SML

(Spice Manipulation Language)
Final Report

Columbia University
Department of Computer Science
Professor Edwards
Programming Languages and Translators

Spencer Greenberg
Michael Apap
Rob Toth
Ron Alleyne

Table of Contents

1. SPICE Manipulation Language (White Paper) ... 4
  1.1 Abstract ... 4
  1.2 Introduction ... 4
  1.3 Background ... 5
  1.4 Language Features ... 6
    1.4.1 Scope ... 6
    1.4.2 Object Types ... 6
    1.4.2 Spice Insertions ... 6
    1.4.3 Lists ... 6
  1.5 Properties of circuit objects ... 6
2. Tutorial ... 7
  2.1 Sample SML Program ... 7
  2.2 Interpreting and Translating ... 8
  2.3 Analyzing Output ... 8
3. Language Reference Manual ... 9
  3.1 Introduction ... 9
  3.2 Lexical Conventions ... 9
    3.2.1 Comments ... 9
    3.2.2 Identifiers ... 9
    3.2.3 Numbers ... 9
    3.2.4 Strings ... 9
    3.2.5 Objects ... 10
    3.2.6 Numerical Objects ... 10
    3.2.7 Tokens ... 10
    3.2.8 Keywords ... 12
  3.3 Data Types ... 13
    3.3.1 Basic Object Types ... 13
    3.3.2 SPICE Object types ... 13
  3.4 Expressions and Statements ... 15
    3.4.1 Primary Expressions ... 15
    3.4.3 Relational Expressions ... 15
  3.5 Statements ... 15
    3.5.1 Assignment ... 15
    3.5.2 Conditional Statement ... 15
    3.5.3 Iterative Statement ... 16
  3.6 Operator precedence ... 16
  3.7 SPICE Insertion ... 17
  3.8 Sample SPICE insertions ... 17
4. Project Plan ... 19

4.1 Phases 19
4.1.1 Planning Phase 19
4.1.2 Specification Phase 19
4.1.3 Development Phase 19
4.1.4 Testing Phase 19
4.2 Programming Style 20
4.3 Project Roles 20
4.4 Project Timeline 20
4.5 Development Environment 21
4.6 Project Log 21
5. Architectural Design 22
5.1 Components of Translator 22
5.2 Specifics 22
5.2.1 Lexer 22
5.2.2 Parser 22
5.2.3 Static and Semantic Analyzer 23
5.2.4 Interpreter 23
5.2.5 Code Generator 23
5.3 Extra Components 23
5.3.1 Error Checking 23
5.3.2 "SPICE" Injection 23
6 Test Plan 24
6.1 Unit Testing 24
6.2 Integrated Testing 24
6.3 Test Tools 24
6.4 Sample Testing 25
6.4.1 Basic Test 25
6.4.2 List Test 27
6.4.3 Resistor Bank 30
6.4.4 Resistor Capacitor Bank 50 33
7 Lessons Learned 39
7.1 Spencer Greenberg 39
7.2 Ron Alleyne 39
7.3 Michael Apap 39
7.4 Robert Toth 40
8 Appendix 41
8.1 Grammar File 41
8.2 Static and Semantic Analyzer 47
8.3 Interpreter 63
8.4 Code Generator 88
8.5 SML Header File 111
8.6 Main Program 129
8.7 Tree Node Class 132

1. SPICE Manipulation Language (White Paper)

1.1 Abstract

SML has been proposed as a means of simplifying SPICE coding. By developing a wrapper language that generates SPICE code, the developers of SML have given the typical engineer the power to easily manipulate cumbersome circuit configurations, while harnessing the full power of the latest SPICE engine technologies. SML can be done without the hassle of being distracted by the nuances of any one particular SPICE implementation.

1.2 Introduction

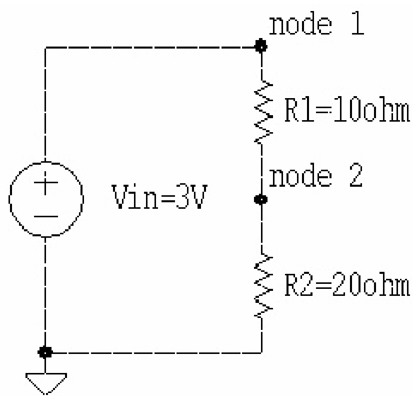
In any engineering discipline, simulation serves as an invaluable tool. The time and energy conserved in early development stages can be re-invested in the overall quality and optimization of the design. For electrical engineering, SPICE (**S**imulation **P**rogram with **I**ntegrated **C**ircuit **E**mphasis) serves as a computer program designed to assist with the simulation of analog electronic circuits. It has become the industrial standard for such experimentation. Early advancement in this area of software engineering saw the development of CANCER (Computer Analysis of Non-Linear Circuits Excluding Radiation) by Ron Roher of the University of California – Berkeley and three other predecessors of modern SPICE variants in the 1970s. In the years since, newer software releases have added greater functionally by introducing support for dynamic and semi-conductor circuit elements and improved analysis.

In order to use these products efficiently, designers must be familiar with the general "nodal" language that is used to create "net-lists" that represent circuits. In addition to being very cryptic, such languages can also vary from one particular SPICE implementation to another. Furthermore, the limitations of the language can make dynamic circuit configuration and manipulation very time-consuming.

SML serves as an interface between the designer's circuit schematic and the subsequent "netlist" that will be pumped into the SPICE translator. It hides the complexity and the implementation-dependent details of any particular SPICE version. It provides portability that allows compliant code to be generated into any particular SPICE dialect. It also provides greater programming functionality to allow for easier manipulation of circuit elements. Furthermore, the readability of the language makes it easier for engineers to collude on simulation parameters and results without spending a lot of time trying to parse a fellow designer's net-list. This language also serves as a viable entry point for further research into the development of high-level languages to model all types of biological and industrial node-based networks.

1.3 Background

What else is there to know about SPICE? SPICE simulation files are created by defining all nodes, and the circuit elements between each node. There are many variants of SPICE available to engineers. P-SPICE is a PC version commonly used by many researchers as a tool to create circuits, whereas H-SPICE is a UNIX based version. SPICE offers different analyses of circuits such as: AC Analysis, DC Analysis, and Transient Analysis. Below is a sample of circuit and the Spice code that would generate it:



Spice Code:

Title – My Circuit's Spice Code

.options post reltol=1e-6

.op

Vin 1 0 3

R1 1 2 10

R2 2 0 20

.END

As you can see, the overly-structured code above is not easily read unless you understand SPICE programming and take the time to think about the circuit's topology. Thus, SML will offer a simpler and more intuitive approach to simulating and manipulating circuits by creating a wrapper for SPICE programming.

1.4 Language Features

1.4.1 Scope

Every object has global scope, so no matter where an object is created its name can be referred to from anywhere in the program.

1.4.2 Object Types

SML supports the following Circuit Elements :

RES	: a resistor
CAP	: a capacitor
IND	: an inductor
DIODE:	: a diode
BJT	: a bi-polar junction transistor
MOSFET	: a metal oxide semiconductor field effect transistor
VS	: a voltage source
CS	: a current source

1.4.2 Spice Insertions

Spice insertions allow the user the ability to insert hspice code into their SML program. This provides the user with the full functionality of the hspice engine. The user can also insert SML objects within the hspice insertions. For more details see Spice Insertions: Language Reference Manual.

1.4.3 Lists

The most important property of SML is the ability to intuitively define a circuit. The core structure is the List object. By using this abstract List objected, different components can be added and connected, attendant values can be set and initial conditions can be stipulated. The use of list objects allow for the creation of sub-circuits that can be reduplicated over and over without the hassle of redefining them. Lists are 1-indexed, so L[1] gives the first element in list L.

1.5 Properties of circuit objects

The basic components of any circuit have inherent properties and terminals associated with them. For example, a Resistor object would have a resistance property and a positive and negative terminal:

```
res res1
res1.resistance = 3
res1.pos        :returns an object representing the positive terminal of res1
res1.neg        :returns an object representing the negative terminal of res1
```

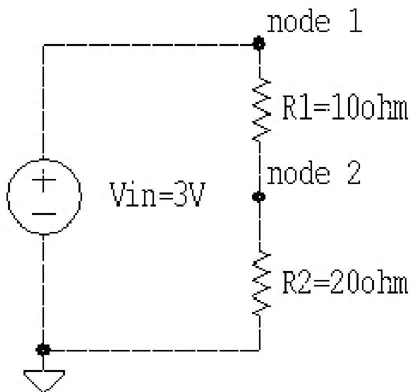
See Language Reference Manual for a full listing of Circuit Elements.

2. Tutorial

This tutorial will provide a novice user with instructions on creating a SML source file and translating it into compliant hspice code.

2.1 Sample SML Program

The first step is to create an SML source file. Below is an example of a circuit and the SML source code that corresponds to it.




```
$[ .op \n.dc #Vin 1 10 1\n.print dc v(#Vin.pos ) v(#r1.pos ,#r1.neg )\n]$
res r1
res r2
vs Vin
Vin.voltage = 3
r1.resistance = 10
r2.resistance = 20
Vin.pos->r1.pos
r1.neg->r2.pos
Vin.neg->ground
R2.neg->ground
```

Save the above code in a file name myCircuit.sml.

First we create 2 resistors (r1,r2) and a voltage source (Vin). Next, we set the voltage and the resistance for these components. The last five lines of code connect the components into a circuit.

2.2 Interpreting and Translating

After creating a SML source file, run the SML engine (sml.exe) on this file. To do this, the user can specify the input file and the output file for the SML engine to interpret the SML code and translate it into hspice. If no command line arguments are specified, the engine will look for a default input file ("input.txt") and generate the default output file ("output.sp")

Example of user defined input and output files with the sml engine:

```
>sml myCircuit.sml myCircuit.sp
```

2.3 Analyzing Output

After running the SML engine on the myCircuit.sml the following will be generated in myCircuit.sp:

```
This spice list generated by SML Compiler for myCircuit.sml
*options
.options post reltol=1e-6
 *simulation commands, all parameters (models, etc)
.op
.dc  vVin 1 10 1
.print dc v( node1 ) v( node1 , node2 )
rr1 node1 node2 20.00
rr2 node2 0 10.00
vVin node1 0 3.00
.end
```

This code is hspice compliant and can now be run in the hspice simulator.

3. Language Reference Manual

3.1 Introduction

SML has been proposed as a means of simplifying SPICE coding. By developing a wrapper language that generates SPICE code, the developers of SML have given the typical engineer the power to easily manipulate cumbersome circuit configurations, while harnessing the full power of the latest SPICE engine technologies. SML can be done without the hassle of being distracted by the nuances of any one particular SPICE implementation.

3.2 Lexical Conventions

3.2.1 Comments

Multiple line comments begin with "${" and end with "$}". Single line comments begin with a single "$" and are terminated by an end of line character.

3.2.2 Identifiers

An identifier is a letter followed by any combination of letters (a...z or A...Z), numbers and underscores ("_"). Identifiers are case-sensitive, and cannot be identical to any keyword. Identifiers are the valid names that can be given to variables.

3.2.3 Numbers

A number is a sequence of digits followed by an optional decimal point which may itself be followed by more digits.

3.2.4 Strings

A string is a sequence of ASCII characters surrounded in double quotes (" "). Just as in C, the character '\' acts as an escape character. '\t' represents a tab, '\n' a line feed, '\r' a carriage return and '\\' is used to represent a single '\'.

3.2.5 Objects

An object is any variable of type int, float, list, string etc. All variables are implemented as generic objects that internally store information about their type and where their data is stored.

3.2.6 Numerical Objects

Numerical values are stored in int and float objects. In practice, int values store the equivalent data of c++ long values and floats store the equivalent data of c++ double values.

3.2.7 Tokens

Let **n** denote a positive integer.
Let expr**n** be any expression that evaluates to an object.
Let list**n** be an expression that evaluates to a list object.
Let num**n** be an expression that evaluates to an int or float object.
Let string**n** be an expression that evaluates to a string object.
Let var**n** be a valid identifier (variable name).
Let vartype**n** be a valid data type, such as "int", "float", "list" or "string".

( ) (expr1)

Guarantees that expr1 will be evaluated before it is combined with surrounding expressions.

{ } {expr1, expr2, expr3}

Constructs a list containing the objects returned by expr1, expr2 and expr3. Any number of expressions can be enclosed in this manner to construct a list on the fly.

[ ] list1[num1]

As long as num1 evaluates to a positive integer N, the Nth element of list1 is returned.

,

The comma token separates items in lists that are constructed on the fly and separates input parameters to functions

+ num1 + num2
- num1 - num2
* num1 * num2

/ num1 / num2

These return the sum, difference, product and, quotient (respectively) of num1 and num2 (as a numerical object). Neither num1 nor num2 is effected by these operations.

= expr1 = expr2

This sets the value of the object returned by expr1 to the value of the object returned by expr2. If expr1 and expr2 evaluate to lists then the list object returned by expr1 is left containing elements that have data identically matching those in the result of expr2. Note that after using the '=' operator, modifying the value (or elements) of what is returned by expr1 will have no effect on the value (or elements) of what is returned by expr2. In particular, if expr1 is a list and expr2 is a list containing SPICE objects then any connections between these objects will be preserved, so that in the left hand list the nth SPICE object will connect to the kth SPICE object in exactly the manner that they were connected in the right hand list for all positive integers n and k.

== expr1 == expr2

Returns an integer of value 1 if expr1 and expr2 contain data that is identical. Otherwise returns an integer of value 0. If expr1 and expr2 evaluate to lists then each element of the lists must contain identical data for this expression to return 1.

> num1 > num2
< num1 < num2
>= num1 >= num2
<= num1 <= num2

Returns an integer of value one if num1 has (respectively) a greater, smaller, greater or equal, or smaller or equal value than num2. Otherwise, returns a zero valued integer.

@ string1 @ string2

Returns a string object that is the result of appending string2 to the end of string1. Neither string1 nor string2 is affected by this operation.

@ list1 @ expr

Returns a list object that is the result of appending expr to the end of list1.

#list1

Returns the number of elements in list1 as an integer value.

#string1

Returns the number of characters in string1 as an integer value.

. expr1.property

Here the dot (.) operator returns an object representing some property of the object returned by expr1. Modifying this property object modifies the corresponding property of expr1. Depending on the type that expr1 evaluates to, property could be of any valid data type.

-> expr1 -> expr2

If expr1 and expr2 evaluate to node objects, this creates a SPICE connection or circuit wire between them. In particular, this can be used to bind one terminal of a device to a terminal of a different device. For example, resistor1.pos -> capacitor1.neg creates a wire (connection) between the positive terminal of resistor1 and the negative terminal of capacitor1 in our generated SPICE circuit.

print print(object_name)

The print operator will print to standard out the value of the object. If the object is a circuit element, the default property will be printed out. For example:

```
res r1
r1.resistance=5
print(r1)
```

This will print out the resistance for the resistor r1. This code prints out the exact same value if print(r1) was replaced with print(r1.resistance). If printing a list, the values of each element in the list will be printed on a new line.

3.2.8 Keywords

Below is a list of the keywords in SML:

if	list	mosfet
while	string	ground
else	res	print
cap	bjt	
ind	diode	
int	cs	
float	vs	

3.3 Data Types

3.3.1 Basic Object Types

float

Stores a c++ double floating point value.

int

Stores a c++ long integer value.

list

Stores a list of objects.

string

Stores a sequence of characters.

3.3.2 SPICE Object types

node

A SPICE node. These objects are stored within resistors, capacitors, etc. to represent the various terminals of a SPICE object.

For all of the following object, obj.pos returns a node object representing the positive terminal of the object obj, and obj.neg returns a node representing its negative terminal. In addition, obj.argument returns a string representing the parameters used to define the element. This can be used to replace a constant value.

res

A SPICE resistor object.

obj.resistance returns a float objecting representing the resistance of a resistor object obj.

cap

A SPICE capacitor object.

obj.capacitance returns a float objecting representing the capacitance of a capacitor object obj.

obj.initial_voltage returns a float objecting representing the initial voltage of a capacitor object obj.

ind

A SPICE inductor object.

obj.inductance returns a float objecting representing the inductance of an inductor object obj.

obj.initial_current returns a float objecting representing the initial current of a inductor object obj.

cs

A SPICE current source object.

obj.current returns a float objecting representing the current of a current source object obj.

vs

A SPICE voltage source object.

obj.voltage returns a float objecting representing the voltage of a voltage source object obj.

Below are the semi-conductor devices that SML supports. The obj.argument field should be used to specify the name of the model that is being used for this device for this simulation. Models can be define using inserting spice code into your SML program.

diode

A SPICE diode object.

Like the two terminal objects above, diodes have a pos and neg terminal that can be reference by obj.pos and obj.neg.

bjt

A SPICE bjt object.

Bjt objects have collector, emitter and base terminals which can be referenced by obj.collector, obj.emitter, and obj.terminal respectively.

mosfet

A SPICE mosfet object.

Mosfets have drain, source and gate terminals which can be referenced by obj.drain, obj.source, and obj.gate respectively.

3.4 Expressions and Statements

An expression is a syntactically permissible sequence of tokens, keywords and identifiers. Some expressions evaluate to objects and others do not. A statement is an expression terminated by one or more end of line character.

3.4.1 Primary Expressions

3.4.1.1 Identifiers

An identifier is a left or right value expression.

3.4.1.2 Constants

A constant is either a number or a string that is a right-value expression.

3.4.2 Arithmetic Expressions

Use of operators such as +,-,*, / to modify primary expressions. Multiplication and division take precedence over addition and subtraction.

3.4.3 Relational Expressions

Comparison using the < (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to), == (equal), != (not equal) operators on two primary expressions.

3.5 Statements

3.5.1 Assignment

An assignment assigns a constant or expression to a specified identifier.
< *left_expression* > = < *right_expression* >

3.5.2 Conditional Statement

Let exprn denote an expression where n is a positive integer

if if (expr1) { expr2}

Evaluates expr2 if and only if expr1 evaluates to a non-zero integer. An error is thrown if expr1 does not evaluate to an integer.

else if (expr1) {expr2}
else {expr5}

Evaluates expression2 if expression1 evaluates to a non-zero integer. Otherwise, evaluates expr4 if expr3 evaluates to a non-zero integer. If neither expr2 nor expr4 was evaluated, then expr5 is evaluated. An error is thrown if expr1 or expr3 does not evaluate to an integer.

3.5.3 Iterative Statement

Let exprn denote an expression where n is a positive integer

while while (expr1) { expr2 }

Repeatedly evaluates expr2 until expr1 evaluates to an integer that has the value zero or the break keyword is reached in expr2. An error is thrown if expr1 does not evaluate to an integer.

3.6 Operator precedence

SML operators follow the following precedence ordering (from weakest to strongest):

```
,
= ->
==
< > <= >=
@
+ -
* /
#
```

()
{}
. []

3.7 SPICE Insertion

Lines of SPICE can be included directly in SML programs. These lines will be inserted into the final generated SPICE file in the order that they are executed in the SML program and will appear higher in the file than all SML related SPICE circuit elements and nodes that are generated. The syntax is as follows:

$[SPICE_commands]$

In particular, we can refer to SML variables in our SPICE insertions. To do this we essentially inject an SML object reference into our injected lines of SPICE. This SML reference will be resolved and the reference will be replaced with the corresponding SPICE circuit element name or the value of the specific object parameter. These SML injections within SPICE injections must be preceded by the # token, and they must be followed by a blank space. Thus, if in our SML we have a resistor object called R we might inject the following SPICE:

$[SPICE_commands #R more_SPICE_commands#R.resistance
even_more_SPICE_commands]$

The interpreter will then replace #R by the appropriate SPICE referring to the circuit element R and #R.resistance will be replaced by the resistance value of resistor R. This power to inject allows SML to harness the power of any SPICE commands that are not directly implemented, such as those used to measure currents and voltages through objects and at nodes. Any hspice reference manual can be used to find a listing of valid simulation commands

3.8 Sample SPICE insertions

$[.op]$

> This statement instructs SPICE to compute the DC steady state operating point for the circuit: voltage at the nodes, current in each voltage source, and the operating point for each element.

$[.dc #obj.param start stop incr]$

> The .dc analysis allows you to sweep a variable var and perform a DC steady state solution of each value of that variable. The variable can be the name of an independent source or any element parameter. Start and stop are the starting and

final values of obj.param. And, incr is the incremental value of the parameter being swept. Additionally, the sweep is done independently of the value of obj.param at simulation time.

$[.tran tincr stop START=tini UIC]$

This statement allows for a transient analysis to be conducted in the time domain for basic circuit elements where tincr is the time increment for calculating the variables, tstop and tini are the times at which the transient analysis stops and starts, respectively. The statement UIC (Use Initial conditions) causes SPICE to ttake the specified the initial values of dynamical components such as capacitors and inductors into account. Omitting the START statement is equivalent to setting START=0.

$[.ac TYPE np fstart fstop]$

This statement allows for time-invariant dynamical circuit analysis to be done in the frequency domain where fstart denotes the starting frequency, fstop denotes the final frequency, and TYPE defines how the np points on the frequency axis are taken (DEC for decade variation) and (LOG for logarithmic variation). For AC analysis to be conducted, a voltage or current source must be identified as an AC source with a particular phase and magnitude.

$[.print <analysis type> v(#obj1), i(#obj2), v(#obj2.pos, #gnd)]

This statement represents the simplest way of reporting simulation results in the output log where v(#obj) denotes the voltage across the positive and negative terminals of the object and v(#obj.pos, #gnd) denotes the voltage of the positive terminal with respect to ground. I(obj) denotes the current through the object.

4. Project Plan

4.1 Phases

4.1.1 Planning Phase

The team began the planning phase immediately after being assigned the project. Meetings were scheduled once a week during this phase. During these meetings the team settled on the need that the language would satisfy. It was determined that SML would simplify the creation of SPICE code. The team also used this phase to resolve the implementation tools that would be used and the focus of each member's preparatory research.

4.1.2 Specification Phase

The specification phase was the most important time period for the group. The team settled on the special features of SML and how SML would interoperate with SPICE. This phase took longer than expected, lasting over a month. The team had some minor setbacks in agreeing on the contribution that SML would make to computer assisted circuit analysis in general.

4.1.3 Development Phase

The development phase was a five week period which started with a clear division of labor. It was decided that Spencer would develop the language's grammar and handle the static and semantic analysis. Michael and Robert were assigned the interpreter stage which consisted of traversing the abstract syntax tree. Ronald was assigned the task of generating compliant SPICE code according to the specifications of the hspice variant. He also was given the responsibility of parsing injected SPICE code that occurred within a SML program. The labor was divided carefully so that none of the latter stages depended on each other. However, everything did depend on the framework that Spencer established. During the code consolidation phase, Robert and Spencer worked together to finalize the front end of the system and Michael and Ronald worked to tweak the back end of the system.

4.1.4 Testing Phase

Individual testing was done during each parts own development, but at the end the team collaborated and put the language to the test. The testing Phase was a three day period where all three parts were combined and tested with many sample circuits. Simple circuits were designed, implemented in SML and the resultant hspice code was tested in the hspice simulator that is installed on servers maintained by the Department of Electrical Engineering. To be sure the generated hspice corresponded to the original

circuit topology, hand comparisons were done on the results. As a further sanity check, the output files of hspice network lists (netlists) that the SML translator generated were compared with the output files of hspice netlists that were coded by hand.

4.2 Programming Style

The main programming guidelines that we focused on were writing clear, well understood, and well commented C++ code. Spencer's initial work on the framework set the model for naming identifiers in other parts of the code. While strict programming styles were not enforced and not needed because of efficient labor division, it was decided that all constants would be all capital letters and any class identifiers would be start with a capital letter. Furthermore, a common C++ header file was to maintain all of the utility functions and class definitions the system's implementation would require.

