

dots

## A Graph Language

Hosanna Fuller (hjf2106) — Manager  
Yumeng Liao (yl2908) — Tester  
Adam Incera (aji2112) — C Guru  
Rachel Gordon (rcg2130) — Language Guru

December 2015

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Summary . . . . .	3
1.2	Use Cases . . . . .	3
<b>2</b>	<b>Setup</b>	<b>3</b>
2.1	Installation . . . . .	4
2.2	Running the compiler . . . . .	4
<b>3</b>	<b>Basic d.o.t.s. Tutorial</b>	<b>4</b>
<b>4</b>	<b>Language Reference Manual</b>	<b>7</b>
4.1	Introduction . . . . .	7
4.2	Lexical Conventions . . . . .	7
4.2.1	Comments . . . . .	7
4.2.2	Identifiers . . . . .	8
4.3	Keywords . . . . .	8
4.4	Built-in Data Types . . . . .	8
4.4.1	Nums and Strings . . . . .	9
4.4.2	Node and Graph Objects . . . . .	10
4.4.3	Collections – Dicts and Lists . . . . .	11
4.5	Special Operators . . . . .	12
4.5.1	Node Operators . . . . .	12
4.5.2	Graph Operators . . . . .	14
4.6	Built-in Functions . . . . .	16
4.7	Control Flow . . . . .	17
4.7.1	Logical Expressions . . . . .	17
4.7.2	If Statements . . . . .	17
4.7.3	For Loops . . . . .	18
4.7.4	While . . . . .	18
4.8	User-defined Functions . . . . .	18
<b>5</b>	<b>Example Code</b>	<b>19</b>
5.1	Hello World . . . . .	19
5.2	Breadth-First Search . . . . .	20
5.3	User-Defined Functions Example . . . . .	21
5.4	Dijkstra’s Algorithm . . . . .	24
<b>6</b>	<b>Project Development and Lifetime</b>	<b>26</b>
6.1	Project Plan . . . . .	26
6.2	Planning and Synchronization . . . . .	26
6.3	Project Development . . . . .	26
6.4	Development Environment . . . . .	27
6.5	Style Guide . . . . .	27
6.6	Project Log . . . . .	27
6.7	Roles and Responsibilities . . . . .	29
<b>7</b>	<b>Compiler Design</b>	<b>29</b>
7.1	Input Processing . . . . .	30
7.2	DOTS Handling . . . . .	31
7.3	C Handling . . . . .	31
<b>8</b>	<b>C Library</b>	<b>32</b>

<b>9</b>	<b>Testing</b>	<b>33</b>
9.1	Test Suites . . . . .	33
9.2	Test Automation . . . . .	33
9.3	Source to Target . . . . .	33
<b>10</b>	<b>Lessons Learned</b>	<b>40</b>
10.1	Hosanna’s Advice . . . . .	40
10.2	Adam’s Advice . . . . .	40
10.3	Rachel’s Advice . . . . .	41
10.4	Yumeng’s Advice . . . . .	41
	<b>Appendices</b>	<b>41</b>
<b>A</b>	<b>OCaml Compiler Files</b>	<b>41</b>
A.1	scanner.mll . . . . .	41
A.2	parser.mly . . . . .	42
A.3	ast.ml . . . . .	48
A.4	Sast.ml . . . . .	51
A.5	typeConverter.ml . . . . .	52
A.6	analyzer.ml . . . . .	66
A.7	translate.ml . . . . .	98
A.8	compile.ml . . . . .	102
<b>B</b>	<b>C Library Files</b>	<b>104</b>
B.1	node.h . . . . .	104
B.2	node.c . . . . .	105
B.3	graph.h . . . . .	107
B.4	graph.c . . . . .	108
B.5	list.h . . . . .	111
B.6	list.c . . . . .	114
B.7	main.c . . . . .	122
B.8	test.c . . . . .	124
B.9	snippets.c . . . . .	126
<b>C</b>	<b>Makefiles</b>	<b>131</b>
C.1	src/Makefile . . . . .	131
C.2	src/clib/Makefile . . . . .	132
<b>D</b>	<b>Test Suite</b>	<b>133</b>
D.1	runtest.py . . . . .	133
D.2	testall.py . . . . .	137
<b>E</b>	<b>Example Code Compiled C Programs</b>	<b>138</b>
E.1	hello-world.dots.c . . . . .	138
E.2	sample01.dots.c . . . . .	139
E.3	bst.dots.c . . . . .	147
E.4	tutorial.dots.c . . . . .	152
<b>F</b>	<b>Git Commit History</b>	<b>157</b>

# 1 Introduction

Graphs are a powerful and versatile data structure used across many languages to help visually organize and manipulate data. Many languages do not provide out-of-the-box support for data structures and methods useful in solving graph problems. Because of this, programmers end up spending unnecessary time and energy implementing these critical structures themselves. As a result, graph implementations often widely vary in efficiency and modularity. The goal of d.o.t.s. is to provide an out-of-the-box graph framework so that users can focus on creating the algorithms needed to solve their problems, rather than getting bogged down in implementation issues. With d.o.t.s. users can comprehensively solve a wide variety of graph-based problems. Some example problems include: expressing network relationships such as a series of interconnected routers with edge costs, representing decision trees in probability, and running analyses on propositional models.

