week4

Sidharth S Kumar EE21B130

March 2, 2023

1 Getting the input from the netlist file

In this cell the files are inputted and transfered into a list for easy manipulation.
A class is defined to take in the value from each line. The edges list is included. Then this list is passed to the inbuilt edge -graph function from the networks library.
Now the data from the input file is transfered into a list of dictionaries for later access. And another gate type dictionary is created to find the type of the gate from the output terminal name.
Finally the graph is sorted topologically and saved as nl

[13]:
```
# Importing the necessary libraries
import networkx as nx

# Declaring variables for opening the files required = Give the required file␣
 ↪names as the variables mentioned below
netwrok = "c432.net"
inputs = "c432.inputs"

# Defining a class to take in the vlaues from the network file
class Component:
        def __init__(self,name,g_type,input,output):
                self.name = name
                self.g_type = g_type
                self.input = input
                self.output = output

# Declaring a list to store the components
l = []

# Opening the file mentioned and storing it in the list
with open(netwrok) as f:
    lines = f.readlines()
    for line in lines:
        name,g_type,*input= line.split()
        output = line.split()[-1]
        Gate = Component(name,g_type,input,output)
        l.append(Gate)

# Declaring and creating the  edges list for the topoligal sorting
```

```
edges = []
g = nx.DiGraph()
for gate in l:
    if len(gate.input) == 3:
        edges.append((gate.input[0],gate.output))
        edges.append((gate.input[1],gate.output))
    else:
        edges.append((gate.input[0],gate.output))

# Making the diGraph for the topological sorting
g.add_edges_from(edges)

# Taking inputs from the inputs file and storing them in the values list as␣
 ↪dictionaries
value = []
with open(inputs) as file:
    lineIn = file.readlines()
    pi = lineIn[0].split()
    for i in range(len(lineIn)-1):
        value.append({})
        for j in range(len(pi)):
            value[i].update({lineIn[0].split()[j]:lineIn[i+1].split()[j]})
value1 = value.copy()

# Saving the gate_types of eadh gate as a dictionary for later use in the gate␣
 ↪function
gate_type = {}
for node in pi:
     gate_type.update({node:"PI"})
for node in l:
     gate_type.update({node.output:node.g_type})
nx.set_node_attributes(g,gate_type,name = "gateType")

# Making a topologically sorted list of the nodes
nl = list(nx.topological_sort(g))
```

2 Solving using the topological order

Topological solving works by finding the value of each node from the predeseesors in topological order. The sorted list of nodes are iterated over to find their type and predessors, and then to find the output based on these parameters . All these steps are iterated over the number of input sets given