4.3 Project Roles

Spencer Greenberg	Worked with Antlr to create the lexer and parser, Handled static and semantic analysis, designed framework of language's implementation
Rob Toth	Wrote the main syntax interpreter and implemented the language's operators, worked with Spencer on debugging And testing the Front-end
Michael Apap	Helped manage the project's progression, worked to debug the interpreter and the code generator, worked on the documentation, helped setup development environment , created the test cases
Ron Alleyne	Helped layout language basis and features and provided tutorials on basic circuit simulation and analysis to other team members , implemented code generator and "spice injector", helped to ensure SML's interoperability with SPICE, worked on the documentation

4.4 Project Timeline

Date	Activity
9/9/04	Group Formation
9/15/04	General Idea of Language Confirmed
9/28/04	White Paper Submitted
10/7/04	Language Topic and Features Confirmed
10/21/04	Language Reference Manual Submitted
11/18/04	Lexer and Parser Completed

12/1/04	Static and Semantic Analysis Completed
12/18/04	Interpreter and Code Generator Completed
12/20/04	Project Presentation and Report Completed

4.5 Development Environment

The Antlr code for the Lexer and Parser were written in a UNIX environment and generated C++ files. The Static and Semantic Analysis implementation was done in C++ using XCode for MacOS. The interpreter and the code generator were done using Microsoft Visual Studio C++ (VSC++) for Windows. Antlr's generated libraries were installed in both Integrated Development Environment (IDE) to allow for successful compilation of the entire project

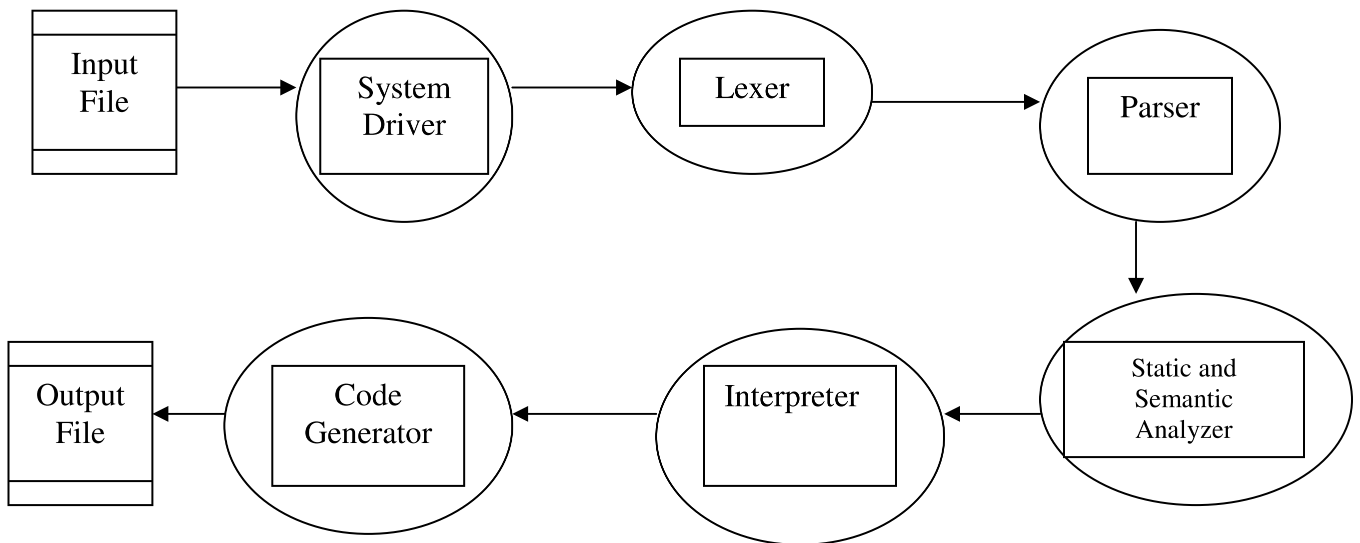
4.6 Project Log

Date	**Activity**
9/9/04	Group Formation
9/15/04	General Idea of Language Confirmed
9/28/04	White Paper Submitted
10/7/04	Language Topic and Features Confirmed
10/14/04	Meeting with Professor on Project Goals
10/16/04	Language Grammar Development Complete
10/21/04	Language Reference Manual Submitted
11/1/04	Meeting to Decide Implementation Labor Division
11/18/04	Lexer and Parser Completed
12/1/04	Static and Semantic Analysis Completed
12/18/04	Interpreter and Code Generator Completed
12/18/04	Testing of sample circuits completed.
12/20/04	Project Presentation and Report Completed

5. Architectural Design

5.1 Components of Translator

As the diagram below depicts, a sml source file is passed to the main driver of the system and lexically analyzed, parsed and undergoes static and semantic testing before it is interpreted. Once interpreted, the resultant state of the circuit is passed to the code generator.



5.2 Specifics

5.2.1 Lexer

The lexer takes a stream of characters and converts them into a stream of tokens. Comments and white space are ignored. The output of the lexer is passed to the parser.

5.2.2 Parser

The parser takes a stream of tokens and builds from it an abstract syntax tree based on our ANTLR grammar.

5.2.3 Static and Semantic Analyzer

The static semantic analyzer walks through the abstract syntax tree. For each node corresponding to a binary operation the types of the left and right expression are tested for consistency. The symbol table is also constructed during this stage and memory is allocated for constant expressions and local variables.

5.2.4 Interpreter

The interpreter walks through the abstract syntax tree after the static and semantic analyzer has verified that it is correct. During this part, the objects are set in memory and operators are evaluated. It is during this stage that conditionals, loops, SML connections and objects are evaluated. After the interpreter has gone through the abstract syntax tree, it has altered the symbol table to have all the appropriate information based on the program.

5.2.5 Code Generator

The code generation functionality is implemented in codeGen.h. Its main purpose is to take the symbol table of circuit element object as its input and create a netlist of the input circuit's topology. Before the netlist is generated, the functions nodeWalker() and nodeCruncher are used to minimize the number of circuit element terminals or nodes needed to identify connections in the circuit. This minimizing is essential to the generation of an accurate netlist. The actual code generation is a simple walk of the symbol table coupled with the outputting fo compliant hspice code.

5.3 Extra Components

5.3.1 Error Checking

To ease the understanding of what the SML engine was trying to evaluate, Spencer wrote a Crash_and_Burn function which outputted to the screen the line and any message that the programmer felt would be helpful in trying to debug during runtime.

5.3.2 "SPICE" Injection

An important feature in the language's implementation is the allowance of SPICE code in an SML program. This allots the user all of the tools of the underlying hspice circuit simulation software. This is implemented by passing the SPICE code along the process flow along to the code generator. The user is allowed to reference SML objects and their properties by using a special delimiter (see Language Properties). This was implemented in the function SMLInjecter() function.

6 Test Plan

The testing of SML was divided into two stages: unit testing and integrated testing. Spencer and Rob worked on the unit testing of the translater's front end and Michael and Ron worked on the unit testing of the system's back end.

6.1 Unit Testing

To facilitate effective unit testing, the project was divided into three main modules. The first module consisted of the lexer,parser and static/semantic analyzer, the second module consisted of the interpreter and third module housed the code generator and the spice injection engine. Each of the modules were tested individually to isolate and correct problems before integration. To test the first and second modules, sample sml files were created and analyzed until deficiencies were eliminated. To test the third module, a sample circuit was designed and connected as it would be by the second module and it state represented with the creation of a dummy symbol table. This dummy symbol table was used as the test bed for module 3.

6.2 Integrated Testing

Once unit tested was completed, the team worked to demonstrate the interoperability of the system. This testing involved taking the same sample circuits that were used in the unit testing of module 3 and plugging the representative SML code into module 1. The outputted spice files were examined for consistency with the successful results of module 3 test trials. As a final test, the outputted spice files were ran through the version of the hspice simulator running on servers maintained by the Department of Electrical Engineering.

6.3 Test Tools

The team decided not to use automated testing because of the nature of the programming language and system's input and output. Real circuits had to be design and drawn and inputted into the system. The resultant netlists were then manually analyzed to ensure their integrity and consistency with the original drawings. This form of testing was difficult, but it was the only way to be absolutely certain of the correctness of our language's implementation. An invaluable tool was Spencer's and Ron's knowledge of circuits from electrical engineering coursework. Additionally, the unit testing and integrated testing were aided by the use of specialized functions that allowed for easy debugging. For debugging purposes, the print function was implemented to display runtime values SML objects.

6.4 Sample Testing

6.4.1 Basic Test

Input file: inputbasic.sml

```
res r1
vs vin
r1.resistance=50
vin.voltage=500
r1.pos->vin.pos
r1.neg->ground
vin.neg->ground
```

Output file: outputbasic.sp

```
This spice list generated by SML Compiler for inputbasic.sml
*options
.options post reltol=1e-6
 *simulation commands, all parameters (models, etc)
rr1 node1 0 50.00
vvin node1 0 500.00
.end
```

HSPICE results running on outputbasic.sp

```
Using: /usr/cad/meta/U-2003.09/linux/hspice
 ******  HSPICE    -- U-2003.09  (20030718) 21:15:39  12/19/2004  linux
  Copyright (C) 2003 Synopsys, Inc. All Rights Reserved.
  Unpublished-rights reserved under US copyright laws.
  This program is protected by law and is subject to the
  terms and conditions of the license agreement found in:
    /usr/cad/meta/U-2003.09/license.txt
  Use of this program is your acceptance to be bound by this
  license agreement. HSPICE is the trademark of Synopsys, Inc.
  Input File: outputbasic.sp
 lic:
 lic: FLEXlm: v6.1g
 lic: USER:   rja2001              HOSTNAME: odessa.cisl.columbia.edu
 lic: HOSTID: 000d56f6032f         PID:      20128
 lic: Using FLEXlm license file:
 lic: /usr/cad/scl/admin/license/license.dat
 lic: Checkout hspice;  Encryption code: BD40E79D27978B35D58A
 lic: License/Maintenance for hspice will expire on 29-may-2005/2004.09
```

```
lic: 1(in_use)/50 FLOATING license(s) on SERVER cadserv1.cisl.columbia.edu
lic:
Init: read install configuration file: /usr/cad/meta/U-2003.09/meta.cfg
Init: hspice initialization file: /usr/cad/meta/U-2003.09/hspice.ini
* reading file:  /usr/cad/meta/u-2003.09/hspice.ini
.options post reltol=1e-6
 *simulation commands, all parameters (models, etc)
rr1 node1 0 50.00
vvin node1 0 500.00
.end

       ***** job concluded
1 ******  HSPICE    -- U-2003.09  (20030718) 21:15:39  12/19/2004  linux
******
this spice list generated by sml compiler for inputbasic.sml
 ******  job statistics summary           tnom=  25.000 temp=  25.000
******

        total memory used         22 kbytes

 # nodes =     2 # elements=     2
 # diodes=     0 # bjts    =     0 # jfets    =     0 # mosfets =     0

   analysis      time      # points  tot. iter  conv.iter

   op point          0.00         1         0
   readin            0.00
   errchk            0.00
   setup             0.00
   output            0.00
        total cpu time          0.00 seconds
           job started at  21:15:39  12/19/2004
           job ended   at  21:15:39  12/19/2004


lic: Release hspice token(s)
HSPICE job outputbasic.sp completed.
Sun Dec 19 21:15:39 EST 2004
```

6.4.2 List Test

Input file: inputlist.sml

```
$[
.op
]$
vs vin
vin.voltage=10
list l
res r1
res r2
res r3
r1.pos -> r2.pos
r1.pos -> r3.pos
r1.neg -> r2.neg
r1.neg -> r3.neg
l @ r1
l @ r2
l @ r3
list g = l
g[1].pos -> l[1].neg

vin.neg->ground
l[1].pos->vin.pos
g[1].neg->ground
```

Output file: outputlist.sp

```
This spice list generated by SML Compiler for inputlist.sml
*options
.options post reltol=1e-6
 *simulation commands, all parameters (models, etc)

.op
vvin node1 0 10.00
rl_xyz1 node1 node2 0.00
rl_xyz2 node1 node2 0.00
rl_xyz3 node1 node2 0.00
rg_xyz4 node2 0 0.00
rg_xyz5 node2 0 0.00
rg_xyz6 node2 0 0.00
.end
```

HSPICE results on outputlist.sp

```
Using: /usr/cad/meta/U-2003.09/linux/hspice
 ****** HSPICE   -- U-2003.09  (20030718) 21:15:58  12/19/2004  linux
  Copyright (C) 2003 Synopsys, Inc. All Rights Reserved.
  Unpublished-rights reserved under US copyright laws.
  This program is protected by law and is subject to the
  terms and conditions of the license agreement found in:
    /usr/cad/meta/U-2003.09/license.txt
  Use of this program is your acceptance to be bound by this
  license agreement. HSPICE is the trademark of Synopsys, Inc.
  Input File: outputlist.sp
 lic:
 lic: FLEXlm: v6.1g
 lic: USER:   rja2001              HOSTNAME: odessa.cisl.columbia.edu
 lic: HOSTID: 000d56f6032f         PID:      20143
 lic: Using FLEXlm license file:
 lic: /usr/cad/scl/admin/license/license.dat
 lic: Checkout hspice;  Encryption code: BD40E79D27978B35D58A
 lic: License/Maintenance for hspice will expire on 29-may-2005/2004.09
 lic: 1(in_use)/50 FLOATING license(s) on SERVER cadserv1.cisl.columbia.edu
 lic:
 Init: read install configuration file: /usr/cad/meta/U-2003.09/meta.cfg
 Init: hspice initialization file: /usr/cad/meta/U-2003.09/hspice.ini
 * reading file:  /usr/cad/meta/u-2003.09/hspice.ini
 .options post reltol=1e-6
  *simulation commands, all parameters (models, etc)

 .op
 vvin node1 0 10.00
 rl_xyz1 node1 node2 0.00
 rl_xyz2 node1 node2 0.00
 rl_xyz3 node1 node2 0.00
 rg_xyz4 node2 0 0.00
 rg_xyz5 node2 0 0.00
 rg_xyz6 node2 0 0.00
 .end
 **warning**    0:rl_xyz1        defined in subckt 0              resistance limited to
1.000E-05

 **warning**    0:rl_xyz2        defined in subckt 0              resistance limited to
1.000E-05

 **warning**    0:rl_xyz3        defined in subckt 0              resistance limited to
1.000E-05

 **warning**    0:rg_xyz4        defined in subckt 0              resistance limited to
1.000E-05
```

```
**warning**    0:rg_xyz5       defined in subckt 0            resistance limited to
1.000E-05

**warning**    0:rg_xyz6       defined in subckt 0            resistance limited to
1.000E-05

1 ******  HSPICE   -- U-2003.09  (20030718) 21:15:58  12/19/2004  linux
******
this spice list generated by sml compiler for inputlist.sml
 ******  operating point information      tnom=  25.000 temp=  25.000
******
***** operating point status is all       simulation time is     0.
   node    =voltage      node    =voltage

+0:node1   =  10.0000  0:node2   =   5.0000


****  voltage sources

subckt
element  0:vvin
 volts     10.0000
 current   -1.5000x
 power     15.0000x


    total voltage source power dissipation=   15.0000x       watts

**** resistors

subckt
element  0:rl_xyz1  0:rl_xyz2  0:rl_xyz3  0:rg_xyz4  0:rg_xyz5  0:rg_xyz6
 r value   10.0000u   10.0000u   10.0000u   10.0000u   10.0000u   10.0000u
 v drop     5.0000     5.0000     5.0000     5.0000     5.0000     5.0000
 current  500.0000k  500.0000k  500.0000k  500.0000k  500.0000k  500.0000k
 power      2.5000x    2.5000x    2.5000x    2.5000x    2.5000x    2.5000x


          ***** job concluded
1 ******  HSPICE   -- U-2003.09  (20030718) 21:15:58  12/19/2004  linux
******
this spice list generated by sml compiler for inputlist.sml
 ******  job statistics summary          tnom=  25.000 temp=  25.000
******
```

```
         total memory used          22 kbytes

 # nodes =      3 # elements=      7
 # diodes=      0 # bjts    =      0 # jfets   =      0 # mosfets =      0

   analysis       time      # points  tot. iter  conv.iter

   op point          0.01          1          4
   readin           0.00
   errchk           0.00
   setup            0.00
   output           0.00
        total cpu time           0.01 seconds
           job started at  21:15:58  12/19/2004
           job ended   at  21:15:58  12/19/2004


 lic: Release hspice token(s)
HSPICE job outputlist.sp completed.
Sun Dec 19 21:15:59 EST 2004
```

6.4.3 Resistor Bank

Input file: inputresbank.sml

```
$[
.op
]$
vs vin
vin.voltage=10
int a = 1
list l
while(a < 5)
{
res r
r.resistance = 5000
r.pos->vin.pos
r.neg->ground
l@r
a=a+1
}
vin.neg->ground
```

Output file: outputresbank.sp

```
This spice list generated by SML Compiler for inputresbank.sml
*options
.options post reltol=1e-6
 *simulation commands, all parameters (models, etc)

.op
vvin node1 0 10.00
rl_xyz1 node1 0 5000.00
rl_xyz2 node1 0 5000.00
rl_xyz3 node1 0 5000.00
rl_xyz4 node1 0 5000.00
.end
```

HSPICE results running on outputresbank.sp

```
Using: /usr/cad/meta/U-2003.09/linux/hspice
 ******  HSPICE    -- U-2003.09  (20030718) 21:16:09  12/19/2004  linux
  Copyright (C) 2003 Synopsys, Inc. All Rights Reserved.
  Unpublished-rights reserved under US copyright laws.
  This program is protected by law and is subject to the
  terms and conditions of the license agreement found in:
    /usr/cad/meta/U-2003.09/license.txt
  Use of this program is your acceptance to be bound by this
  license agreement. HSPICE is the trademark of Synopsys, Inc.
  Input File: outputresbank.sp
 lic:
 lic: FLEXlm: v6.1g
 lic: USER:   rja2001              HOSTNAME: odessa.cisl.columbia.edu
 lic: HOSTID: 000d56f6032f         PID:      20158
 lic: Using FLEXlm license file:
 lic: /usr/cad/scl/admin/license/license.dat
 lic: Checkout hspice;  Encryption code: BD40E79D27978B35D58A
 lic: License/Maintenance for hspice will expire on 29-may-2005/2004.09
 lic: 1(in_use)/50 FLOATING license(s) on SERVER cadserv1.cisl.columbia.edu
 lic:
 Init: read install configuration file: /usr/cad/meta/U-2003.09/meta.cfg
 Init: hspice initialization file: /usr/cad/meta/U-2003.09/hspice.ini
 * reading file:  /usr/cad/meta/u-2003.09/hspice.ini
 .options post reltol=1e-6
  *simulation commands, all parameters (models, etc)

 .op
 vvin node1 0 10.00
 rl_xyz1 node1 0 5000.00
 rl_xyz2 node1 0 5000.00
 rl_xyz3 node1 0 5000.00
```

```
rl_xyz4 node1 0 5000.00
.end
1 ****** HSPICE   -- U-2003.09  (20030718) 21:16:09  12/19/2004  linux
******
this spice list generated by sml compiler for inputresbank.sml
 ******  operating point information      tnom=  25.000 temp=  25.000
******
***** operating point status is all       simulation time is     0.
   node    =voltage

+0:node1   =  10.0000


****  voltage sources

subckt
element  0:vvin
 volts     10.0000
 current   -8.0000m
 power     80.0000m


    total voltage source power dissipation=   80.0000m       watts

**** resistors

subckt
element  0:rl_xyz1  0:rl_xyz2  0:rl_xyz3  0:rl_xyz4
 r value    5.0000k    5.0000k    5.0000k    5.0000k
 v drop    10.0000    10.0000    10.0000    10.0000
 current    2.0000m    2.0000m    2.0000m    2.0000m
 power     20.0000m   20.0000m   20.0000m   20.0000m


         ***** job concluded
1 ****** HSPICE   -- U-2003.09  (20030718) 21:16:09  12/19/2004  linux
******
this spice list generated by sml compiler for inputresbank.sml
 ******  job statistics summary          tnom=  25.000 temp=  25.000
******

          total memory used         22 kbytes

 # nodes =     2 # elements=     5
 # diodes=     0 # bjts    =     0 # jfets   =     0 # mosfets =     0
```

```
     analysis      time      # points  tot. iter  conv.iter

     op point         0.00         1         3
     readin          0.00
     errchk          0.00
     setup           0.00
     output          0.00
          total cpu time          0.00 seconds
               job started at  21:16:09  12/19/2004
               job ended   at  21:16:10  12/19/2004


 lic: Release hspice token(s)
HSPICE job outputresbank.sp completed.
Sun Dec 19 21:16:10 EST 2004
```

6.4.4 Resistor Capacitor Bank 50

Input file: inputrescapbank50.sml

```
$[
.op
.print v( #vin )
]$


vs vin
vin.voltage=10
int a = 1
float f = 0.01
list l
while(a < 25)
{
res r
r.resistance = a * 100
r.pos->vin.pos
r.neg->ground
cap c
c.capacitance = f * 2
c.pos->vin.pos
c.neg->ground
c.initial_voltage= f * f
l@r
l@c
a=a+1
```

```
f=f+0.01
}
vin.neg->ground
```

Output file: outputrescapbank50.sp

```
This spice list generated by SML Compiler for inputrescapbank50.sml
*options
.options post reltol=1e-6
 *simulation commands, all parameters (models, etc)

.op
.print v(  vvin )
vvin node1 0 10.00
rl_xyz1 node1 0 100.00
cl_xyz2 node1 0 0.02 IC=0.00
rl_xyz3 node1 0 200.00
cl_xyz4 node1 0 0.04 IC=0.00
rl_xyz5 node1 0 300.00
cl_xyz6 node1 0 0.06 IC=0.00
rl_xyz7 node1 0 400.00
cl_xyz8 node1 0 0.08 IC=0.00
rl_xyz9 node1 0 500.00
cl_xyz10 node1 0 0.10 IC=0.00
rl_xyz11 node1 0 600.00
cl_xyz12 node1 0 0.12 IC=0.00
rl_xyz13 node1 0 700.00
cl_xyz14 node1 0 0.14 IC=0.00
rl_xyz15 node1 0 800.00
cl_xyz16 node1 0 0.16 IC=0.01
rl_xyz17 node1 0 900.00
cl_xyz18 node1 0 0.18 IC=0.01
rl_xyz19 node1 0 1000.00
cl_xyz20 node1 0 0.20 IC=0.01
rl_xyz21 node1 0 1100.00
cl_xyz22 node1 0 0.22 IC=0.01
rl_xyz23 node1 0 1200.00
cl_xyz24 node1 0 0.24 IC=0.01
rl_xyz25 node1 0 1300.00
cl_xyz26 node1 0 0.26 IC=0.02
rl_xyz27 node1 0 1400.00
cl_xyz28 node1 0 0.28 IC=0.02
rl_xyz29 node1 0 1500.00
cl_xyz30 node1 0 0.30 IC=0.02
rl_xyz31 node1 0 1600.00
cl_xyz32 node1 0 0.32 IC=0.03
```

```
rl_xyz33 node1 0 1700.00
cl_xyz34 node1 0 0.34 IC=0.03
rl_xyz35 node1 0 1800.00
cl_xyz36 node1 0 0.36 IC=0.03
rl_xyz37 node1 0 1900.00
cl_xyz38 node1 0 0.38 IC=0.04
rl_xyz39 node1 0 2000.00
cl_xyz40 node1 0 0.40 IC=0.04
rl_xyz41 node1 0 2100.00
cl_xyz42 node1 0 0.42 IC=0.04
rl_xyz43 node1 0 2200.00
cl_xyz44 node1 0 0.44 IC=0.05
rl_xyz45 node1 0 2300.00
cl_xyz46 node1 0 0.46 IC=0.05
rl_xyz47 node1 0 2400.00
cl_xyz48 node1 0 0.48 IC=0.06
.end
```