## 1.1 Summary

d.o.t.s. is a scripting language, much like python, with the goal to take the headache out of implementing and manipulating graphs and their data. Many fields of study in computer science such as Network Routing and Natural Language Processing depend on graphs to solve complex algorithmic issues. As a result, programmers spend unnecessarily long amounts of time just building up the graph structure they need to represent their network or language heuristic. Similarly, even when they do have their graph framework implemented, operations for manipulating the data are often syntactically confusing or take several lines of code to do a simple operation. For example adding a node to a graph may entail scanning the entire graph for duplicates, creating the new node, and appending it to the end of a list. Our hope with d.o.t.s. is that programmers find it easy to use so they can focus on the real algorithmic problems they are trying to solve instead of getting bogged down in implementation details.

## 1.2 Use Cases

Instead of spending hours implementing standard data structures like graphs, programmers can focus on implementing complex algorithms. d.o.t.s. offers users the opportunity to work with graphs in a simple, intuitive manner. Here are some example use cases below:

1. Build graphs to help visualize data. The graph structure allows users to easily create nodes and assign them data. Similarly you can iterate through all the graphs data easily so displaying your graph is as simple as writing a pretty printer.
2. Social Network Analysis – For instance, if you wanted to have a program that stored a graphical representation of all your Facebook friends you could easily create that graph and simply add or subtract friends from that graph without ever having to write a custom function to do so. d.o.t.s. allows users to maintain graph structures for storing data easily.
3. Graph Algorithms / Routing Algorithms – Users can quickly implement complex graphing algorithms like Dijkstra’s with d.o.t.s.. We were able to create Dijkstra’s example in 30 lines of code. If you were to try and implement Dijkstra’s in java it would be at least 500 lines.
4. Propositional Models – propositional models can be represented as a graph, where edge weights are probabilities conditioned on the parent nodes. Using d.o.t.s., a programmer could run statistical analyses on a graph.

# 2 Setup

The following are instructions for getting the d.o.t.s. compiler up and running.

## 2.1 Installation

*Note:* all commands are run from the `src` directory of the compiler.

1. Unpack the d.o.t.s. compiler tar ball.
2. Add the path to the `clib` directory to the `DOTS` environment variable.

```
> export DOTS=/path/to/dots-compiler/src/clib
```

3. Make the C library.

```
> make setup
```

4. Make the d.o.t.s. compiler.

```
> make
```

## 2.2 Running the compiler

Compiling d.o.t.s. code consists of two pieces: compiling the `.dots` source file into a `.c` file, and compiling the `.c` file into a binary executable.

Both of these two steps are wrapped up in the `gdc` executable file to allow for easy compilation of d.o.t.s. programs. The script takes two arguments: the `.dots` file to compile, and the name of the binary file to compile to. The first argument is required, the second argument is optional.

Example:

Compile the file to the default executable – `exec`

```
> ./gdc dots/src/sample-code/sample01.dots  
> ./exec
```

Compile the file to a specific executable name

```
> ./gdc dots/src/sample-code/sample01.dots example-file  
> ./example-file
```

## 3 Basic d.o.t.s. Tutorial

In this section, we walk you through creating your first d.o.t.s. program.

### Step 1: Creating Your d.o.t.s. source file

Create a new file called `tutorial.dots` in any directory, and open it up.

### Step 2: Declaring Primitive Variables

In d.o.t.s., you declare variables by writing a datatype, followed by an alphanumeric string that begins with an alphabetic letter. The basic data types are: `num`, `string`, `bool`.

```
1 num x;  
2 string s;
```

### Step 3: Assigning Variables

Now let's give each of those variables a value.

```
1 x = 5.3;  
2 s = "This is a d.o.t.s. program.";
```

### Step 4: Declaring Nodes

The basic element of a graphs in d.o.t.s. are objects of the type: `node`. When declaring a node, you have the option of assigning the node's value with a string in parentheses.

```
1 node n1;  
2 node n2("florida");
```

### Step 5: Declaring and Assigning Graphs

The main data type underlying d.o.t.s. is the `graph` type. A graph represents a collection of node objects. Graphs can be assigned to an expression that evaluates to a graph object. Let's create a graph consisting of the two nodes we just declared.

```
1 graph g;  
2 g = n1 + n2;
```

### Step 6: Assigning Edges and Weights

Now we have a graph consisting of the two nodes `n1` and `n2`. But neither of these nodes have incoming or outgoing edges. Let's spice up the graph a little by adding in some edges.

```
1 n1 -->[2.3] n2;  
2 n2 --> n1;
```

Now `n1` has an edge leading to `n2` with weight 2.3, and `n2` has an edge leading to `n1` with weight 0.

