : primary input not matching with netlist')
            g.nodes[a[0][j]]['logic'] = int(a[i][j])

        #Now traverse the topologically sorted list and calculate the outoput
        for j in range(0,len(o)):

            #if o[j] and primary input the continue to next iteration
            if g.nodes[o[j]]['gate_type'] == 'PI':
                continue

            else :
                # 2 input gates
                if(g.nodes[o[j]]['gate_type'][-1] == '2' ) :
                    (inp1,inp2) = list(g.predecessors(o[j]))
                    g.nodes[o[j]]['logic'] = calculate(g.nodes[inp1]['logic'],␣
↪g.nodes[inp2]['logic'], g.nodes[o[j]]['gate_type'] )
                # single input gates
                else :
                    inp1 = list(g.predecessors(o[j]))
                    inp1 = inp1[0]
                    if(g.nodes[output]['gate_type'] == 'inv') :
                        g.nodes[o[j]]['logic'] = int(not (g.
↪nodes[inp1]['logic']))
                    #else :
                        #g.nodes[o[j]]['logic'] = int((g.nodes[inp1]['logic']))
        with open ("topology.output",'a') as f :
            for node in sort_node :
```

```
                f.write(f"{ (g.nodes[node]['logic']) : <10}")
            f.write(f"\n")
    return g
```

2 Event-driven evaluation

Instead of using a topological sort, we use queues here for updating the states in an event driven approach. whenever a boolean value of a node gets updated, we enqueue all the successors of the node in the queue, because the boolean value of these nodes might change and has to be evaluated. When the queue becomes empty, the evaluation process stops and all node values are updated.

[55]:
```
def append_successors (g,temp,q) :
    for ele in list(g.successors(temp)):
                    if ele not in q :
                        q.append(ele)
    pass
def que(g,a,sort_node) :
    for i in range (1,len(a)) :
        #now we have to solve the one input vector inside this for loop
        for j in range (0,len(a[0])) :
            if g.nodes[a[0][j]]['gate_type'] != 'PI' :
                eflag = 1
                print('error : primary input not matching with netlist')
            # if the logic is changing then in enqueue the Primary node
            if(g.nodes[a[0][j]]['logic'] != int(a[i][j]) ) :
                g.nodes[a[0][j]]['logic'] = int(a[i][j])
                q.append(a[0][j])
        while(len(q) != 0) :
            temp = q.popleft()

            #enqueing all the successors of the primary inputs whose bolean␣
 ↪value had changed
            if g.nodes[temp]['gate_type'] == 'PI' :
                append_successors (g,temp,q)
                continue
            else :
                if(g.nodes[temp]['gate_type'][-1] == '2' ) :
                    (inp1,inp2) = list(g.predecessors(temp))
                    templogic = calculate(g.nodes[inp1]['logic'], g.
 ↪nodes[inp2]['logic'], g.nodes[temp]['gate_type'] )
                    if(templogic == g.nodes[temp]['logic'] ) :
                        continue
                    else :
                        g.nodes[temp]['logic'] = templogic
                        append_successors (g,temp,q)
                else :
                    (inp1) = list(g.predecessors(temp))
```

```
                    inp1 = inp1[0]
                    if(g.nodes[temp]['gate_type'] == 'inv') :
                        templogic == int(not (g.nodes[inp1]['logic']))
                    else :
                        templogic == int((g.nodes[inp1]['logic']))
                    if(templogic == g.nodes[temp]['logic'] ) :
                        continue
                    else :
                        g.nodes[temp]['logic'] = templogic
                        append_successors (g,temp,q)

        with open ("queue.output",'a') as f :
            for node in sort_node :
                f.write(f"{(g.nodes[node]['logic']):<10}")
            f.write(f"\n")
    return g
```

[56]:
```
if (eflag != 1) :
    with open ("topology.output",'w') as f :
        for node in sort_node :
            f.write(f"{node : <10}")
        f.write(f"\n")
    g_temp = g
    g = topo(g,a,o,sort_node)
#print(sort_node)
#print(sorted(g.nodes()))
#print(eflag)
#print(g.nodes()(data = True))
#for node in sort_node :
#    print(node)
#    print(f"{ int(g.nodes[str(node)]['logic']) : <10}")
```

[57]:
```
if (eflag != 1) :
    with open ("queue.output",'w') as f :
        for node in sort_node :
            f.write(f"{node : <10}")
        f.write(f"\n")
    g = que(g_temp,a,sort_node)
```

3 Comparison

When a single input vector is given for a netlist, topological evaluation is faster than the event-driven evaluation, because in event-driven evaluation we find the successesors for the element we pop from the queue (but only if the boolean value is changed) and thus becomes a complex code that the first case in single input vector

But as we increse the number of input vectors for which the output needs to be calculated, the

event-driven evaluation becomes more optimized compared to topological evaluation as we are only enquing those boolean value which have changed from the previous value, this optimization is a result of precalculated boolean values for a different input vector. (the more close the 2 input vectors are the lesser the time event driven apporach will take)

[58]:
```
%timeit topo(gtem3,a,o,sort_node)
%timeit que(gtem4,a,sort_node)
```

```
30.9 ms ± 41.6 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
25.7 ms ± 171 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```