HSPICE results on outputrescapbank50.sp

```
Using: /usr/cad/meta/U-2003.09/linux/hspice
 ******  HSPICE    -- U-2003.09  (20030718) 21:16:22  12/19/2004  linux
  Copyright (C) 2003 Synopsys, Inc. All Rights Reserved.
  Unpublished-rights reserved under US copyright laws.
  This program is protected by law and is subject to the
  terms and conditions of the license agreement found in:
    /usr/cad/meta/U-2003.09/license.txt
  Use of this program is your acceptance to be bound by this
  license agreement. HSPICE is the trademark of Synopsys, Inc.
  Input File: outputrescapbank50.sp
 lic:
 lic: FLEXlm: v6.1g
 lic: USER:   rja2001              HOSTNAME: odessa.cisl.columbia.edu
 lic: HOSTID: 000d56f6032f         PID:      20181
 lic: Using FLEXlm license file:
 lic: /usr/cad/scl/admin/license/license.dat
 lic: Checkout hspice;  Encryption code: BD40E79D27978B35D58A
 lic: License/Maintenance for hspice will expire on 29-may-2005/2004.09
 lic: 1(in_use)/50 FLOATING license(s) on SERVER cadserv1.cisl.columbia.edu
 lic:
 Init: read install configuration file: /usr/cad/meta/U-2003.09/meta.cfg
 Init: hspice initialization file: /usr/cad/meta/U-2003.09/hspice.ini
 * reading file:  /usr/cad/meta/u-2003.09/hspice.ini
 .options post reltol=1e-6
  *simulation commands, all parameters (models, etc)
```

```
.op
.print v(  vvin )
vvin node1 0 10.00
rl_xyz1 node1 0 100.00
cl_xyz2 node1 0 0.02 ic=0.00
rl_xyz3 node1 0 200.00
cl_xyz4 node1 0 0.04 ic=0.00
rl_xyz5 node1 0 300.00
cl_xyz6 node1 0 0.06 ic=0.00
rl_xyz7 node1 0 400.00
cl_xyz8 node1 0 0.08 ic=0.00
rl_xyz9 node1 0 500.00
cl_xyz10 node1 0 0.10 ic=0.00
rl_xyz11 node1 0 600.00
cl_xyz12 node1 0 0.12 ic=0.00
rl_xyz13 node1 0 700.00
cl_xyz14 node1 0 0.14 ic=0.00
rl_xyz15 node1 0 800.00
cl_xyz16 node1 0 0.16 ic=0.01
rl_xyz17 node1 0 900.00
cl_xyz18 node1 0 0.18 ic=0.01
rl_xyz19 node1 0 1000.00
cl_xyz20 node1 0 0.20 ic=0.01
rl_xyz21 node1 0 1100.00
cl_xyz22 node1 0 0.22 ic=0.01
rl_xyz23 node1 0 1200.00
cl_xyz24 node1 0 0.24 ic=0.01
rl_xyz25 node1 0 1300.00
cl_xyz26 node1 0 0.26 ic=0.02
rl_xyz27 node1 0 1400.00
cl_xyz28 node1 0 0.28 ic=0.02
rl_xyz29 node1 0 1500.00
cl_xyz30 node1 0 0.30 ic=0.02
rl_xyz31 node1 0 1600.00
cl_xyz32 node1 0 0.32 ic=0.03
rl_xyz33 node1 0 1700.00
cl_xyz34 node1 0 0.34 ic=0.03
rl_xyz35 node1 0 1800.00
cl_xyz36 node1 0 0.36 ic=0.03
rl_xyz37 node1 0 1900.00
cl_xyz38 node1 0 0.38 ic=0.04
rl_xyz39 node1 0 2000.00
cl_xyz40 node1 0 0.40 ic=0.04
rl_xyz41 node1 0 2100.00
cl_xyz42 node1 0 0.42 ic=0.04
rl_xyz43 node1 0 2200.00
```

```
cl_xyz44 node1 0 0.44 ic=0.05
rl_xyz45 node1 0 2300.00
cl_xyz46 node1 0 0.46 ic=0.05
rl_xyz47 node1 0 2400.00
cl_xyz48 node1 0 0.48 ic=0.06
.end


**warning** in element=  0:cl_xyz12         defined in subckt 0
    capacitance =   0.120     >= 0.1 farad, please verify it.


**warning**  attempt to reference undefined node     0:vvin
  branch - output ignored

**warning** print/plot=out0         of analysis=tran
contains unused outputs
this entire output statement is ignored

1 ******  HSPICE    -- U-2003.09  (20030718) 21:16:22  12/19/2004  linux
******
this spice list generated by sml compiler for inputrescapbank50.sml
 ******  operating point information      tnom=  25.000 temp=  25.000
******
***** operating point status is all       simulation time is     0.
   node    =voltage

+0:node1   =  10.0000


****  voltage sources

subckt
element  0:vvin
 volts     10.0000
 current -377.5958m
 power      3.7760


    total voltage source power dissipation=    3.7760        watts

**** resistors

subckt
element  0:rl_xyz1  0:rl_xyz3  0:rl_xyz5  0:rl_xyz7  0:rl_xyz9  0:rl_xyz11
 r value  100.0000   200.0000   300.0000   400.0000   500.0000   600.0000
```

```
 v drop    10.0000    10.0000    10.0000    10.0000    10.0000    10.0000
 current  100.0000m   50.0000m   33.3333m   25.0000m   20.0000m   16.6667m
 power      1.0000   500.0000m  333.3333m  250.0000m  200.0000m  166.6667m

subckt
element  0:rl_xyz13 0:rl_xyz15 0:rl_xyz17 0:rl_xyz19 0:rl_xyz21 0:rl_xyz23
 r value  700.0000   800.0000   900.0000     1.0000k    1.1000k    1.2000k
 v drop    10.0000    10.0000    10.0000    10.0000    10.0000    10.0000
 current   14.2857m   12.5000m   11.1111m   10.0000m    9.0909m    8.3333m
 power    142.8571m  125.0000m  111.1111m  100.0000m   90.9091m   83.3333m

subckt
element  0:rl_xyz25 0:rl_xyz27 0:rl_xyz29 0:rl_xyz31 0:rl_xyz33 0:rl_xyz35
 r value    1.3000k    1.4000k    1.5000k    1.6000k    1.7000k    1.8000k
 v drop    10.0000    10.0000    10.0000    10.0000    10.0000    10.0000
 current    7.6923m    7.1429m    6.6667m    6.2500m    5.8824m    5.5556m
 power     76.9231m   71.4286m   66.6667m   62.5000m   58.8235m   55.5556m

subckt
element  0:rl_xyz37 0:rl_xyz39 0:rl_xyz41 0:rl_xyz43 0:rl_xyz45 0:rl_xyz47
 r value    1.9000k    2.0000k    2.1000k    2.2000k    2.3000k    2.4000k
 v drop    10.0000    10.0000    10.0000    10.0000    10.0000    10.0000
 current    5.2632m    5.0000m    4.7619m    4.5455m    4.3478m    4.1667m
 power     52.6316m   50.0000m   47.6190m   45.4545m   43.4783m   41.6667m


          ***** job concluded
1 ******  HSPICE    -- U-2003.09  (20030718) 21:16:22  12/19/2004  linux
 ******
this spice list generated by sml compiler for inputrescapbank50.sml
 ******  job statistics summary           tnom=  25.000 temp=  25.000
******

          total memory used          154 kbytes

 # nodes =      2 # elements=     49
 # diodes=      0 # bjts    =      0 # jfets   =      0 # mosfets =      0

    analysis      time      # points  tot. iter  conv.iter

    op point          0.00         1          3
    readin            0.01
    errchk            0.00
    setup             0.00
    output            0.00
          total cpu time          0.01 seconds
```

```
job started at  21:16:22  12/19/2004
job ended   at  21:16:23  12/19/2004


 lic: Release hspice token(s)
HSPICE job outputrescapbank50.sp completed.
Sun Dec 19 21:16:23 EST 2004
```

7 Lessons Learned

7.1 Spencer Greenberg

I learned from this project that it is crucial to decide how each element of a language will be implemented before deciding on the language itself. In our case, we had to augment our abstract syntax tree nodes in order to accommodate the various functionalities that we had planned to implement. This occasionally involved demolishing old code that could not support this functionality. Furthermore, I learned the extreme importance of code reuse. In a number of cases I found the same lines of code appearing many times, and would have saved myself a lot of trouble had I written functions to handle frequently used tasks. I also think it would have been a good idea to test each finished portion of code before moving on since bugs are a lot easier to weed out when dealing with a few functions than (for example) when the interpreter crashes with no helpful messages.

7.2 Ron Alleyne

Our organization was very clear and straightforward; however when we tried to combine our efforts bugs began to develop. I learned that creating your own language requires a different type of workload then regular programming. Weekly deadlines and clear cut goals are a necessity. I also learned that testing as parts were programmed would have been easier than waiting till the end, and testing all of the code at one time.

7.3 Michael Apap

The main lesson learned was that no matter how much planning and organization is done, when all the parts come together there will be some connection issues. The three other members each had completed their own individual parts and when we put them together, unforeseeable issues developed. The outcome was worth the time and effort that we put in. I learned that developing a sound plan is a necessity when trying to develop something at this scale. I learned how trying to develop your own programming language is a completely different technique then just programming.

7.4 Robert Toth

After working on the Interpreter for SML I learned a lot on the power of programming languages. I was able to create my own while loops (nested!) and if statements, all at the same time using our SML objects to develop circuits. The results, after a few hours of fixing minor bugs, were impressive.

8 Appendix

8.1 Grammar File

Author: Spencer Greenberg

```
header {
#include "MyAST.h"
}

options {
	language="Cpp";
}

{
#include <iostream>
}

/*---------------------------------------------
Lexer
---------------------------------------------*/

class Lex extends Lexer;
options
{
	testLiterals = false;
	genHashLines = true;
	k=3;
	//charVocabulary = '\3'..'\377'; //allow all 8 bit characters
	charVocabulary='\u0000'..'\u007F'; // allow ascii
}


PLUS		: '+'	;
MINUS			: '-'	;
MULT		: '*'	;
DIV			: '/'	;
ASSIGN			: '='	;
COMPARE			: "=="	;
RASSIGN			: "&=" ;
RCOMPARE	: "&=="		;
COMMA			: ','	;
APPEND			: '@'	;
POUND			: '#'	;
LESS		: '<'	;
```

```
GREATER              : '>'   ;
LTE                  : "<="  ;
GTE                  : ">="  ;
ARROW                : "->"  ;
DOT                  : "."   ;

LPAREN               : '(' ;
RPAREN               : ')' ;

LBRACE               : '{' ;
RBRACE               : '}' ;

LBRACK               : '[' ;
RBRACK               : ']' ;

/*strings use the escape character '\'. Since ANTLR does as well, we denote it '\\' below.
*/

STRING_CONSTANT : '"'! ( ('\\'! ('"' | '\\' | 't' | 'n' | 'r') ) | ~('"' | '\\') )*  '"'! ;
//STRING_CONSTANT : '"'! (~( '"' | '\\'))* '"'!

protected LETTER : ('A'..'Z' | 'a'..'z') ;
protected DIGIT  : '0'..'9' ;

ID options {testLiterals = true;}
: LETTER (LETTER | DIGIT | '_')* ;

protected INTEGER_CONSTANT : (DIGIT)+ ;
protected FRACTION: ( /*nothing*/ | DOT (DIGIT)*) ;
//NUMBER: (INTEGER_CONSTANT FRACTION | DOT (DIGIT)+);
NUMBER: INTEGER_CONSTANT FRACTION;

LINEEND :
                ('\r' '\n') => '\r' '\n'      { newline(); }          //DOS
                | '\r'                                               { newline(); }
        //UNIX
                | '\n'                                               { newline(); } ;          //MAC

WS: (' ' | '\t')
        { $setType(ANTLR_USE_NAMESPACE(antlr)Token::SKIP); } ;

COMMENT: '$'!
                 (
                        (
```

```
                                '{'

        /*match multi line comment*/
                                ( options {greedy = false;}:
                                        (
                                                LINEEND | ~('\n'|'\r')
                                        )
                                )*
                                '}'! '$'!
                                {
$setType(ANTLR_USE_NAMESPACE(antlr)Token::SKIP); }
                        )


                        |
                        (
                        (/*nothing*/ | LINEEND | ~('[' | '{' | '\r' | '\n') (~('\n'|'\r'))*
LINEEND)                        /*match single line comment*/
                                {
$setType(ANTLR_USE_NAMESPACE(antlr)Token::SKIP); }
                        )

                        |

        /*match spice injection*/
                        ('[') => '['!
                        ( options {greedy = false;}:
                                (
                                        LINEEND | ~('\n'|'\r')
                                )
                        )*
                        ']'! '$'!
                )

                ;


/*---------------------------------------------
operator precedence (weakest to strongest):

,
= &= ->
AND
OR
== &==
< > <= >=
```

```
@
+ -
* /
.
#
()
{}
[]

This chart agrees with c++ operator precedence.
--------------------------------------------*/


/*--------------------------------------------
Parser
--------------------------------------------*/

class Pars extends Parser;
options
{
	codeGenMakeSwitchThreshold = 3;
	codeGenBitsetTestThreshold = 4;
	ASTLabelType = "RefMyAST";
	buildAST = true;
	k=2;
}

tokens
{
	AUTOLIST;		//token for a list generated on the fly of specified elements
	SAMETYPELIST;	//token for a list generated on the fly of specified type
	BRACKETEXPR;	//token for object[val] and object[val1,val2]
	PROGRAM;
	ELSE_IF;			//specifies that we've reached not just an 'else' but
an 'else if'
	SEQ_OF_EXPRS;		//specifies a sequence of expressions in a while
loop, if statement or function decleration
	SPICE;				//token for a spice injection
}

/*end of lines get recorded as LINEEND tokens. We can use the rule 'lines' to allow any
number of LINEENDs at a position*/
lines		: ((LINEEND)*)! ;
one_or_more_line  : ((LINEEND)+)! ;

program : lines! (expr lines!)* EOF!
```

```
{#program = #([PROGRAM, "program"], #program); }
;

datatype	: "string"^ | "int"^ | "float"^ | "list"^ | "res"^ | "cap"^ | "ind"^ | "vs"^ | "cs"^
| "node"^ | "diode"^ | "bjt"^ | "mosfet"^;			//if you include this rule the carots
are actually ignored.
datatypenotroot : "string" | "int" | "float" | "list" | "res" | "cap" | "ind" | "vs" | "cs" | "node" |
"diode" | "bjt" | "mosfet";

seq_of_exprs	: lines! (expr lines!)*
{#seq_of_exprs = #([SEQ_OF_EXPRS, "seq_of_exprs"], #seq_of_exprs); }
;

expr
: if_statement
| "while"^ LPAREN! expr RPAREN! lines! LBRACE! seq_of_exprs RBRACE!
| "print"^ expr1			//  print(3+5)
| "break"^
| "continue"^
| spice				/*spice injection!*/
| expr1
;


spice : COMMENT
{#spice = #([SPICE, "spice"], #spice); }
;

if_statement
	: "if"^ LPAREN! expr RPAREN! lines! LBRACE! seq_of_exprs RBRACE!
	| "else"^ (else_if)? lines! LBRACE! seq_of_exprs RBRACE!
	;

else_if : "if"! LPAREN! expr RPAREN!
{#else_if = #([ELSE_IF, "else_if"], #else_if); }
;

expr1  : expr2 ( (ASSIGN^ | RASSIGN^ | ARROW^) expr2)* ;
expr2  : expr3 ("and"^ expr3)* ;
expr3  : expr4 ("or"^ expr4)* ;
expr4  : expr5 ((LESS^ | GREATER^ | LTE^ | GTE^) expr5)? ;
expr5  : expr6 ( (COMPARE^ | RCOMPARE^) expr6)* ;
expr6  : expr7 ( (APPEND^) expr7)* ;
expr7  : expr8 ( (PLUS^ | MINUS^) expr8)* ;
expr8  : expr9 ( (MULT^ | DIV^) expr9)* ;
expr9  : expr10 (DOT^ ID)*;
```

```
expr10  : expr11 | POUND^ expr10 ;
expr11 : expr12 | LPAREN! expr1 RPAREN! ;                                    //need
to add a unary minus and plus operator!

expr12 : expr13 | autolist;

autolist : LBRACE! expr1 (COMMA! expr1)*  RBRACE!
{#autolist = #([AUTOLIST, "autolist"], #autolist); }
;

expr13 : NUMBER
        | STRING_CONSTANT^ (bracketexpr | /*nothing*/)
        | ID^ (/*nothing*/ | bracketexpr)
        | "ground"^
        | ("string"^ | "int"^ | "float"^ | "list"^ | "res"^ | "cap"^ | "ind"^ | "vs"^ | "cs"^ |
"node"^ | "diode"^ | "bjt"^ | "mosfet"^) (ID | bracketexpr)
        ;

bracketexpr  :  LBRACK! expr1 (COMMA! expr1)? RBRACK!
{#bracketexpr = #([BRACKETEXPR, "bracketexpr"], #bracketexpr); }
;


/*------------------------------------------------
TreeWalker
-not currently being used
------------------------------------------------*/

class Walk extends TreeParser;
options
{
        ASTLabelType = "RefMyAST";
}
{
        //can add c++ here if need be
}

expr: #(PLUS expr expr) {}
        | #(MINUS expr expr) {}
;
```