### Step 7: Declaring Complex Variables

The collection data types are: `list<elem_type>` and `dict<key_type, val_type>`, where `elem_type`, `key_type`, and `val_type` are all d.o.t.s. data types. These data types are used to keep track of multiple values at once.

Lists can be declared and assigned on the same line, and can be assigned to list literals. Let's create a list of our nodes.

```
1 list<node> node_list = [n1, n2];
```

Currently, `dict` literals are not supported, so let's create a `dict` and assign each element separately.

```
1 dict<string, num> lang_prob;  
2 lang_prob["e"] = 12.7;  
3 lang_prob["a"] = 8.17;  
4 lang_prob["d"] = 4.25;
```

### Step 8: Printing Results

Now let's print out some of what we've done. `d.o.t.s.` has a `print` function that can take a comma-separated list of values of any data type<sup>1</sup>. To print out the nodes inside the graph we created, we'll use a `for` loop to iterate over the graph's nodes.

```
1 print(x, "\n", s, "\n", n1, "\n", n2, "\n");
2
3 for (elem in g) {
4     print(elem, "\n");
5 }
6
7 print(node_list, "\n", lang_prob, "\n");
```

### Step 9: Putting It All Together

Your `tutorial.dots` file should now look like this:

```
1 num x;
2 string s;
3
4 x = 5.3;
5 s = "This is a d.o.t.s. program.";
6
7 node n1;
8 node n2("florida");
9
10 graph g;
11 g = n1 + n2;
12
13 n1 -->[2.3] n2;
14 n2 --> n1;
15
16 list<node> node_list = [n1, n2];
17 dict<string, num> lang_prob;
18 lang_prob["e"] = 12.7;
19 lang_prob["a"] = 8.17;
20 lang_prob["d"] = 4.25;
21
22 print(x, "\n", s, "\n", n1, "\n", n2, "\n");
23
24 for (elem in g) {
25     print(elem, "\n");
26 }
27
28 print(node_list, "\n", lang_prob, "\n");
```

Listing 1: `tutorial.dots`

### Step 10: Compiling `tutorial.dots`

*Note: this step assumes you've already done the installation described in Section 2.1*

---

<sup>1</sup>directly printing graph objects not yet supported

Navigate to your `/path/to/dots/compiler/src` directory. In your terminal enter:

```
> ./gdc /path/to/tutorial.dots
```

You should now see a file named `exec` in the `src` directory.<sup>2</sup> Run it by doing:

```
> ./exec
```

Your output should look something like this:

```
1 5.300
2 This is a d.o.t.s. program.
3 N--633324960("")
4 N--633324896("florida")
5 N--633324960("")
6 N--633324896("florida")
7 [N--633324960(""), N--633324896("florida"), ]
8 {a: 8.170, d: 4.250, e: 12.700, }
```

## 4 Language Reference Manual

### 4.1 Introduction

The strict typing and control flow in d.o.t.s. is reminiscent of C and Java, but overall the language is intended to be used more as a scripting language, where the user builds their graphs quickly using the intuitive node and edge operators and then performs operations on the structures.

The d.o.t.s. compiler compiles code written in d.o.t.s. into C, and then uses the GNU C Compiler to build binary executables.

*Note:* In the following sections, the word “graph” is sometimes used to denote a data structure and sometimes to denote the abstract structure from computer science and mathematics:

A graph data structure consists of a finite (and possibly mutable) set of nodes or vertices, together with a set of ordered pairs of these nodes (or, in some cases, a set of unordered pairs). These pairs are known as edges or arcs. As in mathematics, an edge (x,y) is said to point or go from x to y. The nodes may be part of the graph structure, or may be external entities represented by integer indices or references.

A graph data structure may also associate to each edge some edge value, such as a symbolic label or a numeric attribute (cost, capacity, length, etc.).<sup>3</sup>

For the sake of clarity, from this point forward we will refer to the language-specific data structure using the lowercase “graph” and the mathematical concept using the uppercase “Graph.”

### 4.2 Lexical Conventions

#### 4.2.1 Comments

A comment is a sequence of characters ignored by the compiler in a d.o.t.s. script.

<sup>2</sup>You can see the C code that’s code that’s compiled in Appendix E.4

<sup>3</sup>[https://en.wikipedia.org/wiki/Graph\\_\(abstract\\_data\\_type\)](https://en.wikipedia.org/wiki/Graph_(abstract_data_type))



Syntax	Comment Style
<code>\*</code> <i>code</i> <code>*\</code>	multi-line comment
<code>#</code>	single-line comment

Table 1: Comment Styles

### 4.2.2 Identifiers

Variable and function names must begin with an alphabetic character and contain only alphanumeric characters.