[18]:
```
# Iterating through all the input sets given
for i in range(len(lineIn)-1):

   # Iterating through the gates except pi
```

```
   for j in range(len(pi),len(nl)):

      # Finding the predecessors of the gate
      p = list(g.predecessors(nl[j]))

      # Checking for all types of gates and performing the necessary operations
      if gate_type[nl[j]] == "nand2":
         out = int(not(value[i][p[0]] and value[i][p[1]]))


      elif gate_type[nl[j]] == "and2":
         out = int(value[i][p[0]] and value[i][p[1]])

      elif gate_type[nl[j]] == "nor2":
         out = int(not(value[i][p[0]] or value[i][p[1]]))

      elif gate_type[nl[j]] == "or2":
         out = int(value[i][p[0]] or value[i][p[1]])

      elif gate_type[nl[j]] == "inv":
         out = int(not(value[i][p[0]]))

      elif gate_type[nl[j]] == "xnor2":
         out = int(not(value[i][p[0]] ^ value[i][p[1]]))

      elif gate_type[nl[j]] == "xor2":
         out = int(value[i][p[0]] ^ value[i][p[1]])

      # Updating the output of the current gate into the value list
      value[i].update({nl[j]:out})
print(value[0])
```

```
{'N1': '0', 'N4': '0', 'N8': '1', 'N11': '0', 'N14': '1', 'N17': '1', 'N21':
'0', 'N24': '0', 'N27': '0', 'N30': '0', 'N34': '1', 'N37': '1', 'N40': '1',
'N43': '0', 'N47': '0', 'N50': '0', 'N53': '0', 'N56': '1', 'N60': '1', 'N63':
'0', 'N66': '1', 'N69': '0', 'N73': '1', 'N76': '1', 'N79': '1', 'N82': '0',
'N86': '1', 'N89': '1', 'N92': '1', 'N95': '1', 'N99': '0', 'N102': '0', 'N105':
'0', 'N108': '1', 'N112': '0', 'N115': '0', 'n_2': 0, 'n_4': 0, 'n_8': 0, 'n_3':
0, 'n_1': 0, 'n_9': 0, 'n_5': 0, 'n_0': 0, 'n_6': 0, 'n_7': 0, 'n_20': 0,
'n_12': 0, 'n_10': 0, 'n_13': 0, 'n_14': 0, 'dummy_37': 1, 'n_15': 0, 'n_21': 0,
'n_19': 0, 'dummy_56': 1, 'n_11': 1, 'dummy_58': 0, 'dummy_57': 0, 'dummy_54':
0, 'n_17': 0, 'n_16': 1, 'dummy_55': 0, 'n_18': 0, 'n_22': 1, 'n_23': 0, 'n_24':
1, 'n_37': 0, 'n_39': 0, 'n_34': 0, 'n_27': 0, 'n_31': 0, 'n_43': 0, 'n_26': 0,
'n_29': 0, 'N223': 0, 'n_42': 0, 'n_38': 0, 'n_41': 0, 'n_40': 0, 'n_36': 0,
'n_35': 0, 'n_33': 0, 'n_28': 0, 'dummy_35': 1, 'n_32': 0, 'n_44': 0,
'dummy_36': 1, 'n_46': 0, 'n_30': 0, 'dummy_16': 0, 'dummy_18': 0, 'dummy_20':
0, 'dummy_22': 0, 'dummy_24': 0, 'dummy_26': 0, 'dummy_40': 0, 'n_25': 1,
'n_59': 1, 'n_58': 1, 'n_57': 1, 'n_56': 1, 'n_48': 1, 'n_52': 1, 'n_54': 1,
```

```
'n_53': 1, 'n_63': 1, 'n_50': 1, 'dummy_48': 0, 'n_47': 1, 'n_62': 1, 'n_45': 0,
'dummy_49': 1, 'dummy_43': 1, 'dummy_50': 1, 'n_49': 0, 'dummy_52': 1,
'dummy_44': 1, 'dummy_51': 1, 'dummy_47': 1, 'dummy_53': 0, 'n_88': 1, 'n_60':
0, 'n_55': 0, 'n_51': 1, 'n_61': 0, 'dummy_46': 0, 'n_64': 1, 'n_65': 0,
'dummy_45': 0, 'N329': 1, 'dummy_39': 1, 'dummy_41': 1, 'dummy_42': 0, 'n_77':
1, 'n_76': 1, 'n_75': 1, 'n_74': 0, 'n_73': 0, 'n_72': 0, 'n_71': 0, 'n_70': 0,
'n_69': 0, 'n_68': 0, 'n_67': 0, 'n_66': 0, 'n_81': 0, 'n_79': 0, 'n_78': 0,
'n_85': 0, 'n_86': 0, 'dummy_38': 1, 'n_82': 1, 'dummy_30': 0, 'n_83': 1,
'dummy_34': 0, 'n_89': 0, 'dummy_32': 0, 'dummy_33': 0, 'n_84': 0, 'n_87': 1,
'n_90': 1, 'dummy_31': 0, 'dummy_29': 1, 'n_91': 1, 'dummy_28': 1, 'N370': 0,
'dummy_15': 0, 'dummy_17': 0, 'dummy_19': 0, 'dummy_21': 0, 'dummy_23': 0,
'dummy_25': 0, 'dummy_27': 0, 'n_92': 1, 'n_101': 1, 'n_98': 1, 'n_97': 1,
'n_96': 1, 'n_95': 1, 'n_94': 1, 'n_93': 0, 'dummy_14': 0, 'dummy_12': 0,
'dummy_8': 0, 'dummy_9': 0, 'dummy_11': 0, 'dummy_13': 0, 'dummy_10': 0,
'dummy_0': 0, 'n_104': 1, 'n_103': 1, 'n_106': 1, 'n_117': 1, 'n_102': 1,
'n_108': 1, 'n_113': 1, 'n_109': 0, 'dummy_2': 1, 'n_105': 0, 'dummy_4': 1,
'dummy_7': 1, 'n_107': 0, 'n_111': 1, 'n_110': 0, 'n_118': 1, 'dummy_6': 1,
'n_112': 1, 'n_114': 1, 'N430': 0, 'dummy_3': 1, 'dummy_5': 1, 'n_116': 1,
'N431': 0, 'N432': 0, 'dummy_1': 1, 'n_121': 0, 'n_122': 0, 'N421': 1}
```