8.2 Static and Semantic Analyzer

```
/*
        Static Semantic Analyzer by Spencer Greenberg


            notes:
-note that with the append operation you might have  int a = myList[1] @ "hi"   and this
does not produce an error even though it should.
*/


void indent(int indent_level)           //adds the correct number of spaces in the trav_tree
function
{
  if (indent_level > 0) {
    const size_t BUFSIZE = 127;
    char buf[ BUFSIZE+1 ];
    int i;

    for (i = 0; i < indent_level && i < (int) BUFSIZE; i++) {
      buf[i] = ' ';
    }
    buf[i] = '\0';
    printf("%s", buf );
  }
} // pr_indent


void trav_tree(RefMyAST top, int indIn )     //walks through a tree and prints out its
structure
{
  if (top != NULL)
  {
        std::cout << top->getLine() << " ";
        std::string str;
        indent(indIn);

    str = top->getText();
    std::cout << str <<  "\n";
    if (top->getFirstChild() != NULL) {
```

```
      printf("kid: ");
      trav_tree( (RefMyAST) top->getFirstChild(), indIn+2 );
    }
    if (top->getNextSibling()) {
      printf("sib: ");
      trav_tree( (RefMyAST) top->getNextSibling(), indIn );
    }
  }
}


string SemanticErrorString = "";
//adds the line number, the node string, and the error message passed to our list of
semantic errors so far
void SemanticError(string error, RefMyAST node)
{
	SemanticErrorString += itos(node->getLine()) + ": '";
	SemanticErrorString += node->getText() + "' \t|\t";
	SemanticErrorString += error;
	SemanticErrorString += "\n";
}

string GetSemanticErrors(void)
{
	return SemanticErrorString;
}


//builds a static semantics node propertly for an instantiation such as  "int a" and "int[10]"
void BuildInstantiation(string child1text, SymbolTable *table, RefMyAST child1,
RefMyAST node, int type)
{
	node->evaltype(type);		//set the evaluation type of this node to the type
declared
	if(child1 != NULL) child1->evaltype(type);

	if(child1->getType() == Lex::ID)		//if we have an identifier
		{
			int pos = table->getLocal(child1text);  //look for it in the local
symbol table
			if(pos == -1)						//if it is not
already in the symbol table
			{
				Obj *loc = new Obj(type);
				loc->happy = "yay!";//debug
```

```
                                        table->add(loc, child1text);      //add it to the symbol table
(note that we won't actualy use this object.. we'll build a new one

//each time we encounter an instantiation during runtime



                                        node->setPosition(table->size() - 1, 0);   //tell the node
where its object can be found in the symbol table
                                        child1->setPosition(table->size() - 1, 0);

                                        //will have to actually allocate the memory for the object
itself each time the statement is encountered during runtime
                              }
                              else SemanticError("'" + child1text + "' is already declared in this
scope.", node);
                    }
          else if(child1->getType() == Lex::BRACKETEXPR)  //something of the form
int[10] (which would create a list of ten ints)
          {

          /*        child1 = child1->getFirstChild();
                    //make sure that they are creating an integer number of these!
                    if(child1->evaltype() != INTEGER && child1->evaltype() !=
UNKNOWN)
                    SemanticError("'" + node->getText() + "[" + numToType(child1-
>evaltype()) + "]' is undefined. Need " + node->getText() + "[integer]", node);
                    else
                    {
                              table->add(new Obj(LIST), "");
                              child1->setPosition(table->size() -1, 0);  //allocate memory for the
pointer. Will have to actually allocate the list itself
                                                                                    //each time
this statement is encountered during runtime
                    }*/
          }
          else SemanticError("Unknown instantiation error!", node);
}

//attach the appropriate symbol table to a given child node
void BuildChildSymbolTable(RefMyAST child1, SymbolTable *parentTable, string
child1text, RefMyAST curNode)
{
          child1->breakNode(curNode->breakNode());                    //in case we are
entering an 'while' scope, make sure the children know where to break to!
```

```
	//if the child will have the same scope as its parent
	if(child1text != "while" && child1text != "function" && child1text != "if" &&
child1text != "else")    child1->table(parentTable);             //give it its parents symbol
table.

	else                                    //otherwise give it a new symbol table
	{
		if(child1text == "while" || child1text == "if" || child1text == "else")
child1->table(new SymbolTable(parentTable, 1));  //since its a while loop/if/else it should
inheret the parent scope
		else child1->table(new SymbolTable(parentTable, 0));                   //since
its a function it should not inheret the parent scope (that's what the zero specifies)
	}
}

//performs static semantics check and restructuring
void MakeLoveToThisTree(RefMyAST node)
{
	string text = node->getText();
	string child1text= "", child2text = "";
	string sibtext = "";
	int t = node->getType();
	int c1 =201, c2=201;
	RefMyAST child1 = (RefMyAST) node->getFirstChild();
	RefMyAST child2 = NULL;

	if(node->table() == NULL) node->table(&globalScope);
	SymbolTable *table = node->table();

	if(text == "while") node->breakNode(node); //if this is a while loop then it breaks
to itself (so its children will break to it too)

	if(node->getNextSibling() != NULL) sibtext = node->getNextSibling()-
>getText();

	if(sibtext == "else")   //else statements can only follow if statements or other else
statements
	{
		if(text != "if" && text != "else") SemanticError("'if' statements can only
follow 'if' or 'else' statements. ", (RefMyAST) node->getNextSibling());
	}

	if(child1 != NULL)
	{
		child2 = (RefMyAST) child1->getNextSibling();
		child1text = child1->getText();
```

```
			if(t == Lex::DOT || text == "while" || text == "print" || text == "if" || text
== "else" || t == Lex::ELSE_IF || t == Lex::ASSIGN || t == Lex::RASSIGN || t ==
Lex::ARROW || text == "and" || text == "or" || t == Lex::LESS || t == Lex::GREATER || t
== Lex::LTE || t == Lex::GTE || t == Lex::COMPARE || t == Lex::RCOMPARE || t ==
Lex::APPEND || t== Lex::PLUS || t == Lex::MINUS || t== Lex::MULT || t == Lex::DIV
||  t == Lex::BRACKETEXPR || t== Lex::POUND)
			{
				BuildChildSymbolTable(child1, table, child1text, node);
				MakeLoveToThisTree(child1);

			}
			else if(t == Lex::ID && child1->getType() == Lex::BRACKETEXPR)
		//deal with the bracket expr case for identifiers
			{
					BuildChildSymbolTable(child1, table, child1->getText(),
node);
					RefMyAST temp1 = (RefMyAST) child1->getFirstChild();
					if(temp1 == NULL) SemanticError("Bracket expression
with no children!", node);
					else
					{
						BuildChildSymbolTable(temp1, table, temp1-
>getText(), child1);
						MakeLoveToThisTree(temp1);
						temp1 = (RefMyAST) temp1->getNextSibling();
						if(temp1 != NULL)
		//if it is of the form myList[3,5]
						{
							BuildChildSymbolTable(temp1, table,
temp1->getText(), node);
							MakeLoveToThisTree(temp1);
						}
					}
			}
			c1 = child1->evaltype();
		}
		if(child2 != NULL)
		{
			child2text = child2->getText();
			if(text == "while" || text == "if" || text == "else" || t == Lex::ASSIGN || t
== Lex::RASSIGN || t == Lex::ARROW || text == "and" || text == "or" || t == Lex::LESS
|| t == Lex::GREATER || t == Lex::LTE || t == Lex::GTE || t == Lex::COMPARE || t ==
Lex::RCOMPARE || t == Lex::APPEND || t== Lex::PLUS || t == Lex::MINUS || t==
Lex::MULT || t == Lex::DIV || t == Lex::BRACKETEXPR)
			{
```

```
			BuildChildSymbolTable(child2, table, child2text, node);
			MakeLoveToThisTree(child2);
		//recurse on the first child. In some cases this will have to be shut off! tofix
		}
		else if(t == Lex::DOT)  //e.g. res.resistance
		{
			//canhave expr.ID but not expr.expr
			if(child2->getType() != Lex::ID)
SemanticError("'expression.expression' is invalid. Can only have 'expression.identifier'",
node);
		}

		c2 = child2->evaltype();
	}


	if(t == Lex::AUTOLIST || t == Lex::PROGRAM || t==Lex::SEQ_OF_EXPRS)
	//autolists take the form {7, 5, "hi"} etc.
	{
		RefMyAST cur = child1;
		while(cur != NULL)			//build the symbol tables for all of
the children
		{
			BuildChildSymbolTable(cur, table, cur->getText(), node);
			cur = cur->getNextSibling();
		}

		cur = child1;
		while(cur != NULL)			//recurse on all of the children
		{
			MakeLoveToThisTree(cur);
			cur = cur->getNextSibling();
		}
	}

	//for debugging!
	//if(c1 == DEFAULT) SemanticError("Found a node of default evaluation type
after type should have been gained! text: " + child1text + "  parent text: " + text, node);
	//if(c2 == DEFAULT) SemanticError("Found a node of default evaluation type
after type should have been gained! text: " + child2text + "  parent text: " + text, node);

	//****************************************************************
***********
	//below we are doing all the stuff that needs to be done on the way up the tree!
	//****************************************************************
***********
```

```
	//if we have found a variable instantiation  (will be in one of the following forms: 
"type" or "type identifier" or "type[integer]")
	if(text == "int" || text == "float" || text == "string" || text == "list" || text == "vs" || 
text == "cs" || text == "ind" || text == "cap" || text == "res" || text == "node" || text == 
"diode" || text == "bjt" || text == "mosfet")
	{
		BuildInstantiation(child1text, table, child1, node, typeToNum(text));

		/*int pos, height;
		node->getPosition(pos, height);
		Obj *myOut = node->table()->getObj(pos,height);
		printf("\ngot: %s\n", myOut->happy.c_str());*/
	}

	//if we've found an identifier
	else if(t == Lex::ID)
	{
		int loc, height;
		table->get(text, loc, height);		//look this identifier up and find its 
symbol table and symbol table position

		if(loc == -1)	//if it isn't found
		{
			SemanticError("Undeclared identifier '" + text + "'", node);
	//if the identifier can't be found
			node->evaltype(UNDECLARED);
		}
		else
		{
			node->setPosition(loc, height);
			Obj *id = table->getObj(loc,height);
			node->evaltype(id->type);

			//Obj *test = table->getObj(node->getPosition());	//for debugging
			//printf("\nGoof ball! %s\n", test->happy.c_str());

			if(child1 != NULL && child1->getType() == 
Lex::BRACKETEXPR)			//if we have something like myList[5]
			{
				child1->setPosition(loc,height);

				RefMyAST temp1 = (RefMyAST) child1->getFirstChild(), 
temp2 = (RefMyAST) temp1->getNextSibling();
```

```
					if(id->type != LIST && id->type != STRING)
SemanticError("Can only use square brackets with lists and strings. " +
numToType(node->evaltype()) + "[] is undefined.", node);
					if(temp2 == NULL && temp1->evaltype() != INTEGER
&& temp1->evaltype() != UNKNOWN) SemanticError("Square brackets take integer
arguments. " + numToType(node->evaltype()) + "[" + numToType(temp1->evaltype()) +
"] is undefined.", node);
					else if(temp2 != NULL && ( (temp1->evaltype() !=
INTEGER && temp1->evaltype() != UNKNOWN) || (temp2->evaltype() != INTEGER
&& temp2->evaltype() != UNKNOWN) ) ) SemanticError("Square brackets take integer
arguments. " + numToType(node->evaltype()) + "[" + numToType(temp1->evaltype()) +
", " + numToType(temp2->evaltype()) + "] is undefined.", node);

					if(id->type == STRING) node->evaltype(STRING);
//when accessing a string will always return a string
					else node->evaltype(UNKNOWN);
	//if accessing a list we don't know what the type returned will be
			}
		}
	}

	//remember to include the '.' operator!

	else if(t == Lex::ASSIGN)
	{
		node->evaltype(child1->evaltype());
		if((c1 == INTEGER || c1 == FLOAT) && (c2 == INTEGER || c2 ==
FLOAT)) ;
		else if(child1->getType() == Lex::STRING_CONSTANT)
SemanticError("Cannot assign to a string constant. '" + child1text + " = " +
numToType(c2) + "' is undefined." , node);
		else if(child1->getType() == Lex::NUMBER) SemanticError("Cannot
assign to a numerical constant. '" + child1text + " = " + numToType(c2) + "' is
undefined." , node);
		else if(child1->getType() == Lex::AUTOLIST) SemanticError("Cannot
assign to a static list. '{...} = " + numToType(c2) + "' is undefined." , node);
		else if(c1 != c2 && c1 != UNKNOWN && c2 != UNKNOWN)
SemanticError("Type mismatch. '" + numToType(c1) + " = " + numToType(c2) + "'" ,
node);

		//else printf("\ntypes assigned: %s & %s\n", numToType(c1).c_str(),
numToType(c2).c_str()); //for debugging
	}
	else if(t == Lex::RASSIGN)
	{
		node->evaltype(child1->evaltype());
```

```
		if(child1->getType() == Lex::STRING_CONSTANT) 
SemanticError("Cannot assign to a string constant. '" + child1text + " &= " + 
numToType(c2) + "' is undefined." , node);
		else if(child1->getType() == Lex::NUMBER) SemanticError("Cannot 
assign to a numerical constant. '" + child1text + " &= " + numToType(c2) + "' is 
undefined." , node);
		else if(child1->getType() == Lex::AUTOLIST) SemanticError("Cannot 
assign to a static list. '{...} &= " + numToType(c2) + "' is undefined." , node);
		else if(c1 != c2 && c1 != UNKNOWN && c2 != UNKNOWN) 
SemanticError("Type mismatch. '" + numToType(c1) + " &= " + numToType(c2) + "'", 
node);
	}
	else if(text == "and" || text == "or")
	{
		node->evaltype(INTEGER);
		if((c1 != INTEGER && c1 != UNKNOWN)|| (c2 != INTEGER && c2 != 
UNKNOWN)) SemanticError("Type mismatch. the 'and' and 'or' operators require integer 
arguments" , node);
	}
	else if(t == Lex::LESS)
	{
		node->evaltype(INTEGER);
		Obj *ob = new Obj(INTEGER); table->add(ob, uniqueName());

		node->setPosition(table->size() - 1, 0);
		if(c1 != c2 && c1 != UNKNOWN && c2 != UNKNOWN) 
SemanticError("Type mismatch. '" + numToType(c1) + " < " + numToType(c2) + "'" , 
node);
	}
	else if(t == Lex::GREATER)
	{
		node->evaltype(INTEGER);
		Obj *ob = new Obj(INTEGER); table->add(ob, uniqueName());

		node->setPosition(table->size() - 1, 0);
		if(c1 != c2 && c1 != UNKNOWN && c2 != UNKNOWN) 
SemanticError("Type mismatch. '" + numToType(c1) + " > " + numToType(c2) + "'" , 
node);
	}
	else if(t == Lex::LTE)
	{
		node->evaltype(INTEGER);
		Obj *ob = new Obj(INTEGER); table->add(ob, uniqueName());

		node->setPosition(table->size() - 1, 0);
```

```
			if(c1 != c2 && c1 != UNKNOWN && c2 != UNKNOWN)
SemanticError("Type mismatch. '" + numToType(c1) + " <= " + numToType(c2) + "'" ,
node);
		}
		else if(t == Lex::GTE)
		{
			node->evaltype(INTEGER);
			Obj *ob = new Obj(INTEGER); table->add(ob, uniqueName());

			node->setPosition(table->size() - 1, 0);
			if(c1 != c2 && c1 != UNKNOWN && c2 != UNKNOWN)
SemanticError("Type mismatch. '" + numToType(c1) + " >= " + numToType(c2) + "'",
node);
		}
		else if(t == Lex::COMPARE)
		{
			node->evaltype(INTEGER);
			Obj *ob = new Obj(INTEGER); table->add(ob, uniqueName());

			node->setPosition(table->size() - 1, 0);
			if(c1 != c2 && c1 != UNKNOWN && c2 != UNKNOWN)
SemanticError("Type mismatch. '" + numToType(c1) + " == " + numToType(c2) + "'" ,
node);
		}
		else if(t == Lex::APPEND)
		{
			if(c1 != STRING && c1 != LIST && c1 != UNKNOWN)
			{
				node->evaltype(ERROR);
				SemanticError("Type mismatch. can only append to list and string
objects. '" + numToType(c1) + " @ " + numToType(c2) + "' is undefined." , node);
			}
			else
			{
				if(c1 != UNKNOWN) node->evaltype(c1);
				else if(c2 == LIST) node->evaltype(LIST);
				else node->evaltype(UNKNOWN);
			}
		}
		else if(t == Lex::PLUS || t == Lex::MINUS || t == Lex::MULT || t == Lex::DIV)
		{
			string op;
			if(t == Lex::PLUS) op = "+";
			else if(t == Lex::MINUS) op = "-";
			else if(t == Lex::MULT) op = "*";
			else if(t == Lex::DIV) op = "/";
```

```
		if( (c1 != INTEGER && c1 != FLOAT && c1 != UNKNOWN) || (c2 !=
INTEGER && c2 != FLOAT && c2 != UNKNOWN) )
		{
			node->evaltype(ERROR);
			SemanticError("Type mismatch. '" + numToType(c1) + " " + op +
" " + numToType(c2) + "'" , node);
		}
		else
		{
			if(c1 == UNKNOWN || c2 == UNKNOWN) node-
>evaltype(UNKNOWN);
			else if(c1 == INTEGER && c2 == INTEGER && t != Lex::DIV)
node->evaltype(INTEGER);
			else if(c1 == FLOAT || c2 == FLOAT) node->evaltype(FLOAT);
			else node->evaltype(FLOAT);
		}
	}
	else if(t == Lex::AUTOLIST)
	{
		node->evaltype(LIST);
		Obj *loc = new Obj(LIST);
		table->add(loc, uniqueName());
		node->setPosition(table->size() - 1, 0);   //tell the node where its object
can be found in the symbol table

	}
	else if(t == Lex::POUND)	//gets the size of a string or list
	{
		node->evaltype(INTEGER);
		if(c1 != STRING && c1 != LIST) SemanticError("Type mismatch. the
pound (#) operator can only be applied to strings and lists. '#" + numToType(c1) + "' is
undefined.", node);
	}
	//note: if we append a constant like this to a list we want to make a copy of it and
not copy it directly
	//the reason being that if the list has a scope that is more global than the object,
the object may have to
	//hang around but we don't want it to be modified by future passes through this
scope
	else if(t == Lex::NUMBER)
	{

		if(text.find(".") != string::npos) //if the string contains a decimal point
		{
			node->evaltype(FLOAT);
```

```
                    table->add(new Obj(FLOAT, atof(text.c_str())) ,  uniqueName());
//add a constant to the symbol table
                    node->setPosition(node->table()->size() - 1, 0);
                              //make this node know where its constant value is
stored
               }
               else
               {
                    node->evaltype(INTEGER);
                    table->add(new Obj(INTEGER, atoi(text.c_str())) ,
uniqueName());  //add a constant to the symbol table
                    node->setPosition(node->table()->size() - 1, 0);
                              //make this node know where its constant value is
stored
               }
          }
          //Like with numbers, we will need to duplicate string constant objects if we
append them to a list
          else if(t == Lex::STRING_CONSTANT)
          {
               node->evaltype(STRING);

               node->table()->add(new Obj(text) , uniqueName());
               //add a string constant to the symbol table
               node->setPosition(node->table()->size() - 1, 0);
                         //make this node know where its constant value is stored

               //printf("const: %s\n", node->table

          }
          else if(t == Lex::DOT)  //e.g. res.resistance
          {
               if(c1 == STRING || c1 == INTEGER || c1 == FLOAT || c1 == LIST || c1
== ERROR || c1 == DEFAULT || c1 == NODE || c1 == UNDEFINED)
               {
                    SemanticError("Can only apply the dot (.) operator to circuit
objects. '" + numToType(c1) + "." + child2text +"' is undefined.", node);
                    node->evaltype(ERROR);
               }
               else if( (child2text == "pos" || child2text == "neg") && (c1 == RES || c1
== CAP || c1 == IND || c1 == VS || c1 == CS || c1 == DIODE) )
               {
                    node->evaltype(NODE);
                    Obj *ob = new Obj(NODE);
                    table->add(ob, uniqueName());
```

```
			node->setPosition(table->size() - 1, 0);
		}
		else if( (child2text == "argument") && (c1 == RES || c1 == CAP || c1 ==
IND || c1 == VS || c1 == CS || c1 == DIODE || c1 == BJT || c1 == MOSFET) )
		{
			node->evaltype(STRING);
			Obj *ob = new Obj(STRING);
			table->add(ob, uniqueName());

			node->setPosition(table->size() - 1, 0);
		}
		else if(c1 == RES)
		{
			if(child2text == "resistance")
			{
				node->evaltype(FLOAT); Obj *ob = new Obj(FLOAT);
				table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
			}
			else
			{
				SemanticError("Resistor objects have no element '." +
child2text + "'", node);
				node->evaltype(ERROR);
			}
		}
		else if(c1 == CAP)
		{
			if(child2text == "capacitance" || child2text == "initial_voltage")
			{
				node->evaltype(FLOAT); Obj *ob = new Obj(FLOAT);
				table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
			}
			else
			{
				SemanticError("Capacitor objects have no element '." +
child2text + "'", node);
				node->evaltype(ERROR);
			}
		}
		else if(c1 == IND)
		{
			if(child2text == "inductance" || child2text == "initial_current")
			{
				node->evaltype(FLOAT); Obj *ob = new Obj(FLOAT);
```

```
					table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
				}
				else
				{
					SemanticError("Inductor objects have no element '." +
child2text + "'", node);
					node->evaltype(ERROR);
				}
			}
			else if(c1 == VS)
			{
				if(child2text == "voltage")
				{
					node->evaltype(FLOAT); Obj *ob = new Obj(FLOAT);
					table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
				}
				else
				{
					SemanticError("Voltage source objects have no element '."
+ child2text + "'", node);
					node->evaltype(ERROR);
				}
			}
			else if(c1 == CS)
			{
				if(child2text == "current")
				{
					node->evaltype(FLOAT); Obj *ob = new Obj(FLOAT);
					table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
				}
				else
				{
					SemanticError("Current source objects have no element '."
+ child2text + "'", node);
					node->evaltype(ERROR);
				}
			}
			else if(c1 == DIODE)
			{
				SemanticError("Diode objects have no element '." + child2text +
"'", node);
				node->evaltype(ERROR);
			}
```

```
		else if(c1 == BJT)
		{
			if(child2text == "base" || child2text == "collector" || child2text ==
"emitter")
			{
				node->evaltype(NODE); Obj *ob = new Obj(NODE);
				table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
			}
			else
			{
				SemanticError("BJT transistor objects have no element '." +
child2text + "'", node);
				node->evaltype(ERROR);
			}
		}
		else if(c1 == MOSFET)
		{
			if(child2text == "gate" || child2text == "source" || child2text ==
"drain")
			{
				node->evaltype(NODE); Obj *ob = new Obj(NODE);
				table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
			}
			else
			{
				SemanticError("MOSFET transistor objects have no
element '." + child2text + "'", node);
				node->evaltype(ERROR);
			}
		}
		else if(c1 == UNKNOWN)
		{
			string w = child2text;
			if(w == "gate" || w == "source" || w == "drain" || w == "base" || w
== "collector" || w == "emitter" || w == "pos" || w == "neg")
			{
				node->evaltype(NODE); Obj *ob = new Obj(NODE);
				table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
			}
			else if(w == "argument")
			{
				node->evaltype(STRING); Obj *ob = new Obj(STRING);
```

```
				table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
			}
			else if(w == "resistance" || w == "capacitance" || w == "inductance"
|| w == "current" ||  w == "voltage" || w == "initial_voltage" || w == "initial_current")
			{
				node->evaltype(FLOAT); Obj *ob = new Obj(FLOAT);
				table->add(ob, uniqueName());		node-
>setPosition(table->size() - 1, 0);
			 }
		}

		//if(child1->evaltype() == DEFAULT) child1->evaltype(node-
>evaltype());	//make sure the children of dot aren't left out in the ghetto
		//if(child2->evaltype() == DEFAULT) child2->evaltype(node-
>evaltype());
	}
	else if(t == Lex::ARROW)
	{
		if( (c1 != NODE && c1 != UNKNOWN) || (c2 != NODE && c2 !=
UNKNOWN) )
		{
			SemanticError("Can only bind node objects to other node objects.
'" + numToType(c1) + "->" + numToType(c2) + "' is undefined.", node);
			node->evaltype(ERROR);
		}
		else node->evaltype(NODE);
	}
	else if(text == "break")
	{
		if(node->breakNode() == NULL) SemanticError("The 'break' keyword
can only be used inside 'while' loops.", node);
		 node->evaltype(UNDEFINED);
	}
	else if(text == "continue")
	{
		if(node->breakNode() == NULL) SemanticError("The 'continue' keyword
can only be used inside 'while' loops.", node);
		node->evaltype(UNDEFINED);
	}
	else if(t == Lex::SPICE)  //this is where we get the text of spice injections. True
comments will already have been obliterated
	{
		//printf("\nInjection: \n%s\n", child1text.c_str());	/*this line
demonstrates how to get the spice injection text! Its just in child1test*/
		node->evaltype(UNDEFINED);
```

```
	}
	else if(text == "ground")  //if it is the ground node
	{
		node->evaltype(NODE);
	}

}
```

