Parser Documentation

Adam Rehman, Brandon Cann, Xin Wang [1]

May 17, 2020

[1]document compiled by Xin Wang

Contents

1 Introduction

Based on the provided project briefing, a netlist describing the circuit will be provided in a file using a reduced SPICE format. It is required by the team to formulate an approach to reading the input format and store the information in a comprehensible and efficient data structure.

This document will research the format of the SPICE netlist and explore the different approaches in storing the information a certain way.

This document is primarily used for planning the most effective way to approach the problem as well as allowing fellow teammates to engage productively on a equal knowledge footing.

2 Input (.cir) file

Source file for any version of SPICE has the following format [1].

```
title
elements
.model statements
analysis commands
output commands
.end
```

```
* C:\Users\alw\Desktop\spice\Draft2.asc
V1 N001 N004 SINE(0 3 1k)
R1 N002 N001 30
R2 N003 N002 10
R3 N006 N003 4k5
R4 N004 N005 590
R5 N002 N005 50
R6 N006 N005 4k5
.backanno
.end
```

Figure 1: Netlist input example

Important points to note:

- First line `title` is used as a title on output files.
  - Parser ignores the first line of the netlist.
- File must end with command `.end`.
  - Parser exits when `.end` is encountered.
- Comment begins with '*' which covers entire line.
  - Parser ignores entire line if a '*' char is at the beginning.

3 Netlist Element Format

A **netlist** will contain a set of statements defining elements in a circuit.

Connections are described by naming nodes. The program will automatically assign a number to the nodes in the circuit starting from 1.

Node 0 is defined as Ground. It is necessary to have a Node 0 since it is the reference point for all voltages specified.

Initial stage of the parser will only be able to support input with nodes starting from N1 and ground defined as N0. SPICE defined ground node as 0. Suport for complex values such as 4k5 will be added at a later stage too. This will be revised when basic function of Analysis module is implemented.

We have to consider a situation where the input syntax is correct but the circuit represented is unrealizable. To overcome this, we will need a function to check if the circuit described is realistic. This is explored more in the *Analysis* module.

3.1 Basic components

Current supported elements are basic passive elements.

Passive elements are composed of the following:[1]

- Resistors: `R`

  $$\texttt{R} < \texttt{description} >< \texttt{n1} >< \texttt{n2} >< \texttt{value} >$$

- Capacitor: `C`.

  $$\texttt{C} < \texttt{description} >< \texttt{n1} >< \texttt{n2} >< \texttt{value} >< \texttt{initial} - \texttt{condition} >$$

- Inductor: `L`.

  $$\texttt{L} < \texttt{description} >< \texttt{n1} >< \texttt{n2} >< \texttt{value} >< \texttt{initial} - \texttt{condition} >$$

Both C and L will be considered at a later stage. First versions implemented will cover resistors only. This approach will allow the core functionalities of the program to be consolidated and, with the experience gained, will allow us to find the best approach to implement more complex functions.

[1] Initial conditions for C and L are optional and default set to 0

3.2 Independent Sources

The character after the letter must be a unique instance name.

3.2.1 Independent DC sources

- Voltage Sources:

  V < description >< n+ >< n− >< value >

- Current Sources:

  I < description >< n+ >< n− >< value >

3.2.2 Independent *Sine* sources

- Voltage Sources:

  V < description >< n+ >< n− > SINE < Offset >< Amplitude >< Frequency >

- Current Sources:

  I < description >< n+ >< n− > SINE < Offset >< Amplitude >< Frequency >

When users are entering component values, it is important that the software recognises common abbreviations for units.

3.2.3 Diode

3.3 Analysis functions

3.3.1 .tran function

.tran < tstep >< tstop >< tstart < tmax >>

3.4 Support for powers of ten

In order to support `spice` format, the following abbreviations for powers of ten must be recognised and are case sensitive:

- f [femto]: 10^{-15}
- p [pico]: 10^{-12}
- n [nano]: 10^{-9}
- u [micro]: 10^{-6}
- m [milli]: 10^{-3}
- k [kilo]: 10^{+3}
- Meg [mega]: 10^{+6}
- g [giga]: 10^{+9}
- t [tera]: 10^{+12}

Any unrecognised characters are ignored.

4 Network Diagram/Graphs

Network Graphs show the relationships between a set of entries and a circuit is a good example of a graph. Each entry is represented by a *Node* and the connections between nodes are represented through *Edges*.