3 Using the Event - driven method

Event-driven method works by updating the states in an event driven approach. This utilises a queue that contuially gets update by popping elements and adding its sucessors, given that both its predesessors are present in the value list already. The popped value if not already in the value list , is then added by considering the type of gate and the predessors. All these steps are iterated over the number of input sets given

[20]:
```
# Importing the
from queue import Queue

# Declaring and initializing the queue
Event = Queue()
for item in nl[:len(pi)]:
    Event.put(item)

# Iterating through all the input sets given
for i in range(len(lineIn)-1):


    while not Event.empty():
        e = Event.get()
        # Adding the succesors of the thing to the queue only is all the␣
 ↪predessors are there in the value list already
        for thing in list(g.successors(e)):
            if all(elem in value1[i] for elem in list(g.predecessors(thing))):
                Event.put(thing)
```

```
        # Checking if the value of the key already exist
        if e not in value1[i]:
            print(e)
            p = list(g.predecessors(e))
            out = 0

            # Checking and executing all the gates
            if gate_type[e] == "nand2":
                out = int(not(value[i][p[0]] and value[i][p[1]]))

            elif gate_type[e] == "and2":
                out = int(value[i][p[0]] and value[i][p[1]])

            elif gate_type[e] == "nor2":
                out = int(not(value[i][p[0]] or value[i][p[1]]))

            elif gate_type[e] == "or2":
                out = int(value[i][p[0]] or value[i][p[1]])

            elif gate_type[e] == "inv":
                out = int(not(value[i][p[0]]))

            elif gate_type[e] == "xnor2":
                out = int(not(value[i][p[0]] ^ value[i][p[1]]))

            elif gate_type[e] == "xor2":
                out = int(value[i][p[0]] ^ value[i][p[1]])

            # Adding the key value pair to the value list
            value1[i].update({e:out})
print(value1[0])
```

```
{'N1': '0', 'N4': '0', 'N8': '1', 'N11': '0', 'N14': '1', 'N17': '1', 'N21':
'0', 'N24': '0', 'N27': '0', 'N30': '0', 'N34': '1', 'N37': '1', 'N40': '1',
'N43': '0', 'N47': '0', 'N50': '0', 'N53': '0', 'N56': '1', 'N60': '1', 'N63':
'0', 'N66': '1', 'N69': '0', 'N73': '1', 'N76': '1', 'N79': '1', 'N82': '0',
'N86': '1', 'N89': '1', 'N92': '1', 'N95': '1', 'N99': '0', 'N102': '0', 'N105':