8.3 Interpreter

```
/*
	Interpreter by Rob Toth


  */
void ObjAssign(Obj *left, Obj *right, int lineNumber);

void RuntimeError(string in, int lineNumber)		//kills the program after printing the
passed string to stdout
{
   printf("\nline %d| Runtime Error! %s\n", lineNumber, in.c_str());
   exit(0);	//assert(1);
}

int isSMLType(int type)
{
	if(type == RES || type == CAP || type == IND || type == VS || type ==  CS || type
== DIODE || type == BJT || type == MOSFET) return 1;
	return 0;
}

void printNodeList(SMLNode *in)
{
	int count = in->count();
	for(int i = 0; i < count; i++)
	{
		printf(" %p ", (*in)[i]);
	}
}

void printNodes(Obj *in)
{
	int t = in->type;
	if(isSMLType(t) == 0) return;
```

```
	SMLData *smlD = (SMLData*) (in->data);

	printf(" pos[ %p|", smlD->pos); printNodeList(smlD->pos); printf("] ");
	printf(" neg[ %p|", smlD->neg); printNodeList(smlD->neg); printf("] ");

	if(t == BJT)
	{
		Bjt *bjt = (Bjt*) (in->data);

		printf(" base[ %p|", bjt->base); printNodeList(bjt->base); printf("] ");
		printf(" emit[ %p|", bjt->emitter); printNodeList(bjt->emitter); printf("] ");
		printf(" collect[ %p|", bjt->collector); printNodeList(bjt->collector);
printf("] ");
	}
	else if(t == MOSFET)
	{
		Mosfet *mosfet = (Mosfet*) (in->data);

		printf(" gate[ %p|", mosfet->gate); printNodeList(mosfet->gate); printf("]
");
		printf(" source[ %p|", mosfet->source); printNodeList(mosfet->source);
printf("] ");
		printf(" drain[ %p|", mosfet->drain); printNodeList(mosfet->drain);
printf("] ");
	}
}

int posInVector(Obj *toFind, ObjList *vec)
{
	int count = vec->size();
	for(int i = 0; i < count; i++)
	{
		if((*vec)[i] == toFind) return i;
	}
	return -1;
}

//makes the left SML object have the same relative node connections in lvec has right has
in rvec.
void MakeRelativeConnections(ObjList *lvec, ObjList *rvec, SMLData *left, SMLData
*right, int type)
{
	Obj *curParent;
	int loc;
	SMLNode *curNode;
```

```
SMLNode *newNode;
int nodeCount;

for(int k = 0; k <= 4; k++)
{

	if(k > 1 && type != BJT && type != MOSFET) break;

	if(k == 0)
	{
		curNode = right->pos;
		newNode = left->pos;
	}
	else if(k == 1)
	{
		curNode = right->neg;
		newNode = left->neg;
	}
	else if(k == 2)
	{
		if(type == BJT)
		{
			curNode = ((Bjt *) right)->base;
			newNode = ((Bjt *) left)->base;
		}
		else if(type == MOSFET)
		{
			curNode = ((Mosfet *) right)->gate;
			newNode = ((Mosfet *) left)->gate;
		}
	}
	else if(k == 3)
	{
		if(type == BJT)
		{
			curNode = ((Bjt *) right)->collector;
			newNode = ((Bjt *) left)->collector;
		}
		else if(type == MOSFET)
		{
			curNode = ((Mosfet *) right)->source;
			newNode = ((Mosfet *) left)->source;
		}
	}
	else if(k == 4)
	{
```

```
			if(type == BJT)
			{
				curNode = ((Bjt *) right)->emitter;
				newNode = ((Bjt *) left)->emitter;
			}
			else if(type == MOSFET)
			{
				curNode = ((Mosfet *) right)->drain;
				newNode = ((Mosfet *) left)->drain;
			}
		}

		nodeCount = curNode->count();

		for(int i = 0; i < nodeCount; i++)			//for each node in the current
nodes connection list
		{
			curParent = ((*curNode)[i])->parent;
			loc = posInVector(curParent, rvec); //if this node connects to the
node of an object in the list
			if(loc != -1)
			{
				if((*curNode)[i] == ((SMLData*)(curParent->data))->pos)
newNode->connect((((SMLData *)((*lvec)[loc]->data))->pos);			//if it
connects to the pos field of this object
				else if((*curNode)[i] == ((SMLData*)(curParent->data))-
>neg ) newNode->connect((((SMLData *)((*lvec)[loc]->data))->neg);		//if it
connects to the neg field of this object
				else if(type == BJT)
				{
					Bjt *bjtParent = (Bjt *) curParent->data;
					if((*curNode)[i] == bjtParent->base) newNode-
>connect((((Bjt *)((*lvec)[loc]->data))->base);
					else if((*curNode)[i] == bjtParent->emitter)
newNode->connect((((Bjt *)((*lvec)[loc]->data))->emitter);
					else if((*curNode)[i] == bjtParent->collector)
newNode->connect((((Bjt *)((*lvec)[loc]->data))->collector);
				}
				else if(type == MOSFET)
				{
					Mosfet *mosfetParent = (Mosfet *) curParent-
>data;
					if((*curNode)[i] == mosfetParent->gate) newNode-
>connect((((Mosfet *)((*lvec)[loc]->data))->gate);
					else if((*curNode)[i] == mosfetParent->source)
newNode->connect((((Mosfet *)((*lvec)[loc]->data))->source);
```

```
					else if((*curNode)[i] == mosfetParent->drain)
newNode->connect(((Mosfet *)((*lvec)[loc]->data))->drain);
				}
			}
		}

	}


}

//assigns one list object to another duplicating the objects within and retaining the relative
structure of the SML objects duplicated
void ListAssign(Obj* left, Obj *right, int lineNumber)
{
		ObjList *lvec = (ObjList*) left->data;
		ObjList *rvec = (ObjList*) right->data;

		Obj *curObj;
		Obj *newObj;
		int t;			//cur obj type

		lvec->clear();
		int count = rvec->size();
		for(int i = 0; i < count; i++)
		{
			curObj = (*rvec)[i];
			t = curObj->type;

			if(t == INTEGER) newObj = new Obj(INTEGER);
			else if(t == FLOAT) newObj = new Obj(FLOAT);
			else if(t == STRING) newObj = new Obj(STRING);
			else if(t == LIST) newObj = new Obj(LIST);
			else if(t == RES) newObj = new Obj(RES);
			else if(t == CAP) newObj = new Obj(CAP);
			else if(t == IND) newObj = new Obj(IND);
			else if(t == VS) newObj = new Obj(VS);
			else if(t == CS) newObj = new Obj(CS);
			else if(t == DIODE) newObj = new Obj(DIODE);
			else if(t == BJT) newObj = new Obj(BJT);
			else if(t == MOSFET) newObj = new Obj(MOSFET);
			else RuntimeError("Tried to assign an object of an unknown type.
'object type " + itos(t) + " = object type " + itos(t) + "' is undefined.", lineNumber);

			if(t != LIST) ObjAssign(newObj, curObj, lineNumber);
			else newObj->data = curObj->data;
```

```
			lvec->push_back(newObj);
		}

		for(i = 0; i < count; i++)
		{
			curObj = (*rvec)[i];
			newObj = (*lvec)[i];
			t = curObj->type;
			if(isSMLType(t)) MakeRelativeConnections(lvec, rvec, (SMLData
*)(newObj->data), (SMLData *)(curObj->data), t);
		}
}


//assigns the data of the left object to the data of the right object
void ObjAssign(Obj *left, Obj *right, int lineNumber)
{
	int tl = left->type, tr = right->type;
	void *dl = left->data; void *dr = right->data;

	if( isSMLType(tl) && isSMLType(tr))
	{
		SMLData *l = (SMLData*) dl, *r = (SMLData*) dr;
		l->argument = r->argument;
	}

	if(tl == INTEGER && tr == FLOAT) *((long*)dl) = (long) *((double*)dr);
	else if(tl == FLOAT && tr == INTEGER) *((double*)dl) = (double)
*((long*)dr);
	else if(left->type != right ->type) RuntimeError("Tried to assign two objects of
incompatible type. '" + numToType(tl) + " = " + numToType(tr) + "' is undefined.",
lineNumber);
	else if(tl == INTEGER) *((long*)dl) = *((long*)dr);
	else if(tl == FLOAT)
	{
		//float test = *((double*)dr);   //for debugging
		*((double*)dl) = *((double*)dr);
		//test = *((double*)dl);                    //for debugging
		//test = test-1;
	}
	else if(tl == STRING) *((string*)dl) = *((string*)dr);
	else if(tl == LIST) ListAssign(left, right, lineNumber);
	else if(tl == RES)
	{
		Res *l = (Res*) dl, *r = (Res*) dr;
```

```
			l->resistance = r->resistance;
		}
		else if(tl == CAP)
		{
			Cap *l = (Cap*) dl, *r = (Cap*) dr;
			l->capacitance = r->capacitance;
			l->initial_voltage = r->initial_voltage;
		}
		else if(tl == IND)
		{
			Ind *l = (Ind*) dl, *r = (Ind*) dr;
			l->inductance = r->inductance;
			l->initial_current = r->initial_current;
		}
		else if(tl == VS)
		{
			Vs *l = (Vs*) dl, *r = (Vs*) dr;
			l->voltage = r->voltage;
		}
		else if(tl == CS)
		{
			Cs *l = (Cs*) dl, *r = (Cs*) dr;
			l->current = r->current;
		}
		else if(tl == DIODE)
		{

		}
		else if(tl == BJT)
		{

		}
		else if(tl == MOSFET)
		{

		}
}



void FinishOffThisTree(RefMyAST node)
{
		string text = node->getText();
		string child1text= "", child2text = "";
		string sibtext = "";
		int t = node->getType();
```

```
	int c1 =201, c2=201;
	RefMyAST child1 = (RefMyAST) node->getFirstChild();
	RefMyAST child2 = NULL;

	if(node->table() == NULL) node->table(&globalScope);
	SymbolTable *table = node->table();


	//not sure if this is needed in this case
	//come back and check it out later
	if(text == "if" || text == "else" || text == "while") node->breakNode(node); //if this
is an if or else node then it breaks to itself (so its children will break to it too)

	if(node->getNextSibling() != NULL) sibtext = node->getNextSibling()-
>getText();

	if(sibtext == "else")   //else statements can only follow if statements or other else
statements
	{
	//	if(text != "if" && text != "else") SemanticError("'if' statements can only
follow 'if' or 'else' statements. ", (RefMyAST) node->getNextSibling());
	}



	if(child1 != NULL)
	{
		child2 = (RefMyAST) child1->getNextSibling();
		child1text = child1->getText();

		//find all operators that will have a smaller scope and build a symbol table
specific to them
		if(t == Lex::DOT || text == "while" || text == "if" || text == "print" || text
== "else" || t == Lex::ELSE_IF || t == Lex::ASSIGN || t == Lex::RASSIGN || t ==
Lex::ARROW || text == "and" || text == "or" || t == Lex::LESS || t == Lex::GREATER || t
== Lex::LTE || t == Lex::GTE || t == Lex::COMPARE || t == Lex::RCOMPARE || t ==
Lex::APPEND || t== Lex::PLUS || t == Lex::MINUS || t== Lex::MULT || t == Lex::DIV
||  t == Lex::BRACKETEXPR || t== Lex::POUND)
		{
			if (text == "if"   ){
				int posNode, heightNode;
				FinishOffThisTree(child1);
				child1->getPosition(posNode, heightNode);
				Obj *objNode = child1->table()-
>getObj(posNode,heightNode);
				if ( *((long*)objNode->data) == 0 ){
```

```
					child2->setText("FALSE");
					if ( node->getNextSibling() != NULL && node-
>getNextSibling()->getText() == "else") {
						RefMyAST elseNode = (RefMyAST) node-
>getNextSibling();
						RefMyAST elseChild = (RefMyAST)
elseNode->getFirstChild();
						elseChild->setText("seq_of_exprs");
					}
				}else if ( node->getNextSibling() != NULL && node-
>getNextSibling()->getText() == "else" ){
					RefMyAST elseNode = (RefMyAST) node-
>getNextSibling();
					RefMyAST elseChild = (RefMyAST) elseNode-
>getFirstChild();
					elseChild->setText("seq_of_exprs");
					child2->setText("seq_of_exprs");
				}else child2->setText("seq_of_exprs");

			}else if ( text == "while" ){
				int posNode, heightNode;
				child1->getPosition(posNode, heightNode);
				FinishOffThisTree(child1);
				Obj *objNode = child1->table()->getObj(posNode,
heightNode);
				while ( *((long*)objNode->data) != 0 ){
					FinishOffThisTree(child1);
					objNode = child1->table()->getObj(posNode,
heightNode);
					if ( *((long*)objNode->data) != 0 ){
						FinishOffThisTree(child2);
						child2->setText("seq_of_exprs");
					}
					else child2->setText("FALSE");
				}
				if ( *((long*)objNode->data) == 0) child2-
>setText("FALSE");
			}
			FinishOffThisTree(child1);
		}

		if(t == Lex::ID && child1->getType() == Lex::BRACKETEXPR)
	//deal with the bracket expr case for identifiers
		{
```

```
				FinishOffThisTree((RefMyAST)child1->getFirstChild());
				RefMyAST child3 = (RefMyAST) child1->getFirstChild();
				int pos, height, pos2, height2;
				child1->getPosition(pos, height);
				Obj *objNode = child1->table()->getObj(pos, height);
				child3->getPosition(pos2, height2);
				Obj *objChild3 = child3->table()->getObj(pos2, height2);
				ObjList *listNode =(ObjList*)objNode->data;

				unsigned int element = *((long*)objChild3->data);
				if(element <= 0) RuntimeError("Tried to access element " +
itos(element) + " of a list.", node->getLine());
				if(element > listNode->size()) RuntimeError("Tried to access
element " + itos(element) + " of a list of length " + itos(listNode->size()) + ".", node-
>getLine());
				Obj *listObj = (*listNode)[element-1];

				node->table()->add(listObj, uniqueName());
				node->setPosition(node->table()->size() - 1, 0 );
			}
			c1 = child1->evaltype();
	}

	//same thing for child2
	if(child2 != NULL)
	{
		child2text = child2->getText();
		if(text == "while" || text == "if" || text == "else" || t == Lex::ASSIGN || t
== Lex::RASSIGN || t == Lex::ARROW || text == "and" || text == "or" || t == Lex::LESS
|| t == Lex::GREATER || t == Lex::LTE || t == Lex::GTE || t == Lex::COMPARE || t ==
Lex::RCOMPARE || t == Lex::APPEND || t== Lex::PLUS || t == Lex::MINUS || t==
Lex::MULT || t == Lex::DIV || t == Lex::BRACKETEXPR)
		{
			if (text == "if"){
				int posNode, heightNode;
				FinishOffThisTree(child1);
				child1->getPosition(posNode, heightNode);
				Obj *objNode = child1->table()-
>getObj(posNode,heightNode);
				if ( *((long*)objNode->data) == 0 ){
					printf("if statement failed yay!");
					child2->setText("FALSE");
					if ( node->getNextSibling() != NULL && node-
>getNextSibling()->getText() == "else") {
						RefMyAST elseNode = (RefMyAST) node-
>getNextSibling();
```

```
                                        RefMyAST elseChild = (RefMyAST)
elseNode->getFirstChild();
                                        elseChild->setText("seq_of_exprs");
                                }
                        }else if ( node->getNextSibling() != NULL && node-
>getNextSibling()->getText() == "else" ){
                                RefMyAST elseNode = (RefMyAST) node-
>getNextSibling();
                                RefMyAST elseChild = (RefMyAST) elseNode-
>getFirstChild();
                                elseChild->setText("seq_of_exprs");
                                child2->setText("seq_of_exprs");
                        }else{
                                child2->setText("seq_of_exprs");
                        }

                }else if ( text == "while"  ){
        //                      if ( c1 != -55 ) FinishOffThisTree(child2);
                }

                FinishOffThisTree(child2);
    //recurse on the first child. In some cases this will have to be shut off! tofix

        }
        else if(t == Lex::DOT)  //e.g. res.resistance
        {
                //canhave expr.ID but not expr.expr
        //      if(child2->getType() != Lex::ID){}
//SemanticError("'expression.expression' is invalid. Can only have 'expression.identifier'",
node);
        }

        c2 = child2->evaltype();
    }

    //recurses all the way down
    if(t == Lex::AUTOLIST || t == Lex::PROGRAM || t==Lex::SEQ_OF_EXPRS)
    //autolists take the form {7, 5, "hi"} etc.
    {
        RefMyAST cur = child1;
    //  while(cur != NULL)                      //build the symbol tables for all of
the children
    //  {
    //          BuildChildSymbolTable(cur, table, cur->getText(), node);
    //table is already built simply go all the way down
    //          cur = cur->getNextSibling();
```

```
//		}

		if ( t == Lex::PROGRAM ){
			cur = child1;

			while(cur != NULL)			//recurse on all of the
children
			{
				FinishOffThisTree(cur);
				cur = cur->getNextSibling();
			}

		}
		else if ( t == Lex::SEQ_OF_EXPRS && text != "FALSE")
		{
			cur = child1;

			while(cur != NULL)			//recurse on all of the
children
			{
				FinishOffThisTree(cur);
				cur = cur->getNextSibling();
			}
		}
		else  if( t == Lex::AUTOLIST )
		{
			cur = child1;
			while(cur != NULL)			//recurse on all of the
children
			{
				FinishOffThisTree(cur);
				cur = cur->getNextSibling();
			}
		}
	}

		//if we have found a variable instantiation  (will be in one of the following
forms: "type" or "type identifier" or "type[integer]")
	if(text == "int" || text == "float" || text == "string" || text == "list" || text == "vs" ||
text == "cs" || text == "ind" || text == "cap" || text == "res" || text == "node")
	{
		int pos, height;
		node->getPosition(pos, height);
		Obj *newObj = new Obj(typeToNum(text));
		newObj->happy = "my friend";
		node->table()->setObj(newObj, pos, height);
```

```
		//node->table()->get(child1->getText(), pos, height);
		//Obj *res = node->table()->getObj(pos, height);

	}

	//if we've found an identifier
	else if(t == Lex::ID)
	{
	}


	else if(t == Lex::ASSIGN)
	{
		int posChild1, posChild2, heightChild1, heightChild2;
		child1->getPosition(posChild1, heightChild1);
		child2->getPosition(posChild2, heightChild2);
		Obj *objChild1 = child1->table()->getObj(posChild1,heightChild1);
		Obj *objChild2 = child2->table()->getObj(posChild2,heightChild2);

		ObjAssign(objChild1, objChild2, node->getLine()); //this now handles the
assignment for all object types

		/*
		if(objChild1->type == INTEGER && objChild2->type == FLOAT)
*((long*)objChild1->data) = (long) *((double*)objChild2->data);
		else if(objChild1->type == FLOAT && objChild2->type == INTEGER)
*((double*)objChild1->data) = (double) *((long*)objChild2->data);
		else if(objChild1->type == INTEGER && objChild2->type ==
INTEGER) *((long*)objChild1->data) = *((long*)objChild2->data);
		else if(objChild1->type == FLOAT && objChild2->type == FLOAT)
*((double*)objChild1->data) = *((double*)objChild2->data);
		else if (objChild1->type == STRING && objChild2->type == STRING)
*((string*)objChild1->data) = *((string*)objChild2->data);
		else objChild1->data = objChild2->data;*/

	}
	else if(text == "and" || text == "or")
	{
		printf ( "iosndf ");
	}
	else if(t == Lex::LESS)
	{
		int posChild1, posChild2, heightChild1, heightChild2;
		child1->getPosition(posChild1, heightChild1);
		child2->getPosition(posChild2, heightChild2);
```

```
		Obj *objChild1 = child1->table()->getObj(posChild1, heightChild1);
		Obj *objChild2 = child2->table()->getObj(posChild2, heightChild2);
		int posNode, heightNode;
		node->getPosition(posNode, heightNode);
		Obj *objNode = node->table()->getObj(posNode, heightNode);
		int eval = child1->evaltype();
		if (eval == INTEGER ){
			if ( *((long*)objChild1->data) >= *((long*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;
		}else if (eval == FLOAT ){
			if ( *((double*)objChild1->data) >= *((double*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;
		}
	}
	else if(t == Lex::GREATER)
	{

		int posChild1, posChild2, heightChild1, heightChild2;
		child1->getPosition(posChild1, heightChild1);
		child2->getPosition(posChild2, heightChild2);
		Obj *objChild1 = child1->table()->getObj(posChild1, heightChild1);
		Obj *objChild2 = child2->table()->getObj(posChild2, heightChild2);
		int posNode, heightNode;
		node->getPosition(posNode, heightNode);
		Obj *objNode = node->table()->getObj(posNode, heightNode);
		int eval = child1->evaltype();
		if (eval == INTEGER ){
			if ( *((long*)objChild1->data) <= *((long*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;
		}else if (eval == FLOAT ){
			if ( *((double*)objChild1->data) <= *((double*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;
		}

	}
	else if(t == Lex::LTE)
	{

		int posChild1, posChild2, heightChild1, heightChild2;
		child1->getPosition(posChild1, heightChild1);
		child2->getPosition(posChild2, heightChild2);
		Obj *objChild1 = child1->table()->getObj(posChild1, heightChild1);
```

```
		Obj *objChild2 = child2->table()->getObj(posChild2, heightChild2);
		int posNode, heightNode;
		node->getPosition(posNode, heightNode);
		Obj *objNode = node->table()->getObj(posNode, heightNode);
		int eval = child1->evaltype();
		if (eval == INTEGER ){
			if ( *((long*)objChild1->data) > *((long*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;
		}else if (eval == FLOAT ){
			if ( *((double*)objChild1->data) > *((double*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;

		}

	}
	else if(t == Lex::GTE)
	{

		int posChild1, posChild2, heightChild1, heightChild2;
		child1->getPosition(posChild1, heightChild1);
		child2->getPosition(posChild2, heightChild2);
		Obj *objChild1 = child1->table()->getObj(posChild1, heightChild1);
		Obj *objChild2 = child2->table()->getObj(posChild2, heightChild2);
		int posNode, heightNode;
		node->getPosition(posNode, heightNode);
		Obj *objNode = node->table()->getObj(posNode, heightNode);
		int eval = child1->evaltype();
		if (eval == INTEGER ){
			if ( *((long*)objChild1->data) < *((long*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;
		}else if (eval == FLOAT ){
			if ( *((double*)objChild1->data) < *((double*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;
		}
	}
	else if(t == Lex::COMPARE)
	{

		int posChild1, posChild2, heightChild1, heightChild2;
		child1->getPosition(posChild1, heightChild1);
		child2->getPosition(posChild2, heightChild2);
```

```
		Obj *objChild1 = child1->table()->getObj(posChild1, heightChild1);
		Obj *objChild2 = child2->table()->getObj(posChild2, heightChild2);

		int posNode, heightNode;
		node->getPosition(posNode, heightNode);
		Obj *objNode = node->table()->getObj(posNode, heightNode);

		int eval = child1->evaltype();

		if (eval == INTEGER ){
			if ( *((long*)objChild1->data) != *((long*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;
		}else if (eval == FLOAT ){
			if ( *((double*)objChild1->data) != *((double*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;

		}else if (eval ==STRING){
			if ( *((string*)objChild1->data) != *((string*)objChild2->data) )
*((long*)objNode->data) = 0;
			else *((long*)objNode->data) = 1;
		}
	}
	else if(t == Lex::APPEND)
	{

		int eval = child1->evaltype();
		int posChild1, posChild2, heightChild1, heightChild2;
		child1->getPosition(posChild1, heightChild1);
		child2->getPosition(posChild2, heightChild2);
		Obj *objChild1 = child1->table()->getObj(posChild1, heightChild1);
		Obj *objChild2 = child2->table()->getObj(posChild2, heightChild2);
		if ( eval == STRING ) {
			string ch1, ch2;
			ch1 = *((string*)objChild1->data);
			ch2 = *((string*)objChild2->data);
			ch1.append(ch2);
			*((string*)objChild1->data) = ch1;
		}else if ( eval == LIST){
			ObjList *listChild1 = (ObjList*)objChild1->data;
			listChild1->push_back(objChild2);

			//vector<Obj *>::iterator theIterator;
		/*	for( theIterator = listChild1->begin(); theIterator != listChild1-
>end(); theIterator++ ){
```

```
				Obj *test = *theIterator;
				printf(" printning it %d ", *((long*)test->data));
			}*/

		}


	}
	else if(t == Lex::PLUS || t == Lex::MINUS || t == Lex::MULT || t == Lex::DIV)
	{
		int posChild1, heightChild1, posChild2, heightChild2;
		child1->getPosition(posChild1, heightChild1);
		child2->getPosition(posChild2, heightChild2);
		Obj *objChild1 = child1->table()->getObj(posChild1, heightChild1);
		Obj *objChild2 = child2->table()->getObj(posChild2, heightChild2);
		int e1 = child1->evaltype(), e2 = child2->evaltype();
		if (e1 == INTEGER && e2 == INTEGER)
		{
			if (t==Lex::PLUS) table->add(new Obj(INTEGER,
(*((long*)objChild1->data))+(*((long*)objChild2->data))),uniqueName());
			else if (t==Lex::MINUS) table->add(new Obj(INTEGER,
(*((long*)objChild1->data))-(*((long*)objChild2->data))),uniqueName());
			else if (t==Lex::MULT) table->add(new Obj(INTEGER,
(*((long*)objChild1->data))*(*((long*)objChild2->data))),uniqueName());
			else  table->add(new Obj(INTEGER, (*((long*)objChild1-
>data))/(*((long*)objChild2->data))),uniqueName());
		}
		else if (e1 == FLOAT && e2 == FLOAT)
		{
			if (t==Lex::PLUS) table->add(new Obj(FLOAT,
(*((double*)objChild1->data))+(*((double*)objChild2->data))),uniqueName());
			else if (t==Lex::MINUS) table->add(new Obj(FLOAT,
(*((double*)objChild1->data))-(*((double*)objChild2->data))),uniqueName());
			else if (t==Lex::MULT) table->add(new Obj(FLOAT,
(*((double*)objChild1->data))*(*((double*)objChild2->data))),uniqueName());
			else  table->add(new Obj(FLOAT, (*((double*)objChild1-
>data))/(*((double*)objChild2->data))),uniqueName());
		}
		else if (e1 == FLOAT && e2 == INTEGER)
		{
			if (t==Lex::PLUS) table->add(new Obj(FLOAT,
(*((double*)objChild1->data))+(*((long*)objChild2->data))),uniqueName());
			else if (t==Lex::MINUS) table->add(new Obj(FLOAT,
(*((double*)objChild1->data))-(*((long*)objChild2->data))),uniqueName());
			else if (t==Lex::MULT) table->add(new Obj(FLOAT,
(*((double*)objChild1->data))*(*((long*)objChild2->data))),uniqueName());
```

```
				else  table->add(new Obj(FLOAT, (*((double*)objChild1-
>data))/(*((long*)objChild2->data))),uniqueName());
			}
			else if (e1 == INTEGER && e2 == FLOAT)
			{
				if (t==Lex::PLUS) table->add(new Obj(FLOAT,
(*((long*)objChild1->data))+(*((double*)objChild2->data))),uniqueName());
				else if (t==Lex::MINUS) table->add(new Obj(FLOAT,
(*((long*)objChild1->data))-(*((double*)objChild2->data))),uniqueName());
				else if (t==Lex::MULT) table->add(new Obj(FLOAT,
(*((long*)objChild1->data))*(*((double*)objChild2->data))),uniqueName());
				else  table->add(new Obj(FLOAT, (*((long*)objChild1-
>data))/(*((double*)objChild2->data))),uniqueName());
			}
			else RuntimeError("Attempted to apply an arithmetic operation to non-
arithmetic types " + numToType(e1) + " and " + numToType(e2) + ".", node->getLine());

			node->setPosition(node->table()->size() - 1, 0);
		}
		else if(t == Lex::AUTOLIST)
		{
			int posNode, heightNode;
			node->getPosition(posNode, heightNode);
			Obj *objNode = node->table()->getObj(posNode, heightNode);
			ObjList *listNode = (ObjList*)objNode->data;

			if ( child1 != NULL )
			{
				int posChild1, heightChild1;
				child1->getPosition(posChild1, heightChild1);
				Obj *objChild1 = child1->table()->getObj(posChild1,
heightChild1);
				listNode->push_back(objChild1);
				if (child2 != NULL )
				{
					int posChild2, heightChild2;
					child2->getPosition(posChild2, heightChild2);
					Obj *objChild2 = child2->table()->getObj(posChild2,
heightChild2);
					listNode->push_back(objChild2);
					RefMyAST childNextSibling = (RefMyAST) child2-
>getNextSibling();
					while (childNextSibling != NULL)
					{
						int posChildNext, heightChildNext;
```

```
                                        childNextSibling->getPosition(posChildNext,
heightChildNext);
                                        Obj *objChildNext = childNextSibling->table()-
>getObj(posChildNext, heightChildNext);
                                        listNode->push_back(objChildNext);
                                        childNextSibling = childNextSibling-
>getNextSibling();
                                }
                        }
                }

        }
        else if(t == Lex::POUND)      //gets the size of a string or list
        {
                int posChild1, heightChild1;
                child1->getPosition(posChild1, heightChild1);
                Obj *objChild1 = child1->table()->getObj(posChild1, heightChild1);
                int eval = child1->evaltype();

                if ( eval == STRING ){
                        string ch1 = *((string*)objChild1->data);
                        int l = ch1.length();
                        Obj *nodeObj = new Obj(INTEGER, l);
                        table->add(nodeObj, uniqueName());
                        node->setPosition(node->table()->size() - 1, 0);

                }else if ( eval == LIST){
                        int counter = 0;
                        ObjList *listChild1 = (ObjList*)objChild1->data;
                        vector<Obj *>::iterator theIterator;
                        for( theIterator = listChild1->begin(); theIterator != listChild1-
>end(); theIterator++ )
                                counter++;
                        Obj *nodeObj = new Obj(INTEGER, counter);
                        table->add(nodeObj, uniqueName());
                        node->setPosition(node->table()->size() - 1, 0);
                }
        }
        //note: if we append a constant like this to a list we want to make a copy of it and
not copy it directly
        //the reason being that if the list has a scope that is more global than the object,
the object may have to
        //hang around but we don't want it to be modified by future passes through this
scope
        else if(t == Lex::NUMBER)
        {
```

```
		int pos, height;
		node->getPosition(pos,height);
		Obj *nodeObj = node->table()->getObj(pos, height);
		if ( node->evaltype() == INTEGER){
			node->table()->setObj(new Obj(node->evaltype(),
*((long*)nodeObj->data)), pos, height);
		}else{
			node->table()->setObj(new Obj(node->evaltype(),
*((double*)nodeObj->data)), pos, height);
		}
	}
	else if(t == Lex::STRING_CONSTANT)
	{
		int pos, height;
		node->getPosition(pos,height);
		Obj *nodeObj = node->table()->getObj(pos, height);
		string test = *((string*)nodeObj->data);

		node->table()->setObj(new Obj(*((string*)nodeObj->data)), pos, height);

	}

	else if(t == Lex::DOT)  //e.g. res.resistance
	{
		int pos, height;
		Obj* dotObj;
		Obj *objC1;
		node->getPosition(pos, height);
		dotObj = node->table()->getObj(pos, height);
		child1->getPosition(pos, height);
		objC1 = child1->table()->getObj(pos, height);

		//printf("\nnode: %s  child1: %s\n", node->getText().c_str(), child1-
>getText().c_str()); //for debugging
		//printf("\nrestest: %f\n", ((Res*)objC1->data)->resistance); //for
debugging
		//printf("\ndottype: %s\n", numToType(dotObj->type).c_str()); //for
debugging

		if( (child2text == "pos" || child2text == "neg") && (c1 == RES || c1 ==
CAP || c1 == IND || c1 == VS || c1 == CS || c1 == DIODE || c1 == UNKNOWN) )
		{

			if(child2text == "pos") dotObj->data = ((SMLData*) (objC1-
>data))->pos;
```

```
			else if(child2text == "neg") dotObj->data = ((SMLData*) (objC1->data))->neg;
		}
		else if(c1 == RES || c1 == UNKNOWN)
		{

			if(child2text == "resistance") dotObj->data = & ( ((Res*) (objC1->data))->resistance );
			else if(child2text == "argument") dotObj->data = & ( ((Res*) (objC1->data))->argument );

			//printf("\nset to: %f\n", dotObj->data);
		}
		else if(c1 == CAP || c1 == UNKNOWN)
		{
			if(child2text == "capacitance") dotObj->data = & ( ((Cap*) (objC1->data))->capacitance );
			else if(child2text == "initial_voltage") dotObj->data = & ( ((Cap*) (objC1->data))->initial_voltage );
			else if(child2text == "argument") dotObj->data = & ( ((Cap*) (objC1->data))->argument );
		}
		else if(c1 == IND || c1 == UNKNOWN)
		{
			if(child2text == "inductance") dotObj->data = & ( ((Ind*) (objC1->data))->inductance );
			else if(child2text == "initial_current") dotObj->data = & ( ((Ind*) (objC1->data))->initial_current );
			else if(child2text == "argument") dotObj->data = & ( ((Ind*) (objC1->data))->argument );
		}
		else if(c1 == VS || c1 == UNKNOWN)
		{
			if(child2text == "voltage") dotObj->data = & ( ((Vs*) (objC1->data))->voltage );
			else if(child2text == "argument") dotObj->data = & ( ((Vs*) (objC1->data))->argument );
		}
		else if(c1 == CS || c1 == UNKNOWN)
		{
			if(child2text == "current") dotObj->data = & ( ((Cs*) (objC1->data))->current );
			else if(child2text == "argument") dotObj->data = & ( ((Cs*) (objC1->data))->argument );
		}
		else if(c1 == DIODE || c1 == UNKNOWN)
```

```
		{
			if(child2text == "argument") dotObj->data = & ( ((Diode*)
(objC1->data))->argument );
		}
		else if(c1 == BJT)
		{
			if(child2text == "base") dotObj->data = & ( ((Bjt*) (objC1-
>data))->base );
			else if(child2text == "collector") dotObj->data = & ( ((Bjt*)
(objC1->data))->collector );
			else if(child2text == "emitter") dotObj->data = & ( ((Bjt*) (objC1-
>data))->emitter );
			else if(child2text == "argument") dotObj->data = & ( ((Bjt*)
(objC1->data))->argument );
		}
		else if(c1 == MOSFET || c1 == UNKNOWN)
		{

			if(child2text == "gate") dotObj->data = & ( ((Mosfet*) (objC1-
>data))->gate );
			else if(child2text == "source") dotObj->data = & ( ((Mosfet*)
(objC1->data))->source );
			else if(child2text == "drain") dotObj->data = & ( ((Mosfet*)
(objC1->data))->drain );
			else if(child2text == "argument") dotObj->data = & ( ((Mosfet*)
(objC1->data))->argument );
		}

	}

	else if(t == Lex::ARROW)
	{



		if (child1text == "ground"){
			int posChild2, heightChild2;
			child2->getPosition(posChild2, heightChild2);
			Obj *objChild2 = child2->table()-
>getObj(posChild2,heightChild2);
			ground.connect((SMLNode*)(objChild2->data));
		}else if ( child2text == "ground"){
			int posChild1, heightChild1;
			child1->getPosition(posChild1, heightChild1);
			Obj *objChild1 = child2->table()-
>getObj(posChild1,heightChild1);
			ground.connect((SMLNode*)(objChild1->data));
```

```
		}else{
			int posChild1, posChild2, heightChild1, heightChild2;
			child1->getPosition(posChild1, heightChild1);
			child2->getPosition(posChild2, heightChild2);
			Obj *objChild1 = child1->table()-
>getObj(posChild1,heightChild1);
			Obj *objChild2 = child2->table()-
>getObj(posChild2,heightChild2);
			((SMLNode*)(objChild1->data))-
>connect((SMLNode*)(objChild2->data));
		}

		//int a = ((SMLNode*)(objChild1->data))->count();
		//a = ((SMLNode*)(objChild2->data))->count();

	}else if (t == Lex::SPICE){

		spiceInjection.append(child1text);
		spiceInjection.append("\n");   //separate diffent spice injections with !!
		printf("spice injection %s", spiceInjection.c_str());
	}else if ( text == "print")
	{
		int loc, height;
		child1->getPosition(loc, height);
		Obj *objChild1 = child1->table()->getObj(loc,height);
		int eval = objChild1->type;

		if(isSMLType(eval)) printNodes(objChild1);

		if ( eval == INTEGER)printf("%ld\n",*((long*)objChild1->data));
		else if ( eval == FLOAT ) printf("%f\n",*((double*)objChild1->data));
		else if ( eval == STRING ) {
			string s = *((string*)objChild1->data);
			printf("%s\n",s.c_str());
		}

		else if (eval == RES){
			Res *resChild3  = (Res*)objChild1->data;
			if ( resChild3->resistance != 0) printf("  res=%f\n", resChild3-
>resistance);
			else printf("  %s\n", resChild3->argument.c_str());
		}else if (eval == IND){
			Ind *indChild3  = (Ind*)objChild1->data;
			if ( indChild3->inductance != 0 )  printf("  ind=%f\n", indChild3-
>inductance);
```

```
                        else if ( indChild3->initial_current != 0 ) printf("  initcur=%f\n",
indChild3->initial_current);
                        else printf("%s\n", indChild3->argument.c_str());
                }else if (eval == VS){
                        Vs *vsChild3  = (Vs*)objChild1->data;
                        if (vsChild3->voltage != 0) printf("  volt=%f\n", vsChild3-
>voltage);
                        else printf("  %s\n", vsChild3->argument.c_str());
                }else if (eval == CAP){
                        Cap *capChild3  = (Cap*)objChild1->data;
                        if (capChild3->capacitance != 0) printf("  cap=%f\n", capChild3-
>capacitance);
                        else if (capChild3->initial_voltage != 0) printf(" initvolt=%f\n",
capChild3->initial_voltage);
                        else printf("  %s\n", capChild3->argument.c_str());
                }else if (eval == CS){
                        Cs *csChild3  = (Cs*)objChild1->data;
                        if ( csChild3->current != 0) printf("  cur=%f\n", csChild3-
>current);
                        else printf("  %s\n", csChild3->argument.c_str());
                }else if (eval == DIODE){
                        Diode *diodeChild3 = (Diode*)objChild1->data;
                        printf("  %s\n", diodeChild3->argument.c_str());
                }else if (eval == BJT ){
                        Bjt *bjtChild3 = (Bjt*)objChild1->data;
                        printf("  %s\n", bjtChild3->argument.c_str());
                }else if ( eval == MOSFET ){
                        Mosfet *mosChild3 = (Mosfet*)objChild1->data;
                        printf("  %s\n",  mosChild3->argument.c_str());
                }
                else if (eval == NODE){
                        printNodeList((SMLNode*)objChild1->data);
                }

                else if ( eval == LIST) {
                        ObjList *listChild1 = (ObjList*)objChild1->data;
                        vector<Obj *>::iterator theIterator;
                        for( theIterator = listChild1->begin(); theIterator != listChild1-
>end(); theIterator++ ){
                                Obj *objChild3 = *theIterator;
                                int eval2 = objChild3->type;

                                if(isSMLType(eval2)) printNodes(*theIterator);

                                if (eval2 == RES){
                                        Res *resChild3  = (Res*)objChild3->data;
```

```
                                        if ( resChild3->resistance != 0) printf("  res=%f\n",
resChild3->resistance);
                                        else printf("  %s\n", resChild3->argument.c_str());
                                }else if (eval2 == IND){
                                        Ind *indChild3  = (Ind*)objChild3->data;
                                        if ( indChild3->inductance != 0 )  printf("
ind=%f\n", indChild3->inductance);
                                        else if ( indChild3->initial_current != 0 ) printf("
initcur=%f\n", indChild3->initial_current);
                                        else printf("  %s\n", indChild3->argument.c_str());
                                }else if (eval2 == VS){
                                        Vs *vsChild3  = (Vs*)objChild3->data;
                                        if (vsChild3->voltage != 0) printf("  volt=%f\n",
vsChild3->voltage);
                                        else printf("  %s\n", vsChild3->argument.c_str());
                                }else if (eval2 == CAP){
                                        Cap *capChild3  = (Cap*)objChild3->data;
                                        if (capChild3->capacitance != 0) printf("
cap=%f\n", capChild3->capacitance);
                                        else if (capChild3->initial_voltage != 0) printf("
initvolt=%f\n", capChild3->initial_voltage);
                                        else printf("  %s\n", capChild3->argument.c_str());
                                }else if (eval2 == CS){
                                        Cs *csChild3  = (Cs*)objChild3->data;
                                        if ( csChild3->current != 0) printf("  cur=%f\n",
csChild3->current);
                                        else printf(  "%s\n", csChild3->argument.c_str());
                                }else if (eval2 == DIODE){
                                        Diode *diodeChild3 = (Diode*)objChild3->data;
                                        printf("  %s\n", diodeChild3->argument.c_str());
                                }else if (eval2 == BJT ){
                                        Bjt *bjtChild3 = (Bjt*)objChild3->data;
                                        printf("  %s\n", bjtChild3->argument.c_str());
                                }else if ( eval2 == MOSFET ){
                                        Mosfet *mosChild3 = (Mosfet*)objChild3->data;
                                        printf("  %s\n",  mosChild3->argument.c_str());
                                }else if ( eval2 ==
INTEGER)printf("%ld\n",*((long*)objChild3->data));
                                else if ( eval2 == FLOAT )
printf("%f\n",*((double*)objChild3->data));
                                else if ( eval2 == STRING ) {
                                         string s = *((string*)objChild3->data);
                                         printf("%s\n",s.c_str());
                                }
                                else if (eval2 == NODE){
                                        printNodeList((SMLNode*)objChild3->data);
```

```
                                    }
                        }

                }

        }
}
```