### 4.3 Keywords

Table 2 shows the list of reserved words in d.o.t.s.. Keywords have special meaning in d.o.t.s. and cannot be overwritten.

if
else
true
false
in
for
while
return
bool
num
string
node
graph
list
dict

Table 2: Keywords

### 4.4 Built-in Data Types

d.o.t.s. provides three different types of variables: basic types, built-in types, and collections. The three basic types are `num` `bool` `string`. Each of these basic types have raw value representations, which can be used with no prior declaration of variables. Such values can also be assigned as the values of variables.

The two built-in data types, `node` and `graph`, provide the basis for algorithms written in d.o.t.s.. The built-in collection types are `list` and `dict`, and can be used to contain any of the other two kinds of types.

d.o.t.s. is a strictly-typed language, meaning that the types of all variables must be declared at the same time that the variable is declared.

Data Type	Fields
num	
string	
bool	
list	
dict	
node	value, in, out
graph	vertices

Table 3: Built-in Data Types

#### 4.4.1 Nums and Strings

Category	Data Type	Operator	Explanation
comparison	num, string	<, >, <=, >=, !=, ==	Operate in the same way as languages such as C/C++, with the exception that string equality compares the <i>value</i> contained in the string.
computation	num	+, -, *, /, %	Operate in the same way as languages such as C/C++.
concatenation	string	+	String concatenation operator

Table 4: Num and String Operators

Both nums and strings can be assigned on the same line as their declaration.

In addition to literal num values, d.o.t.s. provides an infinity value for the num type: `INF`. The operators perform a little differently for these values.<sup>4</sup> As the primary use of infinity in graph problems is to define edge weight and not to perform mathematical calculations, the computation operators cause an exception whenever infinity is an operand.

For comparison operators, `INF` is greater than all non-null non-infinity values and equal to other infinity values. Defining the comparison operators for `INF` values allows them to be used as valid edge weights, which can be useful for graph problems.

```

1 num x = 12.0;
2 num y = 6 * 2.0;
3 num z = 6.5;
4 x == y; # returns true
5 y < z; # returns false
6
7
8 string phrase = "Hello" + " " + "World"; # phrase = "Hello World"
9 string a = "cat";
10 string b = "bear";
11 a < b; # returns true;
12
13 num i = INF;
14 y > INF; # returns false
15 num h = 2 + i; # exception

```

Listing 2: Demonstration of “num” and “string” types.

<sup>4</sup>Note: We didn’t have time to implement special handling of `INF`.

## 4.4.2 Node and Graph Objects

The data types which underpin d.o.t.s. and give it its advantage in the Graph domain over languages such as C are `node` and `graph`. From the get-go any programmer using d.o.t.s. can use these data types to quickly build Graphs without the need to waste time creating these data structures from scratch.

A `node` object represents a single vertex in a Graph, whereas a `graph` object represents a collection of graphs (which can be empty). Conceptually: A node is a graph, but a graph is not a node.

Recursive definition of graph objects:

- An empty graph is a graph.
- A node is a graph.
- A graph added to a graph is a graph.

A graph contains only the field `nodes`, which is a list of all node objects contained within the graph.

A node contains the fields `val`, `in`, `out`. Internally, a node is uniquely identified by its address in memory, but this distinction is visible to the programmer only when they print a node. The `val` field is a string object, and simply represents some value that the node contains. One possible use of the value field is to allow users to assign a more semantic meaning to nodes (ex. setting the value to the name of a city). The `in` field is a dict mapping nodes that the current node has edges into to weights. Similarly, the `out` field is a dict mapping nodes that have edges into the current node to weights. The keys of the two dicts are nodes. The `in` and `out` fields of a node can only be accessed by calling the `ine()` and `oute()` member functions of `node`. An example of accessing the `in` and `out` dicts of a node can be seen in Listing 5.

Nodes can be declared in two different ways. In the first, the variable can simply be declared with the `node` keyword and a variable name. This creates a basic node with an empty value, `in` list, and `out` list. In the second manner, a node can be declared by giving it an initial value inside parentheses after the variable name (as seen in Listing 3).

Graphs can be declared with the keyword `graph`. There is also a special graph-creation syntax that can *only* be used at declaration time of a graph object (as seen in Listing 3). This special syntax consists of a comma-separated list of node operations (an example of this syntax can be found in Listing 5). Each node referenced in this type of assignment must have been previously declared. This special syntax adds the nodes within the declaration to the declared graph and adds the specified edges to each of the nodes.

