COP290 : Assignment 2 - AC Circuit Solver
Design Document

Sahil Bansal (2016CSJ0008) & Sahil Singh(2016CSJ0025)

March 31, 2018

1 Overall Design :

An object-oriented design is followed and the code is properly structured into several cpp and header files. The sub-components are described in detail below. Also, all the algorithms used are briefly touched upon.

2 Sub-Components :

- **ac_circuit_solver.cpp** is the main file for the entire project and contains the code for solving the circuit, and also calls the **parse** and **draw** functions.
- **draw.h** and **draw.cpp**: They contain the main function to draw the entire circuit.
- **parse.h** and **parse.cpp**: They contain the element and source structures and their vectors obtained after parsing the input netlist.
- **parseCircuit.y** is the yacc file used while parsing.
- **scanCircuit.l** is the lex file used while scanning.
- **svg.h** and **svg.cpp**: They contain the SVG functions used for drawing individual elements and implement onClick events on the text to get an alert box showing statistics of the element when clicked.
- **Eigen** library is used for generating and solving the matrices.

3 Testing Sub-Components :

3.1 Scanner :

- The parser is checked by writing some cases that may violate the syntax or semantic rules. Such case(s) are defined in the **Public_cases/syntacticCases** folder. Also, in the test cases provided, cases numbered 2 and 5 don't follow the rules of the netlist as enforced in the README.

3.2 Parser :

- It is checked by printing the values stored in the **sourceElements** and **circuitElements** vectors declared in the **parse.h** file.
- This print function is now moved to **ac_circuit_solver.cpp** and prints to the terminal. It can be verified that the input netlist is stored properly from the terminal.

3.3 SVG :

- The SVG functions defined were checked manually by calling all of them in a temporary main function defined in **svg.cpp** just for the purpose of testing.

3.4 Draw :

- Some test cases were designed and are kept in the **Public_cases/solveCases** folder.
- A script **testScript.sh** was written to simultaneously run all these test cases.
- The generated SVG outputs were manually then checked for any errors and the errors were corrected as and when they came.

3.5 Solve :

- The solving part was checked by separately running the file **ac_circuit_solver.cpp**.
- For the two solved test cases provided, the matrices used in **MNA algorithm** were all printed to the screen and calculated manually and matched.
- The output currents and voltages along with phases were also then matched with the outputs provided.
- Further testing was done on all the cases in the **Public_cases/solveCases** folder to make sure that the algorithm terminated.

4 Algorithm used for drawing the Circuit :

- The components are seperated into different vectors each containing those components whose difference between starting and ending nets is the same.
- It is made sure that for each component the starting net value is smaller than the ending net value. Also, this does not effect the polarity of current or voltage sources as this part is separate from solving. Moreover, **drawMain function**

is called after solving the circuit to make sure that elements can be made interactable easily.

- Each of these vector is then sorted based on the ending net value.
- A loop is then run, starting from those elements with the highest net difference value.
- All the elements with the same net difference values are drawn in a particular iteration of the loop.
- It contains another nested loop which is for a greedy technique applied to draw the elements such that the next elements' starting net will be higher or equal to the ending net of previous component. This makes sure that maximum no of disjoint components are drawn in a single line.
- If there are some components which are connected in parallel, this case has also been handled by drawing all such elements first and then moving to the next iteration in another nested loop.

5 Modified Nodal Analysis for solving the Circuit :

- In brief, this method, after applying **Kirchoff's current law** to each node by using some conventions and then writing equations for voltages of each voltage source, reduces the circuit into the form:

$$AX = Z$$

where A, X, and Z are all complex matrices. Further,

$$A = \begin{pmatrix} G & B \\ C & D \end{pmatrix}$$

where G contains the admittances of the passive elements & B, C and D depend on the active sources and contain only either 1, 0 or -1.

$$Z = \begin{pmatrix} I \\ E \end{pmatrix}$$

where I contains the net current at each node and E contains the known voltage sources. The matrix **X** obtained after solving the equation is:

$$X = \begin{pmatrix} v \\ j \end{pmatrix}$$

where **v** obtained contains the voltages at each node and **j** contains the current supplied by each voltage source.

- These matrices are suitably constructed and then after solving:
  - The voltage across an element is simply the difference of voltages across its starting and ending net.
  - The current, thus across it is simply the product of its **admittance** and the voltage thus obtained.
- For the case when there are **multiple sources of different frequencies**:
  - First, they are sorted in non-decreasing order of their frequencies.
  - Then, only those which have the same frequency are considered active at a time.
  - The matrices are suitable reconstructed or adjusted to handle such case.

6 Interactable SVG :

The text used for a particular element is provided with a **onClick** attribute containing the function which takes all the values of the element to be shown as paramters. This function is included in the **script** tag nested inside the SVG tag and handled by the **addInteraction** function defined in the **svg.cpp** file. The arguments are directly passed through the **drawText** function called while drawing the text in the **draw.cpp** file.