8.4 Code Generator

```
/*
        Code Generator by Ron Alleyne
        Bugs fixed by Michael Apap



  */


#ifndef __codeGen_h__
#define __codeGen_h__


#define OUTFILE "out.sp"

#include <stdio.h>
#include <string.h>
#include <fstream>
#include <iostream>

using namespace std;


// invokes node cruncher on every node that is in Obj in symbol table
//this include list
//this is the function that should be invoked in main
int listobjectcounter=1;
char *listname="list";
int nodeCounter=1; //global counter to make sure that node id is unique accross board
//extern SymbolTable globalScope;  //the global symbol table for our program
void nodeListWalker(ObjList *list);
string SMLInjecter (string original, SymbolTable table);
void nodeCruncher(SMLNode *tempNodeA);
void spicifyList(Obj *list, ofstream &out);
```

```
//utility function courtesy of http://linuxselfhelp.com/HOWTO/C++Programming-
HOWTO-7.html

void Tokenize(const string& str,vector<string>& tokens,const string&delimiters = " ")
{
   // Skip delimiters at beginning.
   string::size_type lastPos = str.find_first_not_of(delimiters, 0);
   // Find first "non-delimiter".
   string::size_type pos     = str.find_first_of(delimiters, lastPos);

   while (string::npos != pos || string::npos != lastPos)
   {
      // Found a token, add it to the vector.
      tokens.push_back(str.substr(lastPos, pos - lastPos));
      // Skip delimiters.  Note the "not_of"
      lastPos = str.find_first_not_of(delimiters, pos);
      // Find next "non-delimiter"
      pos = str.find_first_of(delimiters, lastPos);
   }
} // end tokenizer



 void nodeWalker () {


         Obj *tempObj = NULL;

printf("Consolidating Circuit Nodes \n");
         //go thru each Obj in symbol table
         for (int i=0; i<globalScope.size(); i++) {

                  tempObj = globalScope.getObj(i, 0);

                  //make sure its a circuit element or a list
                  if ((int) tempObj->type < LIST) {//see sml_h for numbers
                           continue;
                  }
                  else if (tempObj->type ==LIST) {
                           nodeListWalker((ObjList*) tempObj->data);
                  }

                  //now for each standard element: look @ its nodes
                  else if (tempObj->type==RES || tempObj->type==CAP || tempObj-
>type==IND ||tempObj->type==VS ||tempObj->type==CS || tempObj->type==DIODE) {
                           nodeCruncher( ((SMLData*)tempObj->data)->pos );
```

```
			nodeCruncher(((SMLData*)tempObj->data)->neg);
		}
		else if (tempObj->type==BJT){
			nodeCruncher(((Bjt*)tempObj->data)->base);
			nodeCruncher(((Bjt*)tempObj->data)->emitter);
			nodeCruncher(((Bjt*)tempObj->data)->collector);
		}
		else if (tempObj->type==MOSFET){
			nodeCruncher(((Mosfet*)tempObj->data)->drain);
			nodeCruncher(((Mosfet*)tempObj->data)->source);
			nodeCruncher(((Mosfet*)tempObj->data)->gate);
		}

	} // end for loop for symbol table traversal in search of node Objs
} //end void noid walker



void nodeListWalker(ObjList *list) {

	//go thru vector pointer and look at time of entry. if its a list, make recursive call;
if its not call node Cruncher

		Obj *tempObj;

		for (unsigned int i=0; i<list->size(); i++) {

			tempObj = (*list)[i]; //get ith Obj

			if ((int) tempObj->type < LIST) { //see sml_h for numbers
			continue;
			}
			else if ((int) tempObj->type == LIST) {
				nodeListWalker((ObjList*) tempObj->data); //make
recursive call if its a list
			}
			//now for each standard element: look @ its nodes
			else if (tempObj->type==RES || tempObj->type==CAP || tempObj-
>type==IND ||tempObj->type==VS ||tempObj->type==CS || tempObj->type==DIODE) {
			nodeCruncher(((SMLData*)tempObj->data)->pos);
			nodeCruncher(((SMLData*)tempObj->data)->neg);
			}
			else if (tempObj->type==BJT){
			nodeCruncher(((Bjt*)tempObj->data)->base);
			nodeCruncher(((Bjt*)tempObj->data)->emitter);
			nodeCruncher(((Bjt*)tempObj->data)->collector);
```

```
                    }
                    else if (tempObj->type==MOSFET){
                    nodeCruncher(((Mosfet*)tempObj->data)->drain);
                    nodeCruncher(((Mosfet*)tempObj->data)->source);
                    nodeCruncher(((Mosfet*)tempObj->data)->gate);
                    }

            } // end step thru all list elements


        }


        //this is routine that implements node crunching algorithm
        //every node touched will be named

        void nodeCruncher(SMLNode *tempNodeA) {

                SMLNode *tempNodeB, *tempNodeC; Obj *tempObj;

                //if node has name already, its been crunched through a connection to
another Obj so we'll leave it alone .
cout<<"Node Name considered"<<endl;


                if (tempNodeA->name!="") {
                        return; }


                //for all of this node's connections
                for (int j=0; j<tempNodeA->count(); j++) {

                        tempNodeB = (*tempNodeA)[j]; //gets jth Connecton
cout<<"This nodes's "<<itos(j)<<" th connection being tested"<<endl;
                        tempObj=tempNodeB->parent;



                        //change parent circuit element Obj of node B to point to Node A
                        if(tempNodeB->parent!=NULL && tempNodeB->parent-
>type<BJT) {
                                cout<<"Setting parent for standard Device"<<endl;
                                if (tempNodeB==(((SMLData*)tempObj->data))->pos) {
                                        ((SMLData*)tempObj->data)->pos=tempNodeA;
                                }
```

```
					else if (tempNodeB==(((SMLData*)tempObj->data))-
>neg) {
							((SMLData*)tempObj->data)-
>neg=tempNodeA;
					}
					cout<<"Done Setting parent for standard Device"<<endl;
				} // end if <bjt
				else if(tempNodeB->parent!=NULL && tempNodeB->parent-
>type==BJT) {
					if (tempNodeB==(((Bjt*)tempObj->data))->base) {
					((Bjt*)tempObj->data)->base=tempNodeA;
					}
					else if (tempNodeB==(((Bjt*)tempObj->data))->emitter) {
					((Bjt*)tempObj->data)->emitter=tempNodeA;
					}
					else if (tempNodeB==(((Bjt*)tempObj->data))->collector)
{
					((Bjt*)tempObj->data)->collector=tempNodeA;
					}

				} //end if =bjt
				else if(tempNodeB->parent!=NULL && tempNodeB->parent-
>type==MOSFET) {
					if (tempNodeB==(((Mosfet*)tempObj->data))->drain) {
					((Mosfet*)tempObj->data)->drain=tempNodeA;
					}
					else if (tempNodeB==(((Mosfet*)tempObj->data))-
>source) {
					((Mosfet*)tempObj->data)->source=tempNodeA;
					}
					else if (tempNodeB==(((Mosfet*)tempObj->data))->gate) {
					((Mosfet*)tempObj->data)->gate=tempNodeA;
					}

				}//end if =mosfet
				else {
				CrashAndBurn(__LINE__, __FILE__, "Circuit Node w/out parent
encountered");
				//some other type of Obj binded
				}
				//tempNodeB-

				//for all of this node B's connections
				cout<<"Disconnecting Node B's Devices"<<endl;
				int temp=tempNodeB->count();
				for (int k=0; k<temp; k++) {
```

```
                                        // get node b's kth connecting node
                                        tempNodeC=(*tempNodeB)[0];

                                        //disconnect the node from this array or really remove it
from node b's array of node pointers
                                        tempNodeB->disconnect(0);


                                        if (tempNodeC!=tempNodeA) {
                                                tempNodeA->connect(tempNodeC);

                                        }// if this is Node B's connection to node A, then it need not
be added to A's list, but A must have a connection to every other one.



                                } // end for loop for tempNodeB (a node connected to this node
(B))
                                cout<<"Done Disconnecting Node B's Devices"<<endl;

                        }// end for loop for this node's connections

                                cout<<"Attempting to name device "<<endl;
                //give node a new name based on counter iff this is a not a node that has been
gutted
                        if (tempNodeA->count()>0 && tempNodeA->parent!=NULL) {

                                tempNodeA->name = "node" + itos(nodeCounter++ );

                        }
                        else if (tempNodeA->parent==NULL){ //case of ground - it will not have
a parent and will still be passed into invocation of this function.

                                tempNodeA->name="0";
                        } //end else
                        //cout<<"done trying to name device "<<endl;
                }
//then we do spice SMLInjecter here

                //useful aid http://www.msoe.edu/eecs/ce/courseinfo/stl/string.htm
string SMLInjecter (string original, SymbolTable table) {

string objName, fieldName, aToken;
bool hasDot;  int tableHeight, tablePosition;
//used to find indices of substrings of interest.
```

```
string::size_type start=0, finish, dotStart;
Obj *tempObj;

//loop of some sort ?????????? till string is done

//find al #....." " interval
do {
hasDot=false;
objName="";
//look for the next/first pound
start = original.find ("#",start);

if (start==string::npos) {//done with string, there is no more #
        break; }

//look for the space
finish= original.find (" ",start);

if (finish==string::npos) {//done with string, there is no more " "

        CrashAndBurn(__LINE__, __FILE__, "# Sign w/out space after expression
noted");
}

//get rid of # from string
original.replace(start,1," ");

//get index of dot start
dotStart=original.find(".", start);

if (dotStart==string::npos) {//get location of first # indicating spice injection
objName=original.substr(start+1,finish-(start+1));
//hasDot=false;
}
else {
        hasDot=true;
        objName=original.substr(start+1,dotStart-(start+1));
        fieldName=original.substr(dotStart+1, finish-(dotStart+1));
}

table.get(objName, tablePosition, tableHeight);

if (tablePosition==-1) {
        cout<<"WARNING:SML object in this line "<<original<<" not found";
        return original;//out<<"SML object in this line "<<original<<" not found";
```

```
	//CrashAndBurn(__LINE__, __FILE__, "Attempting to inject Obj not in symbol
table");
}

//get the Obj
tempObj=table.getObj(tablePosition,tableHeight);

//replace based on what it is string, integer, float, cast appropriately.
if (!hasDot && tempObj->type==INTEGER) {
//straightforward substitution
long *val = (long *) tempObj->data;
original.replace(start+1, finish-(start+1), itos((int) *val));
}
else if (!hasDot && tempObj->type==FLOAT) {
//straightforward substitution
double *val = (double *) tempObj->data;
original.replace(start+1, finish-(start+1), dtosp((float) *val));
}
else if (!hasDot && tempObj->type==STRING) {
//straightforward substitution
string *val = (string *) tempObj->data;
original.replace(start+1, finish-(start+1), *val);
}
else if (!hasDot && tempObj->type==RES) {
//straightforward substitution
string val = "r" + objName;
original.replace(start+1, finish-(start+1), val);
}
else if (!hasDot && tempObj->type==IND) {
//straightforward substitution
string val = "l" + objName;
original.replace(start+1, finish-(start+1), val);
}
else if (!hasDot && tempObj->type==CAP) {
//straightforward substitution
string val = "c" + objName;
original.replace(start+1, finish-(start+1), val);
}
else if (!hasDot && tempObj->type==VS) {
//straightforward substitution
string val = "v" + objName;
original.replace(start+1, finish-(start+1), val);
}
else if (!hasDot && tempObj->type==CS) {
//straightforward substitution
string val = "i" + objName;
```

```
original.replace(start+1, finish-(start+1), val);
}
else if (!hasDot && tempObj->type==DIODE) {
//straightforward substitution
string val = "d" + objName;
original.replace(start+1, finish-(start+1), val);
}


else if (!hasDot || tempObj->type==LIST){CrashAndBurn(__LINE__, __FILE__,
"Attempting to inject SML Obj without specifying field. Lists or Transistors aren't elgible
for this");}


 //it has dot so for each type we have to see if fieldName matches a particular field
else if (hasDot && tempObj->type==RES) {

                        if (fieldName=="resistance") {
                //straightforward substitution
                double val =  ((Res *) tempObj->data)->resistance;
                original.replace(start+1, finish-(start+1), dtosp((float) val));
                }
                else if (fieldName=="argument") {
                //straightforward substitution
                string val =  ((Res *) tempObj->data)->argument;
                original.replace(start+1, finish-(start+1), val);
                }
                else if (fieldName=="pos") {
                //straightforward substitution
                string val =  ((Res *) tempObj->data)->pos->name;
                original.replace(start+1, finish-(start+1), val);
                }
                else if (fieldName=="neg") {
                //straightforward substitution
                string val =  ((Res *) tempObj->data)->neg->name;
                original.replace(start+1, finish-(start+1), val);
                }
                else {CrashAndBurn(__LINE__, __FILE__, "Attempting to inject
unrecognized resistor property value");}

} // end Res
else if (hasDot && tempObj->type==CAP) {

                        if (fieldName=="capacitance") {
                //straightforward substitution
                double val =  ((Cap *) tempObj->data)->capacitance;
```

```
		original.replace(start+1, finish-(start+1), dtosp((float) val));
		}
		else if (fieldName=="argument") {
		//straightforward substitution
		string val =  ((Cap *) tempObj->data)->argument;
		original.replace(start+1, finish-(start+1), val);
		}
		else if (fieldName=="pos") {
		//straightforward substitution
		string val =  ((Cap *) tempObj->data)->pos->name;
		original.replace(start+1, finish-(start+1), val);
		}
		else if (fieldName=="neg") {
		//straightforward substitution
		string val =  ((Cap *) tempObj->data)->neg->name;
		original.replace(start+1, finish-(start+1), val);
		}
		else {CrashAndBurn(__LINE__, __FILE__, "Attempting to inject
unrecognized capacitor property value");}

} // end Cap
else if (hasDot && tempObj->type==IND) {

			if (fieldName=="inductance") {
		//straightforward substitution
		double val =  ((Ind *) tempObj->data)->inductance;
		original.replace(start+1, finish-(start+1), dtosp((float) val));
		}
		else if (fieldName=="argument") {
		//straightforward substitution
		string val =  ((Ind *) tempObj->data)->argument;
		original.replace(start+1, finish-(start+1), val);
		}
		else if (fieldName=="pos") {
		//straightforward substitution
		string val =  ((Ind *) tempObj->data)->pos->name;
		original.replace(start+1, finish-(start+1), val);
		}
		else if (fieldName=="neg") {
		//straightforward substitution
		string val =  ((Ind *) tempObj->data)->neg->name;
		original.replace(start+1, finish-(start+1), val);
		}
		else {CrashAndBurn(__LINE__, __FILE__, "Attempting to inject
unrecognized inductor property value");}
```

```
} // end Ind
else if (hasDot && tempObj->type==DIODE) {

         if (fieldName=="argument") {
        //straightforward substitution
        string val =  ((Diode *) tempObj->data)->argument;
        original.replace(start+1, finish-(start+1), val);
        }
        else if (fieldName=="pos") {
        //straightforward substitution
        string val =  ((Diode *) tempObj->data)->pos->name;
        original.replace(start+1, finish-(start+1), val);
        }
        else if (fieldName=="neg") {
        //straightforward substitution
        string val =  ((Diode *) tempObj->data)->neg->name;
        original.replace(start+1, finish-(start+1), val);
        }
        else {CrashAndBurn(__LINE__, __FILE__, "Attempting to inject
unrecognized diode property value");}

} // end Diode
else if (hasDot && tempObj->type==VS) {

                if (fieldName=="voltage") {
        //straightforward substitution
        double val =((Vs *) tempObj->data)->voltage;
        original.replace(start+1, finish-(start+1), dtosp((float) val));
        }
        else if (fieldName=="argument") {
        //straightforward substitution
        string val =  ((Vs *) tempObj->data)->argument;
        original.replace(start+1, finish-(start+1), val);
        }
        else if (fieldName=="pos") {
        //straightforward substitution
        string val =  ((Vs *) tempObj->data)->pos->name;
        original.replace(start+1, finish-(start+1), val);
        }
        else if (fieldName=="neg") {
        //straightforward substitution
        string val =  ((Vs *) tempObj->data)->neg->name;
        original.replace(start+1, finish-(start+1), val);
        }
        else {CrashAndBurn(__LINE__, __FILE__, "Attempting to inject
unrecognized voltage source property value");}
```

```
} // end Vs
else if (hasDot && tempObj->type==CS) {

                if (fieldName=="current") {
            //straightforward substitution
            double val =  ((Cs *) tempObj->data)->current;
            original.replace(start+1, finish-(start+1), dtosp((float) val));
            }
            else if (fieldName=="argument") {
            //straightforward substitution
            string val =  ((Cs *) tempObj->data)->argument;
            original.replace(start+1, finish-(start+1), val);
            }
            else if (fieldName=="pos") {
            //straightforward substitution
            string val =  ((Cs *) tempObj->data)->pos->name;
            original.replace(start+1, finish-(start+1), val);
            }
            else if (fieldName=="neg") {
            //straightforward substitution
            string val =  ((Cs *) tempObj->data)->neg->name;
            original.replace(start+1, finish-(start+1), val);
            }
            else {CrashAndBurn(__LINE__, __FILE__, "Attempting to inject
unrecognized current source property value");}
} // end Cs
else if (hasDot && tempObj->type==BJT) {

             if (fieldName=="argument") {
            //straightforward substitution
            string val =  ((Bjt *) tempObj->data)->argument;
            original.replace(start+1, finish-(start+1), val);
            }
            else if (fieldName=="base") {
            //straightforward substitution
            string val =  ((Bjt *) tempObj->data)->base->name;
            original.replace(start+1, finish-(start+1), val);
            }
            else if (fieldName=="emitter") {
            //straightforward substitution
            string val =  ((Bjt *) tempObj->data)->emitter->name;
            original.replace(start+1, finish-(start+1), val);
            }
            else if (fieldName=="collector") {
            //straightforward substitution
            string val =  ((Bjt *) tempObj->data)->collector->name;
```

```
                    original.replace(start+1, finish-(start+1), val);
                    }
                    else {CrashAndBurn(__LINE__, __FILE__, "Attempting to inject
unrecognized bjt source property       value");}
} // end BJT

else if (hasDot && tempObj->type==MOSFET) {

                     if (fieldName=="argument") {
                    //straightforward substitution
                    string val =  ((Mosfet *) tempObj->data)->argument;
                    original.replace(start+1, finish-(start+1), val);
                    }
                    else if (fieldName=="gate") {
                    //straightforward substitution
                    string val =  ((Mosfet *) tempObj->data)->gate->name;
                    original.replace(start+1, finish-(start+1), val);
                    }
                    else if (fieldName=="drain") {
                    //straightforward substitution
                    string val =  ((Mosfet *) tempObj->data)->drain->name;
                    original.replace(start+1, finish-(start+1), val);
                    }
                    else if (fieldName=="source") {
                    //straightforward substitution
                    string val =  ((Mosfet *) tempObj->data)->source->name;
                    original.replace(start+1, finish-(start+1), val);
                    }

                    else {CrashAndBurn(__LINE__, __FILE__, "Attempting to inject
unrecognized bjt source property value");}
} // end BJT
else { // unmatched obj type w/ dot
CrashAndBurn(__LINE__, __FILE__, "Attempting to inject SML Obj without parameter
values. List cannot be manipulated");
}

}
//! end somehow.
while (start<original.size()-1);


return original;
}
```

```
        //built based on specifications on hspice
//file name is original file, out is file create
//toInject is spice code to be parsed
        void spicify (string filename, string outFile, string toInject) {

         ofstream out;
         string node1Name, node2Name, node3Name, valueString, value2String;
         Obj *tempObj =NULL;
         vector<string>spiceStrings;
        out.open(outFile.c_str());
        //out.open(outFile);
         //put usual stuff in beginning of hspice file
         out<<"This spice list generated by SML Compiler for " + filename + "\n";

         out<<"*options \n";
         out<<".options post reltol=1e-6 ";

        out<<"\n *simulation commands, all parameters (models, etc)\n";

        //go thru string of injected spice and tokenize based on new lines and for each line
run spiceInjection
Tokenize(toInject,spiceStrings, "\n");
unsigned int i=0;
for ( i=0; i<spiceStrings.size(); i++) {

 out<<SMLInjecter(spiceStrings[i],globalScope)<<endl;

} // end injection




        //now go thru symbol table. (if type is > certain value or a value where looking
for)
        //go thru each Obj in symbol table
        for (i=0; i<globalScope.size(); i++) {

        //for every element, print name, need block for each element ( i think)
        //basic idea is to go to terminal (node) get its name (first pos, then neg) (get BJT
convention in hspice down)
        //then print value or string (string argument if value = -1 and value otherwise)
                tempObj = globalScope.getObj(i, 0);

                //make sure its a circuit element or a list or hasn't been seen yet
```

```
		if ((int) tempObj->type < LIST || ((SMLData*)tempObj->data)-
>touched==1 ) {//see sml_h for numbers
			continue;
		}
		else if (tempObj->type ==LIST) {
			//spicifyList((ObjList*) tempObj->data, out);
			spicifyList((Obj*) tempObj, out);
		}


			else if (tempObj->type==VS) {
			((SMLData*) tempObj->data)->touched=1;
			node1Name = ((Vs*)tempObj->data)->pos->name; //get node
names
			node2Name = ((Vs*)tempObj->data)->neg->name; //get node
names

				if (((Vs*)tempObj->data)->argument=="") {
					//double d=((Vs*)tempObj->data)->voltage;

					//sprintf((char *)valueString.c_str(),"%.2d",d);
					valueString= dtosp(((Vs*)tempObj->data)-
>voltage);

				}
					else {valueString= ((Vs*)tempObj->data)-
>argument;
					//inject SML Objs into any argument they may have
made
					valueString=SMLInjecter(valueString,
globalScope);

					}

					out<<"v" + globalScope.getName(i, 0) + " "
+node1Name + " " + node2Name + " " + valueString <<endl;

		}


	else if (tempObj->type==CS) {
			((SMLData*) tempObj->data)->touched=1;
			node1Name = ((Cs*)tempObj->data)->pos->name; //get node
names
			node2Name = ((Cs*)tempObj->data)->neg->name; //get node
names
```

```
                    if (((Cs*)tempObj->data)->argument=="") {
                        valueString= dtosp(((Cs*)tempObj->data)-
>current);


                    }
                        else {valueString= ((Cs*)tempObj->data)-
>argument;
                        //inject SML Objs into any argument they may have
made
                        valueString=SMLInjecter(valueString,
globalScope);

                        }

                        out<<"i" + globalScope.getName(i, 0) + " "
+node1Name + " " + node2Name + " " + valueString <<endl;

        }



                //now for each standard element: look @ its nodes
                else if (tempObj->type==RES) {
                ((SMLData*) tempObj->data)->touched=1;
                node1Name = ((Res*)tempObj->data)->pos->name; //get node
names
                node2Name = ((Res*)tempObj->data)->neg->name; //get node
names

                    if (((Res*)tempObj->data)->argument=="") {
                        valueString= dtosp(((Res*)tempObj->data)-
>resistance);


                    }
                        else {valueString= ((Res*)tempObj->data)-
>argument;
                        //inject SML Objs into any argument they may have
made
                        valueString=SMLInjecter(valueString,
globalScope);

                        }
```

```
					out<<"r" + globalScope.getName(i, 0) + " "
+node1Name + " " + node2Name + " " + valueString <<endl;

			}
				else if (tempObj->type==DIODE) {
					((SMLData*) tempObj->data)->touched=1;
						node1Name = ((Diode*)tempObj->data)->pos-
>name; //get node names
						node2Name = ((Diode*)tempObj->data)->neg-
>name; //get node names

						//has to be argument - nothing else
						valueString= ((Diode*)tempObj->data)->argument;

						//inject SML Objs into any argument they may have
made
						valueString=SMLInjecter(valueString,
globalScope);


						out<<"d" + globalScope.getName(i, 0) + " "
+node1Name + " " + node2Name + " " + valueString <<endl;

			}
					else if (tempObj->type==CAP) {
						((SMLData*) tempObj->data)->touched=1;
						node1Name = ((Cap*)tempObj->data)->pos-
>name; //get node names
						node2Name = ((Cap*)tempObj->data)->neg-
>name; //get node names

						if (((Cap*)tempObj->data)->argument=="") {
							valueString= dtosp(((Cap*)tempObj->data)-
>capacitance);

						}
						else {valueString= ((Cap*)tempObj->data)-
>argument;
						//inject SML Objs into any argument they may have
made
						valueString=SMLInjecter(valueString,
globalScope);

						}
```

```
					value2String = "IC=" + dtosp(((Cap*)tempObj-
>data)->initial_voltage);

				out<<"c" + globalScope.getName(i, 0) + " " +node1Name
+ " " + node2Name + " " + valueString + " " + value2String<<endl;

				}
				else if (tempObj->type==IND) {
					((SMLData*) tempObj->data)->touched=1;
					node1Name = ((Ind*)tempObj->data)->pos->name;
//get node names
					node2Name = ((Ind*)tempObj->data)->neg->name;
//get node names

					if (((Ind*)tempObj->data)->argument=="") {
						valueString= dtosp(((Ind*)tempObj->data)-
>inductance);
					}
					else {valueString= ((Ind*)tempObj->data)-
>argument;
					//inject SML Objs into any argument they may have
made
					valueString=SMLInjecter(valueString,
globalScope);

					}

					value2String = "IC=" + dtosp(((Ind*)tempObj-
>data)->initial_current);

				out<<"l" + globalScope.getName(i, 0) + " " +node1Name +
" " + node2Name + " " + valueString + " " + value2String<<endl;

				}

				else if (tempObj->type==BJT){
				((SMLData*) tempObj->data)->touched=1;
				node1Name=((Bjt*)tempObj->data)->collector->name;
				node2Name=((Bjt*)tempObj->data)->base->name;
				node3Name=((Bjt*)tempObj->data)->emitter->name;

					//has to be argument - nothing else
					valueString= ((Bjt*)tempObj->data)->argument;

					//inject SML Objs into any argument they may have
made
```

```
					valueString=SMLInjecter(valueString,
globalScope);

					out<<"q" + globalScope.getName(i, 0) + " "
+node1Name + " " + node2Name + " " + node3Name + " " + valueString <<endl;

				}
				else if (tempObj->type==MOSFET){
					((SMLData*) tempObj->data)->touched=1;
					node1Name=((Mosfet*)tempObj->data)->drain-
>name;
					node2Name=((Mosfet*)tempObj->data)->gate-
>name;
					node3Name=((Mosfet*)tempObj->data)->source-
>name;

					//has to be argument - nothing else
					valueString= ((Bjt*)tempObj->data)->argument;

					//inject SML Objs into any argument they may have
made
					valueString=SMLInjecter(valueString,
globalScope);

				out<<"q" + globalScope.getName(i, 0) + " " +node1Name
+ " " + node2Name + " " + node3Name + " " + valueString <<endl;


				} // end else
				else {
					//CrashAndBurn(__LINE__, __FILE__, "Circuit
Node w/out parent encountered");
				}
			//some other type of Obj binded			}


	} // end of for loop for symbol table
//make sure to include .end @ end of file
out<<".end";

//close file



}
```

```
void spicifyList(Obj *listobject, ofstream &out) {

        ObjList* list=(ObjList* )listobject->data;



        string name_of_list=globalScope.getName(globalScope.getLocal(listobject),0);
        Obj *tempObj=NULL;
string node1Name, node2Name, node3Name, valueString, value2String;


                for (unsigned int i=0; i<list->size(); i++) {

                        tempObj = (*list)[i]; //get ith Obj

                        if ((int) tempObj->type < LIST) { //see sml_h for numbers
                        continue;
                        }else if (tempObj->type ==LIST) {
                        spicifyList((Obj*) tempObj, out);
                        }

                        //now for each standard element: look @ its nodes
                        else if (tempObj->type==RES) {
                        ((SMLData*) tempObj->data)->touched=1;
                        node1Name = ((SMLData*)tempObj->data)->pos->name; //get
node names
                        node2Name = ((SMLData*)tempObj->data)->neg->name; //get
node names

                                if (((Res*)tempObj->data)->argument=="") {
                                        valueString= dtosp(((Res*)tempObj->data)-
>resistance);


                                }
                                        else {valueString= ((Res*)tempObj->data)-
>argument;
                                        //inject SML Objs into any argument they may have
made
                                        valueString=SMLInjecter(valueString,
globalScope);

                                        }
```

```
                                        out<<"r" + name_of_list +
"_xyz"+itos(listobjectcounter++)+" " +node1Name + " " + node2Name + " " +
valueString <<endl;

                    }
                              else if (tempObj->type==DIODE) {
                                        ((SMLData*) tempObj->data)->touched=1;
                                        node1Name = ((SMLData*)tempObj->data)->pos-
>name; //get node names
                                        node2Name = ((SMLData*)tempObj->data)->neg-
>name; //get node names

                                        //has to be argument - nothing else
                                        valueString=((Diode*)tempObj->data)->argument;

                                        //inject SML Objs into any argument they may have
made
                                        valueString=SMLInjecter(valueString,
globalScope);


                                        out<<"d"+ name_of_list +
"_xyz"+itos(listobjectcounter++)+" " +node1Name + " " + node2Name + " " +
valueString <<endl;

                    }
                                        else if (tempObj->type==CAP) {
                                        ((SMLData*) tempObj->data)->touched=1;
                                        node1Name = ((SMLData*)tempObj->data)->pos-
>name; //get node names
                                        node2Name = ((SMLData*)tempObj->data)->neg-
>name; //get node names

                                        if (((Cap*)tempObj->data)->argument=="") {
                                                  valueString= dtosp(((Cap*)tempObj->data)-
>capacitance);

                                        }
                                        else {valueString= ((Cap*)tempObj->data)-
>argument;
                                        //inject SML Objs into any argument they may have
made
                                        valueString=SMLInjecter(valueString,
globalScope);

                                        }
```

```
					value2String = "IC=" + dtosp(((Cap*)tempObj-
>data)->initial_voltage);

				out<<"c" + name_of_list +
"_xyz"+itos(listobjectcounter++)+" " +node1Name + " " + node2Name + " " +
valueString + " " + value2String<<endl;

				}
				else if (tempObj->type==IND) {
					((SMLData*) tempObj->data)->touched=1;
					node1Name = ((SMLData*)tempObj->data)->pos-
>name; //get node names
					node2Name = ((SMLData*)tempObj->data)->neg-
>name; //get node names

					if (((Ind*)tempObj->data)->argument=="") {
						valueString= dtosp(((Ind*)tempObj->data)-
>inductance);
					}
					else {valueString= ((Ind*)tempObj->data)-
>argument;
					//inject SML Objs into any argument they may have
made
					valueString=SMLInjecter(valueString,
globalScope);

					}

					value2String = "IC=" + dtosp(((Ind*)tempObj-
>data)->initial_current);

				out<<"l" + name_of_list +
"_xyz"+itos(listobjectcounter++)+" "+node1Name + " " + node2Name + " " +
valueString + " " + value2String<<endl;

				}

				else if (tempObj->type==BJT){
				((SMLData*) tempObj->data)->touched=1;
				node1Name=((Mosfet*)tempObj->data)->drain->name;
				node2Name=((Mosfet*)tempObj->data)->source->name;
				node3Name=((Mosfet*)tempObj->data)->gate->name;

					//has to be argument - nothing else
					valueString= ((Bjt*)tempObj->data)->argument;
```

```
                                        //inject SML Objs into any argument they may have
made
                                        valueString=SMLInjecter(valueString,
globalScope);

                                        out<<"q" + name_of_list +
"_xyz"+itos(listobjectcounter++)+" " +node1Name + " " + node2Name + " " +
node3Name + " " + valueString <<endl;

                                }
                                else if (tempObj->type==MOSFET){
                                        ((SMLData*) tempObj->data)->touched=1;
                                        node1Name=((Mosfet*)tempObj->data)->drain-
>name;
                                        node2Name=((Mosfet*)tempObj->data)->source-
>name;
                                        node3Name=((Mosfet*)tempObj->data)->gate-
>name;

                                        //has to be argument - nothing else
                                        valueString= ((Bjt*)tempObj->data)->argument;

                                        //inject SML Objs into any argument they may have
made
                                        valueString=SMLInjecter(valueString,
globalScope);

                                out<<"q" + name_of_list +
"_xyz"+itos(listobjectcounter++)+" "+node1Name + " " + node2Name + " " +
node3Name + " " + valueString <<endl;


                                } // end else
                        //      else {CrashAndBurn(__LINE__, __FILE__, "Circuit Node
w/out parent encountered"); }           //some other type of Obj binded


                } // end for loop


}
```

```
#endif
```