Through Modified Nodal Analysis, an algorithm can be created to form the expression $Ax = b$.

The circuit inputted will be in vector of nodes which will then be read and inputted into the required matrices.

This will be explored further in the analysis module.

5 Implementation

Block diagram depicting the breakdown of Parse Netlist module:

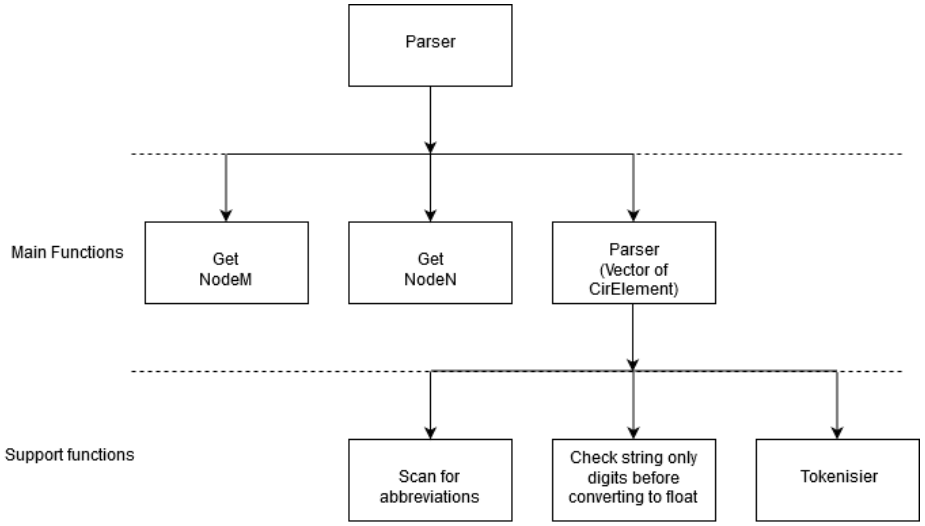

Figure 2: Netlist module breakdown

5.1 CirElement struct

```
struct CirElement
{
    letter: component name
    name: name of node
    node1: node this node is connected to
    node2: node this node is connected to
    value: float
}

struct CirSrc
{
    letter: component name
    name: name of node
    node1: node this node is connected to
    node2: node this node is connected to
    type: string: DEFAULT "DC"
    DC: float
    A: float
    freq: float
}
```

5.2 Parser

```
parser(cin, &vector<CirElement>, &vector<CirSrc>)
{
    Tokenise
    Put in values into respective variables
        if 'v' or 'i' detected:
            Call SrcSort
        if other elements detected:
            Push into CirElement
        Detect values, pass into custom_pow
        Push CirElement into vector
}
```

5.2.1 converter

Basic function: e.g. 5k, 50 etc

```
converter(string: input)
{
    Check if there are keywords e.g. k, m, M, G

    If not present, two scenario:
        Unknown letter present: extract digits
        Convert to float
        Empty string (End of recursion): return 0

    If present:
        Find position where keyword appears
        Take string before keyword and convert
        Multiply/divide the digit by keyword
}
```

5.2.2 tokeniser

```
tokeniser(string: input)
{
    Call regex to tokenise the string
    Push each token into a vector
    Return vector
}
```

5.2.3 isdigit

```
isdigit(string: input)
{
    Iterate over string
    Take each character and into 'isdigit' test
    Return boolean
}
```

5.3 Node N

```
getnodeN(vector<CirElement>: input)
{
    Return size of vector<CirElement>
}
```

5.4 Node M

```
getnodeM(vector<CirSrc>)
{
    Return size of vector<CirSrc>
}
```

6 Version history

6.1 Version 1.0

Basic functionality of the Parser is implemented.

6.1.1 List of functions

- Basic circuit components recognised.

6.1.2 Still to do for v2.0

- Recognise SINE voltage source.
- Recognise grounded nodes appear as 0 only.
- Recognise transient function calls.

7 Add-ons

There are more components to consider such as:

- Voltage controlled dependent sources
- Current controlled dependent sources
- Diodes
- BJT
- MOSFET
- Python implementation
- Efficient way to use recursion and not efficient to use matrix but more accurate.

References

[1] Phyllis R. Nelson. *Introduction to spice source files*. DOI: https://www.cpp.edu/~prnelson/courses/ece220/220-spice-notes.pdf.