After declaration, as with all other types, nodes and graphs can be assigned any expression that evaluates to something of the same type.

```
1 node x;
2 node y("nyc");
3
4 graph g1;
5 graph g2;
6 g2 = g1;
7
8 graph g3 = {
9     x --> [2.3] y,
10    y --> x
11 }; # special graph assignment syntax only available at declaration time
```

Listing 3: Declaration of “node” and “graph” objects.

### 4.4.3 Collections – Dicts and Lists

Lists are declared using the keyword `list` and an indicator of the type of the list, as all objects in a list must be of the same type. Lists can be assigned by putting a comma-separated list of objects inside brackets, or can be filled using list functions. Lists are allowed to be assigned at declaration time.

Dictionary objects in d.o.t.s. represent mappings from objects to objects; a key can be any type for which the comparison operator “==” has been defined, and all keys in a given dict must be of the same type. Similarly, all values in a given dict must be of the same type. Dict objects are declared in a similar manner to lists, using the keyword `dict` and an indicator of the type that the dict maps from and to. Dicts must be assigned after declaration, and can be assigned by putting a comma-separated list of `key:value` pairs inside curly braces.

Operation	Syntax	Explanation
<u>List Ops</u>		
random access	<code>listVar[index]</code>	returns the element at the index
enqueue	<code>listvar.enqueue(elemVar)</code>	appends the given variable to the end of the list
dequeue	<code>listvar.dequeue()</code>	removes and returns the front element of the list
push	<code>listvar.enqueue(elemVar)</code>	appends the given variable to the front of the list
pop	<code>listvar.dequeue()</code>	removes and returns the front element of the list
dequeue	<code>listvar.dequeue()</code>	removes and returns the front element of the list
peek	<code>listvar.enqueue(elemVar)</code>	returns the variable at the front of the list
<u>Dict Ops</u>		
value access	<code>dictVar[key]</code>	returns the value at the given key. <i>key</i> can be a raw value or variable.
value assignment	<code>dictVar[key] = newValue</code>	if <i>key</i> already exists in the dict, overrides the old value with <i>newValue</i> , else adds the ( <i>key,newValue</i> ) pair to the dict.
element deletion	<code>dictVar.remove(keyVal)</code>	removes the (key, value) pair with key <i>keyVal</i> from the dict.

Table 5: List and Dict Operations

```

1 list<num> numList = [1, 2, 3];
2
3 list<string> strList = ["Hello"];
4 strList.enqueue(" ");
5 strList.enqueue("World"); # strList now equals ["Hello", " ", "World"];
6 string s = strList.peek(); # s = "World"
7 strList.dequeue(); # strList = ["Hello", " "]
8
9 dict<string, num> numDict;
10 numDict["one"] = 1;
11 numDict["two"] = 2;
12 numDict["three"] = 3;

```

```

13 numDict.remove("three");
14 /* Now:
15     numDict = {"one" : 1, "two" : 2}
16     success = true
17 */
18
19 dict<node, num> cityRankings;
20 node x("chicago");
21 cityRankings[x] = 2;
22 node y("seattle");
23 cityRankings[y] = 1;
24 /* Now:
25     cityRankings = {x : 2, y : 1}
26 */

```

Listing 4: Declaration of “list” and “dict” types.

## 4.5 Special Operators

### 4.5.1 Node Operators

The node operators outlined in Table 6<sup>5</sup> are all binary operators which take a node object on the left-hand and right-hand sides of the operator.

Operator	Explanation
--	Add undirected edge with no weights between the 2
-->	Add directed edge from left node to right node with no weight
-->[ <i>num</i> ]	Add a directed edge from the left node to the right node with weight <i>num</i>
==	Returns whether the internal ids of 2 nodes match
!=	Returns whether the internal ids of 2 nodes do not match

Table 6: Node Operators

<sup>5</sup>Note: Weighted undirected edges (ex. `x --[7] y`) are buggy in final version