8.5 SML Header File

```
/*	SML Header file by Spencer Greenberg


  */
#ifndef __SML_h__
#define __SML_h__

/*---------------------------------------------
Included files
---------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <sstream>
#include <iostream>
#include <string>
#include <vector>
#include <map>


/*---------------------------------------------
Object Types
-These constants are stored internally in objects to
 specify type and are also stored within tree nodes
 to specify the type of object an expression evaluates to
-Never use these numbers directly! Always use the associated name
---------------------------------------------*/
#define UNDECLARED -5
#define ERROR -4		//if the node should have a type but doesn't
#define DEFAULT -3
#define UNKNOWN -2			//sometimes an expression's return type is unknown
if it accesses elements of a list
#define UNDEFINED -1	//some expressions (such as while loops) return an
"undefined" type since they produce no value
#define INTEGER 0
#define FLOAT 1
#define STRING 2
#define LIST 3
#define RES 4			//resistor
#define CAP 5			//capacitor
#define IND 6			//inductor
#define VS 7			//voltage source
```

```
#define CS 8                    //current source
#define NODE 9                          //a circuit or circuit element node
#define DIODE 10
#define BJT 11                          //BJT transistor
#define MOSFET 12               //MOSFET transistor


/*---------------------------------------------
Relavent c++ namespace inclusions
----------------------------------------------*/
using std::string;
using std::map;
using std::vector;


/*---------------------------------------------
Global utility functions
----------------------------------------------*/
/*
We should all use this function (for now) in
error checking. Since this error function takes a
string it allows us to include more debugging information
than assert would alone.

it is called like this: CrashAndBurn(__LINE__, __FILE__, "a message about why the
crash occured");



The macros __LINE__ and __FILE__ will include the line number and file name of
where CrashAndBurn is called!
*/
void CrashAndBurn(int linenum, char* filename, string in)          //kills the program
after printing the passed string to stdout
{
   printf("\nError! %d|\t%s\t|%s\n", linenum, in.c_str(), filename);
   exit(0);       //assert(1);
}


string itos(int in) //convert an int to a string
{
        char out[30];
        sprintf(out, "%d", in);
        return (string) out;
}
```

```
string dtos(float in)   //convert a double to a string  warning: will this produce a problem
since it uses float instead of Lf (long double)
{
	char out[30];
	sprintf(out, "%f", in);
	return (string) out;
}

string dtosp(float in)
{
	char out[30];
	sprintf(out, "%.2f", in);
	return (string) out;

}
string numToType(int typeNum)			//converts a type number (constant) into a
string with its name (to help report errors)
{
	if(typeNum == INTEGER) return "int";
	if(typeNum == FLOAT) return "float";
	if(typeNum == LIST) return "list";
	if(typeNum == STRING) return "string";
	if(typeNum == UNDEFINED) return "undefined";
	if(typeNum == UNKNOWN) return "unknown";
	if(typeNum == RES) return "res";
	if(typeNum == CAP) return "cap";
	if(typeNum == IND) return "ind";
	if(typeNum == VS) return "vs";
	if(typeNum == CS) return "cs";
	if(typeNum == NODE) return "node";
	if(typeNum == DEFAULT) return "default";
	if(typeNum == ERROR) return "error";
	if(typeNum == UNDECLARED) return "undeclared";
	if(typeNum == DIODE) return "diode";
	if(typeNum == BJT) return "bjt";
	if(typeNum == MOSFET) return "mosfet";
	else
	{
		CrashAndBurn(__LINE__, __FILE__, "Attempted to convert an invalid
type " + itos(typeNum) + " to a string in numToType()");
	}
	return "";
}
```

```
int typeToNum(string type)                  //converts a type number (constant) into a string
with its name (to help report errors)
{
	if(type == "integer") return INTEGER;
	if(type == "int") return INTEGER;
	if(type == "float") return FLOAT;
	if(type == "list") return LIST;
	if(type == "string") return STRING;
	if(type == "undefined") return UNDEFINED;
	if(type == "unknown") return UNKNOWN;
	if(type == "res") return RES;
	if(type == "cap") return CAP;
	if(type == "ind") return IND;
	if(type == "vs") return VS;
	if(type == "cs") return CS;
	if(type == "node") return NODE;
	if(type == "default") return DEFAULT;
	if(type == "error") return ERROR;
	if(type == "undeclared") return UNDECLARED;
	if(type == "diode") return DIODE;
	if(type == "bjt") return BJT;
	if(type == "mosfet") return MOSFET;
	else CrashAndBurn(__LINE__, __FILE__, "Attempted to convert an invalid
string '" + type + "' to a type in typeToNum()");
	return -99999;
}

int namesMade = 0;
string uniqueName()
{
	namesMade++;
	return "sym_var" + itos(namesMade);
}

/*--------------------------------------------
SMLNode class
-The class for circuit element nodes
--------------------------------------------*/
class SMLNode;
typedef vector <SMLNode *> NodeList;    //define a NodeList to be a vector of
SMLNode pointers
class SMLData;
class Obj;

class SMLNode
{
```

```
	public:
	NodeList connections;   //this is a list of pointers to all nodes that this node
connects to

	Obj *parent;			//the Object that this belongs to
	string name;			//used by ron in spice code generation

	SMLNode();

	SMLNode *operator [](int n);		//returns a pointer to the nth node that this
node connects to
	void connect(SMLNode *in);			//establishes a connection (in both
directions) between this node and the passed node
	int count(void);					//returns the number of nodes
that this node connects to
	void disconnect(int n);				//remove the nth node
connection
	int find(SMLNode *checkFor);
};

SMLNode::SMLNode()
{
	parent = NULL;
	name = "";
}

int SMLNode::find(SMLNode *checkFor)
{
	int num = count();
	for(int i = 0; i < num; i++)
	{
		if(connections[i] == checkFor) return i;
	}
	return -1;
}

SMLNode *SMLNode::operator[](int n)
{
	return connections[n];
}

int SMLNode::count(void)
{
	return connections.size();
}
```

```
void SMLNode::connect(SMLNode *in)
{
	if(find(in) == -1)
	{
		connections.push_back(in);
		in->connections.push_back(this);
	}
}

void SMLNode::disconnect(int n)
{
	if(n >= count() || n < 0) CrashAndBurn(__LINE__, __FILE__, "Tried to
disconnect node from an index out of range!");

	NodeList::iterator cur = connections.begin();
	cur += n;
	/*SMLNode *temp=connections[n];

	int count=temp->count();
	for(int k=0;k<count;k++)
	{

		if((*temp)[k]==this)
		{
			NodeList::iterator cur2 = temp->connections.begin();
			cur2+=k;
			temp->connections.erase(cur2);

		}


	}*/
	connections.erase(cur);
}


/*--------------------------------------------
SMLData class
-This class contains the data for SML objects
--------------------------------------------*/
class SMLData
{
	public:
	SMLNode *pos;		//the positive terminal of this circuit element
	SMLNode *neg;		//the negative terminal of this circuit element
	string argument;
```

```
	int touched;	//Ron uses this to check if it has been traversed before. Defaults to 
0

	SMLData();
	~SMLData();
	//may add more fields here as necessary (will be inhereted by all derived circuit 
elements)
};

SMLData::SMLData()
{
	pos = new SMLNode;
	neg = new SMLNode;
	pos->parent = NULL;
	neg->parent = NULL;
	argument = "";
	touched = 0;
}

SMLData::~SMLData()
{
	//delete pos; delete neg;
}

/*---------------------------------------------
Derived SML classes
---------------------------------------------*/
class Res : public SMLData
{
	public:
	double resistance;

	Res() : SMLData()
	{
		resistance = 0;
	}

	~Res()
	{
	//	delete pos; delete neg;
	}
};

class Cap : public SMLData
{
```

```
	public:
	double capacitance;
	double initial_voltage;

	Cap() : SMLData()
	{
		capacitance = 0;
		initial_voltage = 0;
	}

	~Cap()
	{
	//	delete pos; delete neg;
	}
};

class Ind : public SMLData
{
	public:
	double inductance;
	double initial_current;

	Ind() : SMLData()
	{
		inductance = 0;
		initial_current = 0;
	}

	~Ind()
	{
	//	delete pos; delete neg;
	}
};

class Vs : public SMLData
{
	public:
	double voltage;

	Vs() : SMLData()
	{
		voltage = 0;
	}

	~Vs()
	{
```

```
	//	delete pos; delete neg;
	}
};

class Cs : public SMLData
{
	public:
	double current;

	Cs() : SMLData()
	{
		current = 0;
	}

	~Cs()
	{
	//	delete pos; delete neg;
	}
};

class Diode : public SMLData
{
	public:

	Diode()  : SMLData()
	{
	}

	~Diode()
	{
	//	delete pos; delete neg;
	}
};

class Bjt : public SMLData  //bi-polar collector transistor
{
	public:
	SMLNode *base, *collector, *emitter;

	Bjt()  : SMLData()
	{
		base = new SMLNode;
		emitter = new SMLNode;
		collector = new SMLNode;

		base->parent = NULL;
```

```
		collector->parent = NULL;
		emitter->parent = NULL;
	}

	~Bjt()
	{
	//	delete pos; delete neg; delete base; delete collector; delete emitter;
	}
};

class Mosfet : public SMLData
{
	public:
	SMLNode *gate, *source, *drain;

	Mosfet()  : SMLData()
	{
		gate = new SMLNode;
		source = new SMLNode;
		drain = new SMLNode;

		gate->parent = NULL;
		source->parent = NULL;
		drain->parent = NULL;
	}

	~Mosfet()
	{
	//	delete pos; delete neg; delete gate; delete source; delete drain;
	}
};



/*---------------------------------------------
Object class
-This datatype acts as the wrapper for all variables
 in our language
----------------------------------------------*/
class SymbolTable;				//neccesary so that we can use this
class in our definition of the Obj class

class Obj
{
	public:
```

```
	char type;			//this is the type of the object (e.g. int, float, list). See
Global Constants list for values this can take.
	void *data;			//this is a pointer to the memory location where this object
stores its values

	string happy;	//for debug

	SymbolTable *table;			//points to the symbol table containing this
object

	Obj(int objType, double value);			//construct an Obj of type int
or float
	Obj(string value);					//construct an Obj of
type string
	Obj(int objType);					//construct an Obj of
type objType

	~Obj();
				//the destructor
};

typedef vector <Obj *> ObjList;	//define a ObjList to be a vector of Obj pointers

Obj::Obj(int dataType, double value)
{
	type = dataType;
	if(dataType == INTEGER)
	{
		data = new long;
		*((long*)data) = (long) value;
	}
	else if(dataType == FLOAT)
	{
		data = new double;
		*((double*)data) = value;
	}
	else CrashAndBurn(__LINE__, __FILE__, "Invalid Obj constructor! Type
unknown.");
}

Obj::Obj(string value)
{
	type = STRING;
	data = new string;
	*((string*)data) = value;
}
```

```
Obj::Obj(int dataType)
{
	type = dataType;
	if(dataType == INTEGER) data = new long;
	else if(dataType == FLOAT) data = new double;
	else if(dataType == LIST) data = new ObjList;
	else if(dataType == STRING) data = new string;
	else if(dataType == UNKNOWN) data = NULL;
	else if(dataType == UNDEFINED) data = NULL;
	else if(dataType == RES) data = new Res;
	else if(dataType == CAP) data = new Cap;
	else if(dataType == IND) data = new Ind;
	else if(dataType == VS) data = new Vs;
	else if(dataType == CS) data = new Cs;
	else if(dataType == NODE) data = new SMLNode;
	else if(dataType == DIODE) data = new Diode;
	else if(dataType == BJT)
	{
		data = new Bjt;
		((Bjt *) data)->base->parent = this;
		((Bjt *) data)->collector->parent = this;
		((Bjt *) data)->emitter->parent = this;
	}
	else if(dataType == MOSFET)
	{
		data = new Mosfet;
		((Mosfet *) data)->gate->parent = this;
		((Mosfet *) data)->source->parent = this;
		((Mosfet *) data)->drain->parent = this;
	}
	else CrashAndBurn(__LINE__, __FILE__, "Invalid Obj constructor! Type
unknown.");

	if(dataType == RES || dataType == CAP || dataType == IND || dataType == VS ||
dataType == CS || dataType == DIODE || dataType == BJT || dataType == MOSFET)
	{
		((SMLData*) data)->pos->parent = this;
		((SMLData*) data)->neg->parent = this;
	}
}


Obj::~Obj()					//deallocate the memory for this sucker
{
   if(type == INTEGER) delete ((long*) data);
```

```
	else if(type == FLOAT) delete ((double*) data);
	else if(type == LIST) delete ((ObjList*) data);			//doesn't
deallocate memory for each element of the list
	else if(type == STRING) delete ((string*) data);
	else if(type == RES) delete ((Res*) data);
	else if(type == CAP) delete ((Cap*) data);
	else if(type == IND) delete ((Ind*) data);
	else if(type == VS) delete ((Vs*) data);
	else if(type == CS) delete ((Cs*) data);
	else if(type == NODE) delete ((SMLNode*) data);
	else if(type == DIODE) delete ((Diode*) data);
	else if(type == BJT) delete ((Bjt*) data);
	else if(type == MOSFET) delete ((Mosfet*) data);
}


/*---------------------------------------------
SymbolTable class
-A symbol table stores all of the objects in
 a single scope.
---------------------------------------------*/

typedef map <string, Obj*> variableMap;		//define a VariableMap to be a map
storing object pointers and their key strings


class SymbolTable
{
	public:
	vector<Obj*> variables;				//stores the objects in
the table
	vector<string> names;				//stores the names of the
objects

	SymbolTable *parent;				//points to the symbol table
for the next innermost scope

	int inherits;					//set to 1 if this
symbolTable inherits the scope of its parent symbolTable (like a while loop)
							//and
set to 0 otherwise (for when we are in a function)

	int level;						//The number
of parents we must traverse to get to the outermost (global) scope.
							//So if
level = 0 then we are already in the global scope.
```

```
	/*
	AugmentedNode *returnTo;				//this stores the location in
the AST that we are currently at in this scope.
										//Used
for returning to the correct location in code after leaving an interior scope

	//should be uncommented when AugmentNode has been defined as a class.

	//Whenever we enter a new scope we need to set this appropriatly

	AugmentedNode *top;					//can be used to store
where execution should move to after a continue command is called within a loop scope
									//but also, by
storing the node with the definition of the function who's scope we are in
									//we gain
access to the names of the function's parameters. These will be needed in order to
construct
									//our
ParamNodes
	*/

	ObjList stack;				//stores the stack variables...
in this case it is just a list of pointers to objects.
								//notice that
modifying our stack variables will change the value they had in the calling scope
								//these are
referenced by their location in this vector in the ParamNode nodes.

	SymbolTable(SymbolTable *tableParent, int inherits);		//constructor

	int size(void);

	void add(Obj *item, string name);		//when a variable is added we must
provide it with a name. If we do not name it

	//(provide an empty string) then a unique name will be assigned automatically
	void setObj(Obj *obj, int pos, int height);

	int getLocal(string name);
	//gets the object associated with a given name (only trying in the current scope,
not any parent scopes)
	void get(string name, int &pos, int &height);			//gets the
object associated with a given name trying all relavent scopes
	void realGet(string name, int &pos, int &height);			//gets the
object associated with a given name trying all relavent scopes
```

```
	Obj* getObj(int pos, int height);
	//gets the nth object (0-based access)
	string getName(int pos, int height);					//gets
the nth object name (0-based access)
	int getLocal(Obj *toGetNameOf);
	//gets the name of a given object (only trying in current scope, not any parent
scopes)
	void get(Obj *toGetNameOf, int &pos, int &height);				//gets
the name of a given object trying in all relavent scopes
	void realGet(Obj *toGetNameOf, int &pos, int &height);			//gets the
name of a given object trying in all relavent scopes
};

//constructs a new symbol table. If tableParent is null then assumes this is the global
symbol table
SymbolTable::SymbolTable(SymbolTable *tableParent, int inheretsParentScope)
{
	parent = tableParent;
	inherits = inheretsParentScope;
	if(tableParent == NULL)
	{
		level = 0;
		inherits = 0;
	}
	else level = tableParent->level + 1;
}

int SymbolTable::size(void)
{
	return variables.size();
}

void SymbolTable::setObj(Obj *obj, int pos, int height)
{
	SymbolTable *cur = this;
  while(height > 0)
  {
		cur = cur->parent;
		height--;
  }

  if(pos >= (int)  cur->variables.size()) CrashAndBurn(__LINE__, __FILE__, "Tried to
access an element with index greater than or equal to the number of symbol table
elements!");

  if(pos < 0 ) {
```

```
		CrashAndBurn(__LINE__, __FILE__, "Tried to access an element of the 
symbol table with a negative index!");
   }

	cur->variables[pos] = obj;
}

void SymbolTable::add(Obj *item, string name)	//adds this object to the symbol table
{
   if(name.length() == 0)			//check if they did not enter a name for this variable 
(if not, we must make one up)
	{
		char newName[12];
		sprintf(newName, "const%d_%d", level, (int)variables.size() );
		name = newName;
	}
	else	//check if this variable name is already in the symbol table
	{
		int loc = getLocal(name);
		if(loc != -1) CrashAndBurn(__LINE__, __FILE__, "The variable '" + 
name + "' already exists in symbol table!");
	}

	variables.push_back(item);
	names.push_back(name);
	if(item != NULL) item->table = this;			//make the object point to this 
symbol table
}


int SymbolTable::getLocal(string name)	//gets an object from the symbol table using 
its name
{
   for(unsigned int i = 0; i < names.size(); i++)
  {
		if(names[i] == name) return i;
  }
  return -1;
}

void SymbolTable::get(string name, int &pos, int &height)	//gets an object from the 
symbol table using its name
{
	height = 0;
	realGet(name, pos, height);
}
```

```
void SymbolTable::realGet(string name, int &pos, int &height)       //gets an object from
the symbol table using its name
{
        pos = getLocal(name);
        if(pos != -1) return;
        if(level == 0 || inherits == 0)
        {
                pos = -1; height = -1; return;
        }
        height = height + 1;
        parent->realGet(name, pos, height);
}



int SymbolTable::getLocal(Obj *toGetNameOf)                                    //get the name
of the specified object in the symbol table
{
   for(unsigned int i = 0; i < names.size(); i++)
  {
                if(variables[i] == toGetNameOf) return i;
  }
  return -1;
}

void SymbolTable::get(Obj *toGetNameOf, int &pos, int &height)       //gets an object
from the symbol table using its name
{
        height = 0;
        realGet(toGetNameOf, pos, height);
}

void SymbolTable::realGet(Obj *toGetNameOf, int &pos, int &height)       //get the
name of the specified object in the symbol table
{
        pos = getLocal(toGetNameOf);
        if(pos != -1) return;
        if(level == 0 || inherits == 0)
        {
                pos = -1; height = -1; return;
        }
        height = height + 1;
        parent->realGet(toGetNameOf, pos, height);
}
```

```
Obj* SymbolTable::getObj(int pos, int height)                    //gets the nth object (0-based access) from the symbol table. The algorithm for access in this way is innefficient but should do for now
{

  SymbolTable *cur = this;
  while(height > 0)
  {
                cur = cur->parent;
                height--;
  }

        if(pos >= (int) cur->variables.size()) CrashAndBurn(__LINE__, __FILE__, "Tried to access an element with index greater than or equal to the number of symbol table elements!");
  if(pos < 0 ) CrashAndBurn(__LINE__, __FILE__, "Tried to access an element of the symbol table with a negative index!");

  return cur->variables[pos];
}

string SymbolTable::getName(int pos, int height)                 //gets the nth object (0-based access) from the symbol table. The algorithm for access in this way is innefficient but should do for now
{
  SymbolTable *cur = this;
  while(height > 0)
  {
                cur = cur->parent;
                height--;
  }

   if(pos >= (int) cur->variables.size()) CrashAndBurn(__LINE__, __FILE__, "Tried to access an element with index greater than or equal to the number of symbol table elements!");
  if(pos < 0 ) CrashAndBurn(__LINE__, __FILE__, "Tried to access an element of the symbol table with a negative index!");

  return cur->names[pos];
}




#endif
```