'0', 'N108': '1', 'N112': '0', 'N115': '0', 'n_2': 0, 'n_4': 0, 'n_8': 0, 'n_3':
0, 'n_1': 0, 'n_9': 0, 'n_5': 0, 'n_0': 0, 'n_6': 0, 'n_7': 0, 'n_20': 0,
'n_12': 0, 'n_10': 0, 'n_13': 0, 'n_14': 0, 'dummy_37': 1, 'n_15': 0, 'n_21': 0,
'n_19': 0, 'dummy_56': 1, 'n_11': 1, 'dummy_58': 0, 'dummy_57': 0, 'dummy_54':
0, 'n_17': 0, 'n_16': 1, 'dummy_55': 0, 'n_18': 0, 'n_22': 1, 'n_23': 0, 'n_24':
1, 'n_37': 0, 'n_39': 0, 'n_34': 0, 'n_27': 0, 'n_31': 0, 'n_43': 0, 'n_26': 0,
'n_29': 0, 'N223': 0, 'n_42': 0, 'n_38': 0, 'n_41': 0, 'n_40': 0, 'n_36': 0,
'n_35': 0, 'n_33': 0, 'n_28': 0, 'dummy_35': 1, 'n_32': 0, 'n_44': 0,
'dummy_36': 1, 'n_46': 0, 'n_30': 0, 'dummy_16': 0, 'dummy_18': 0, 'dummy_20':
0, 'dummy_22': 0, 'dummy_24': 0, 'dummy_26': 0, 'dummy_40': 0, 'n_25': 1,
```

```
'n_59': 1, 'n_58': 1, 'n_57': 1, 'n_56': 1, 'n_48': 1, 'n_52': 1, 'n_54': 1,
'n_53': 1, 'n_63': 1, 'n_50': 1, 'dummy_48': 0, 'n_47': 1, 'n_62': 1, 'n_45': 0,
'dummy_49': 1, 'dummy_43': 1, 'dummy_50': 1, 'n_49': 0, 'dummy_52': 1,
'dummy_44': 1, 'dummy_51': 1, 'dummy_47': 1, 'dummy_53': 0, 'n_88': 1, 'n_60':
0, 'n_55': 0, 'n_51': 1, 'n_61': 0, 'dummy_46': 0, 'n_64': 1, 'n_65': 0,
'dummy_45': 0, 'N329': 1, 'dummy_39': 1, 'dummy_41': 1, 'dummy_42': 0, 'n_77':
1, 'n_76': 1, 'n_75': 1, 'n_74': 0, 'n_73': 0, 'n_72': 0, 'n_71': 0, 'n_70': 0,
'n_69': 0, 'n_68': 0, 'n_67': 0, 'n_66': 0, 'n_81': 0, 'n_79': 0, 'n_78': 0,
'n_85': 0, 'n_86': 0, 'dummy_38': 1, 'n_82': 1, 'dummy_30': 0, 'n_83': 1,
'dummy_34': 0, 'n_89': 0, 'dummy_32': 0, 'dummy_33': 0, 'n_84': 0, 'n_87': 1,
'n_90': 1, 'dummy_31': 0, 'dummy_29': 1, 'n_91': 1, 'dummy_28': 1, 'N370': 0,
'dummy_15': 0, 'dummy_17': 0, 'dummy_19': 0, 'dummy_21': 0, 'dummy_23': 0,
'dummy_25': 0, 'dummy_27': 0, 'n_92': 1, 'n_101': 1, 'n_98': 1, 'n_97': 1,
'n_96': 1, 'n_95': 1, 'n_94': 1, 'n_93': 0, 'dummy_14': 0, 'dummy_12': 0,
'dummy_8': 0, 'dummy_9': 0, 'dummy_11': 0, 'dummy_13': 0, 'dummy_10': 0,
'dummy_0': 0, 'n_104': 1, 'n_103': 1, 'n_106': 1, 'n_117': 1, 'n_102': 1,
'n_108': 1, 'n_113': 1, 'n_109': 0, 'dummy_2': 1, 'n_105': 0, 'dummy_4': 1,
'dummy_7': 1, 'n_107': 0, 'n_111': 1, 'n_110': 0, 'n_118': 1, 'dummy_6': 1,
'n_112': 1, 'n_114': 1, 'N430': 0, 'dummy_3': 1, 'dummy_5': 1, 'n_116': 1,
'N431': 0, 'N432': 0, 'dummy_1': 1, 'n_121': 0, 'n_122': 0, 'N421': 1}
```

[16]:
```
for i in range(len(lineIn)-1):
    if value[i] != value1[i]:
        print("Some difference is there")
```