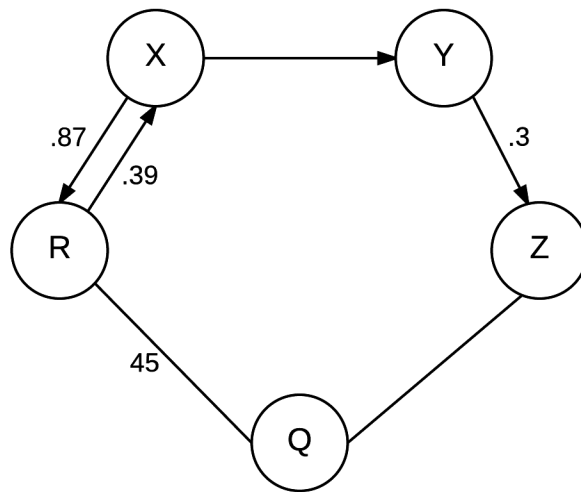


Figure 1: Example Graph showing nodes with different weights and edges.

```

1 node X;
2 node Y;
3 node Z;
4 node Q;
5 node R;
6
7 X --> Y;
8 Y -->[.3] Z;
9 Z -- Q;
10 R [.87]--[.39] X;
11 # and since Q --[45] R is buggy, use:
12 Q -->[45] R;
13 R -->[45] Q;
14
15 node temp;
16 temp = R;
17
18 R == Q; # returns false
19 R == temp; # returns true
20
21 /* accessing edge lists: */
22 X.out() [Y]; # == null
23 Y.out() [Z]; # == .3
24 R.in() [X]; # == .87
25
26 node alt = Z;
27 Y.out() [alt] == Y.out() [Z]; # returns true
28
29 /* Alternate Graph Creation:
30 * adds the nodes to the graph G, while
31 * at the same time it adds edges and weights
32 * between the nodes

```

```

33 */
34 node x, y, z, q, r;
35 graph G = {
36     x --> y,
37     y --> [.3] z,
38     z -- q,
39     q -- [45] r,
40     r [.87] -- [.39] x
41 };

```

Listing 5: Shows the use of node operators that creates the graph in Figure 6.

#### 4.5.2 Graph Operators

The graph operators outlined in Table 7 are all binary operators which take a graph object on the left-hand and right-hand sides of the operator.

Operator	Explanation
<i>graph</i> + <i>graph</i> <i>graph</i> + <i>node</i>	Returns a graph that contains all of the nodes in the left-hand and right-hand graph or node
<i>graph</i> - <i>graph</i> <i>graph</i> - <i>node</i>	Removes the graph on the right-hand side of the operator from the graph on the left-hand side.
<i>graph</i> == <i>graph</i>	Returns whether the two graphs contain the same nodes.

Table 7: Graph Operators

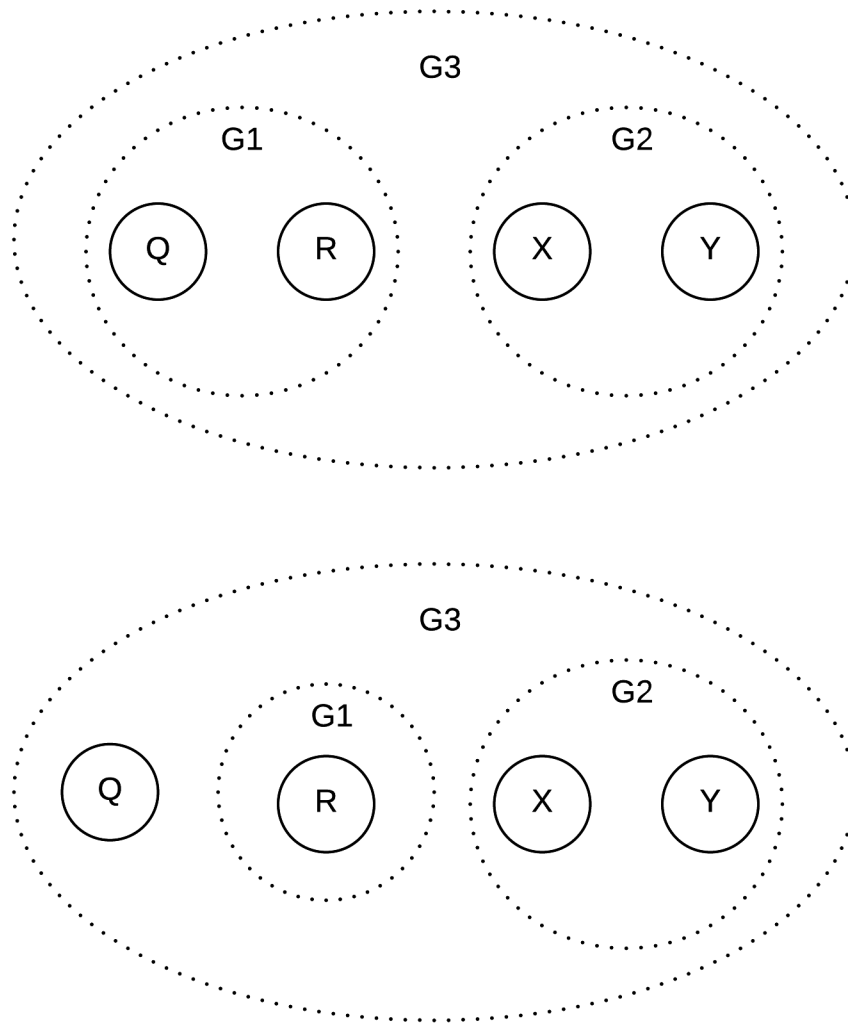


Figure 2: Example showing graphs and graph nesting. The bottom graph is the result of removing the node “Q” from the graph G1.

```

1 node X;
2 node Y;
3 node Q;
4 node R;
5
6 graph G1;
7 graph G2;
8 graph G3;
9
10 G2 = X + Y;
11 G1 = Q;
12 G3 = G1 + G2; # result is the top graph
13
14 G1 = G1 + R;

```



```
15 G1 = G1 - Q; # G1 is back to its original assignment
```

Listing 6: Shows the use of graph operators that creates the top graph in Figure 2 and then alters it to the bottom graph shown.