8.6 Main Program

```
//Collaborated Effort
#pragma warning(disable:4786)
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>

#include "SML.h"
#include "Lex.hpp"
#include "Pars.hpp"
#include "Walk.hpp"
#include "LexTokenTypes.hpp"

using namespace std;

ANTLR_USING_NAMESPACE(std)
ANTLR_USING_NAMESPACE(antlr)

SymbolTable globalScope(NULL, 0);  //the global symbol table for our program
SMLNode ground;                                    //the ground node
string spiceInjection = "";

#include "StaticSemantics.h"
#include "SMLWalker.h"
#include "codeGen.h"

int main(int argc, char **argv)
{
	string outputFile = "out.sp";
	FILE *in_file;
	string theFile;

	//printf("\nparams: %d\n", argc);
	//printf("\nparam 1: %s\n", argv[0]);

	try
	{
		if(argc <= 1)   //if no input file name was passed
		{
				in_file = fopen("input", "rb");
				if(in_file == NULL)
				{
					in_file = fopen("../input", "rb");
```

```
				if(in_file == NULL)
				{
					printf("input file not found! Aborting.\n");
					exit(0);
				}
				else theFile = "../input";
			}
			else theFile = "input";
	}
	else
	{
		in_file = fopen(argv[1], "rb");
		if(in_file == NULL)
		{
			string compound = "../";
			string store = argv[1];
			compound += store;
			in_file = fopen(compound.c_str(), "rb");
			if(in_file == NULL)
			{
				printf("input file '%s' not found! Aborting.\n",
argv[1]);

				exit(0);
			}
			else theFile = compound;
		}
		else theFile = argv[1];
	}

	if(argc >= 3) outputFile = argv[2]; //set the output file

	char curChar = 100;
	istringstream input;
	string data;

	printf("\nProcessing file '%s'\n\n", theFile.c_str());

	while(curChar != EOF)
	{
		curChar = getc(in_file);
		if(curChar == EOF) break;
		data += curChar;
	}

	data += "\n";					//put an end of line at the end
so that the end of file is on its own line!
```

```
printf("%s\n", data.c_str());

input.str(data);

Lex lexer(input);
//Lex lexer(in_file);

Pars parser(lexer);
// set up the ast factory to use a custom AST type per default
// note that here the Ref prefix for the reference counter is
// strippped off.
ASTFactory ast_factory("MyAST", MyAST::factory);

// let the parser add it's stuff to the factory...
parser.initializeASTFactory(ast_factory);
parser.setASTFactory(&ast_factory);

parser.program();

//run static semantics
MakeLoveToThisTree(RefMyAST(parser.getAST()));

printf("\n");
if(GetSemanticErrors().length() == 0) ;//printf("No errors!");
else
{
        printf("%s", GetSemanticErrors().c_str());
        exit(1);
}

printf("--------------------------------\n\n");

//prints the tree
//cout << parser.getAST()->toStringList() << endl;

//added by Rob
if (GetSemanticErrors().length() == 0)
{
        //printf("No semantic errors, start finishoffthistree \n");
        //trav_tree(RefMyAST(parser.getAST()), 0);
        FinishOffThisTree(RefMyAST(parser.getAST()));
}

printf("\n\n");

trav_tree(RefMyAST(parser.getAST()), 0);
```

```
		printf("\n\n");

		nodeCruncher(&ground); //crunch ground first
		nodeWalker();  //walk others and crunch
		//spiceInjection = ".op \n.dc #Vin 1 10 1\n.print dc v(#Vin.pos ) v(#r1.pos
,#r1.neg )\n";
		spicify(theFile, outputFile, spiceInjection); //generate code




		Walk walker;
		// these two are  not really necessary
		// since we're not building an AST
		walker.initializeASTFactory(ast_factory);
		walker.setASTFactory(&ast_factory);

		//walker.block(RefMyAST(parser.getAST()));		// walk tree
		//cout << "done walking" << endl;

		fclose(in_file);

		printf("\nGenerated file '%s'\n", outputFile.c_str());
	}
	catch( ANTLRException& e )
	{
		cerr << "exception: " << e.getMessage() << endl;
		return -1;
	}
	catch( exception& e )
	{
		cerr << "exception: " << e.what() << endl;
		return -1;
	}
	return 0;
}
```

8.7 Tree Node Class

```
//Spencer Greenberg
#ifndef __MY_AST_H__
# define __MY_AST_H__

#include <antlr/CommonAST.hpp>
```

```
#define DEFAULT -3		//the default type for a node. This had better be unique from the other types and match the number for DEFAULT in SML.h!

class Obj;

//#ifndef __SML_h__
//#include "SML.h"
//#endif

class SymbolTable;

class MyAST;

typedef ANTLR_USE_NAMESPACE(antlr)ASTRefCount<MyAST> RefMyAST;

/** Custom AST class that adds line numbers to the AST nodes.
 * easily extended with columns. Filenames will take more work since
 * you'll need a custom token class as well (one that contains the
 * filename)
 */
class MyAST : public ANTLR_USE_NAMESPACE(antlr)CommonAST {
public:
	int const evaltype(void)
	{
		return evaluationtype;
	}

	void evaltype(int in)
	{
		evaluationtype = in;
	}

	void breakNode(MyAST *node)
	{
		breakTo = node;
	}

	MyAST *breakNode()
	{
		return breakTo;
	}

  // Obj* objectPtr(void)
  // {
  //		return table()->getObj(tablePosition);
  // }
```

```
        void setPosition(int pos, int height)
  {
                tablePosition = pos;
                tableHeight = height;
  }

  void getPosition(int &pos, int &height)
  {
                pos = tablePosition;
                height = tableHeight;
  }

  SymbolTable* table(void)
  {
                return symtable;
  }

 void table(SymbolTable *in)
  {
                symtable = in;
  }

  // copy constructor
  MyAST( const MyAST& other )
        : CommonAST(other)
        , line(other.line)
        , tablePosition(other.tablePosition)
        , evaluationtype(other.evaluationtype)
        , symtable(other.symtable)
        , breakTo(other.breakTo)
        , returnTo(other.returnTo)
        {
        }
  // Default constructor
  MyAST( void ) : CommonAST(), line(0)
  {
                breakTo = NULL; returnTo = NULL; evaltype(DEFAULT); setPosition(-
1,-1); symtable = NULL;
  }
  virtual ~MyAST( void ) {}
  // get the line number of the node (or try to derive it from the child node
  virtual int getLine( void ) const
  {
    // most of the time the line number is not set if the node is a
    // imaginary one. Usually this means it has a child. Refer to the
```

```
    // child line number. Of course this could be extended a bit.
    // based on an example by Peter Morling.
    if ( line != 0 )
      return line;
    if( getFirstChild() )
      return ( RefMyAST(getFirstChild())->getLine() );
    return 0;
  }
  virtual void setLine( int l )
  {
    line = l;
  }
        /** the initialize methods are called by the tree building constructs
   * depending on which version is called the line number is filled in.
   * e.g. a bit depending on how the node is constructed it will have the
   * line number filled in or not (imaginary nodes!).
   */
  virtual void initialize(int t, const ANTLR_USE_NAMESPACE(std)string& txt)
  {
    CommonAST::initialize(t,txt);
    line = 0;
          breakTo = NULL; returnTo = NULL; evaltype(DEFAULT); setPosition(-1,-1);
symtable = NULL;
  }
  virtual void initialize( ANTLR_USE_NAMESPACE(antlr)RefToken t )
  {
    CommonAST::initialize(t);
    line = t->getLine();
          breakTo = NULL; returnTo = NULL; evaltype(DEFAULT); setPosition(-1,-1);
symtable = NULL;
  }
  virtual void initialize( RefMyAST ast )
  {
    CommonAST::initialize(ANTLR_USE_NAMESPACE(antlr)RefAST(ast));
    line = ast->getLine();
          breakTo = NULL; returnTo = NULL; evaltype(DEFAULT); setPosition(-1,-1);
symtable = NULL;
  }
  // for convenience will also work without
  void addChild( RefMyAST c )
  {
    BaseAST::addChild( ANTLR_USE_NAMESPACE(antlr)RefAST(c) );
  }
  // for convenience will also work without
  void setNextSibling( RefMyAST c )
  {
```

```
    BaseAST::setNextSibling( ANTLR_USE_NAMESPACE(antlr)RefAST(c) );
  }
  // provide a clone of the node (no sibling/child pointers are copied)
  virtual ANTLR_USE_NAMESPACE(antlr)RefAST clone( void )
  {
    return ANTLR_USE_NAMESPACE(antlr)RefAST(new MyAST(*this));
  }
  static ANTLR_USE_NAMESPACE(antlr)RefAST factory( void )
  {
    return ANTLR_USE_NAMESPACE(antlr)RefAST(RefMyAST(new MyAST()));
  }
public:
  int line;                                       //the line number this node was constructed
at
        /*the reason this is a pointer to a pointer is rather subtle. The symboltable binds
Obj pointers to names. Each time a local variable is instantiated however
         we need to bind this Obj pointer to a new peice of memory. Thus, here we have a
pointer to the Obj pointer that resides in the symbol table so that we
         can modify where it points*/
        int tablePosition;                                  //the position in the symbol
table where the object this node is bound to resides
   int tableHeight;                                   //the number of scopes up that the
table the variable resides in is above from this scope
        int evaluationtype;                       //the object type that this node will evaluate
to according to static semantics. Access this by calling "evalType()"!

  SymbolTable *symtable;               //the innermost symbol table containing this node
  MyAST *breakTo;                            //for when we are inside a while loop, points
back to the node of the actual while loop so that we know where to go on "break" and
"continue"
  MyAST *returnTo;                           //for when we are inside a function, points
back to the node of the actual function call so that we know where to go on "return"
};
#endif
```