## 4.6 Built-in Functions

Along with these built in data types, d.o.t.s. provides some built in functions that serve to enhance the user's ability to easily script. These functions require no special declaration on the part of the user.

The `print` function is unusual in that unlike all functions defined by users, which must take a fixed number of arguments, `print` can take any variable number of arguments. It then writes a string representation of each of its arguments to standard out. The `print` function can directly operate on all the types in d.o.t.s. This means that even special types such as `dicts` and `lists` can directly be used as arguments to `print`.

The motivation for the `range` function was for users to be able to quickly iterate through values in orderly manner instead of having to worry about sometimes painful list iteration.

Syntax	Explanation
<code>print(x, ...)</code>	prints to standard output the string representation of a list of comma-separated values or variables.
<code>range(int_lower, int_upper)</code>	returns a list of the integers from <code>int_lower</code> to <code>int_upper</code> , inclusive. The first argument can be omitted, in which case 0 will be used as the default value of <code>int_lower</code> . The data type of both arguments must be <code>int</code> .
<code>len(iterable_var)</code>	returns the length of the iterable variable
<code>max(iterable_var)</code>	Uses the ">" operator for determining values; if ">" is undefined for the object, returns <code>null</code> . For lists, returns the element with greatest value. For dicts, returns the key of (key,value) pair with the greatest value.
<code>min(iterable_var)</code>	Uses the "<" operator for determining values; if "<" is undefined for the object, returns <code>null</code> . For lists, returns the element with lowest value. For dicts, returns the key of (key,value) pair with the lowest value.

Table 8: Built-in Functions

```
1 list<int> x;
2 x = range(1, 3); # x = [1, 2, 3]
3 dict<string, num> y;
4 y["foo"] = 7;
5 y["bar"] = 8.9;
6
7 print("x: ", x, "\ny:", y);
8 /* prints out -->
9  x: [1, 2, 3]
10 y: ["foo": 7.000, "bar": 8.900]
```

```

11 */
12
13 list<string> q = ["foo", "fah"];
14 len(q); # returns 2
15
16 dict<string, num> cityVals = {"chicago" : 2, "seattle" : 1, "nyc" : -8};
17 max(cityVals); # returns "chicago"
18 min(cityVals); # returns "nyc"

```

Listing 7: Shows the use of built-in functions.

## 4.7 Control Flow

As Section 5.4 includes example usage for each of the different types of control statements, this section omits a separate demonstration of their use.

### 4.7.1 Logical Expressions

Two of the types of control flow mechanisms, `if` statements and `while` loops, use logical expressions in their condition statement. In d.o.t.s., a logical expression consists of a sequence of expressions connected by a logical operator. The list of logical operators is given in Table 9.<sup>6</sup>

>
<
<=
>=
==
!=
&&

Table 9: Logical Operators

### 4.7.2 If Statements

```

1 /*
2  condition -- a logical expression
3 */
4 if (condition) {
5  /* 1st code block */
6 }
7
8 if (condition) {
9
10 }
11 else {
12  /* 2nd code block */
13 }

```

<sup>6</sup>Note: while the parser correctly deals with `&&` and `||`, both operators were not fully implemented in the translation side.

The condition of an `if` statement must be a logical expression. If the logical expression evaluates to true, the first code is executed. If the logical expression evaluates to false and there is an `else` block, then the 2nd code block is executed.

### 4.7.3 For Loops

```
1 /*
2  iterable_var -- a variable with elements inside to iterate over
3  temp_var -- variable with the type of elements in iterable_var
4 */
5 for temp_var in iterable_var {
6  /* code block */
7 }
```

A `for` loop may be executed on any iterable variable: i.e. lists, dicts, and graphs. The code within the `for` loop is executed once for each element inside the iterable variable. On each iteration, the next variable inside the iterable variable is assigned to the temporary variable.

The temporary variable in the `for` loop header is automatically created by the d.o.t.s. compiler, and must have a name not already used by a declared variable.it'

### 4.7.4 While

```
1 /*
2  condition -- a logical expression
3 */
4 while (condition) {
5  /* code block */
6 }
```

The condition of a `while` loop must be a logical expression. On each iteration, if the condition evaluates to true, the code block is executed. If the condition evaluates to false, the `while` loop stops iterating and the program moves on to the next instructions. After the code is executed, another iteration is started.

## 4.8 User-defined Functions

```
1 def return_type function_name (arg1_datatype arg1_name, ...) {
2  /* code */
3 }
```

In d.o.t.s., users are allowed to define their own functions, and use them in their code as they would a built-in function.

Elements in a function definition header:

- `def` – the keyword "def" must precede any function definition
- `return_type` – the type of whatever value is returned by the function  
Return types are the same keywords and structure as the types used to declare variables.
- `function_name` – the name used to call the function  
Function names have the same rules as variable names.
- parameter list – The parameter list is the comment-separated list of arguments inside the parentheses of a function definition header.  
Each argument in the parameter list must be of the form: *datatype argname*.

*datatype* is the type of the argument (ex. `num`). Data types are of the same structure as those used in variable declarations.

*argname* is the alias for the argument that is passed in to the function. The corresponding argument can be referenced using its *argname*. Argument names have the same restrictions as variable names.

An empty parameter list is used to define a function that takes 0 arguments.

```
1 def num foo (num x, string s) {
2   print(s, ": ", x);
3 }
4
5 def string bar () {
6   print("Hello there");
7 }
8
9 num y = 5;
10 string phrase = "Items to count: ";
11
12 foo(y, phrase); # prints --> Items to count: 5
13 bar(); # prints --> Hello there
```

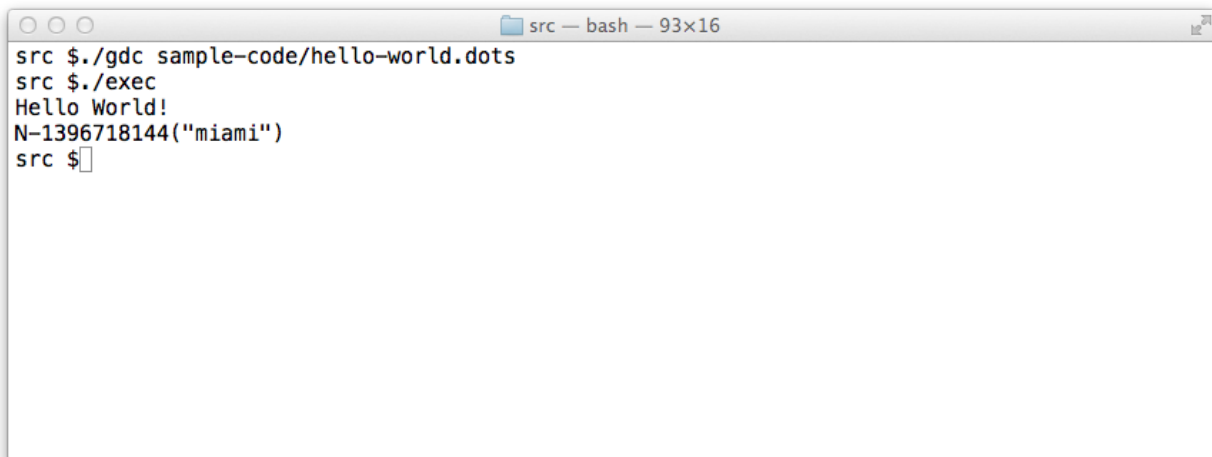
Listing 8: User-defined functions example

## 5 Example Code

### 5.1 Hello World

```
1 node x("miami");
2 string s = "Hello World!";
3 print(s, "\n", x, "\n");
```

Listing 9: Hello World program.



```
src $./gdc sample-code/hello-world.dots
src $./exec
Hello World!
N-1396718144("miami")
src $
```

Figure 3: Output after running `hello-world.dots`.

See Appendix E.1 for the compiled C code.

## 5.2 Breadth-First Search

In this section, we demonstrate a small working example of how you'd use d.o.t.s. to do graph algorithms.

```
1 #breadth first search
2
3 print("Searching\n");
4
5 def bool has_node (list<node> l, node x) {
6     for (n in l) {
7         if (n == x) {
8             return true;
9         }
10    }
11    return false;
12 }
13
14 graph g;
15
16 node x("x");
17 node y("y");
18 node z("z");
19
20 node a("a");
21 node b("b");
22 node c("c");
23
24 g = x + y;
25 g = g + z;
26
27 x -->[2] y;
28 x -->[1.5] z;
29 z -->[4] y;
30 y -->[2] c;
31 z -->[2.5] b;
32 c -->[.5] b;
33 x -->[333] a;
34 z -->[15] a;
35
36 print("Graph Initialized\n");
37
38 list<node> queue;
39 list<node> seen;
40 dict<node, num> dist;
41
42
43 node cur_node;
44 cur_node = x;
45
46 num curr_dist;
47 curr_dist = 0;
48
49 for (n in cur_node.outs()){
```

```

50 print("current node: ", n, "\n", "\n");
51
52 #print("Number of outgoing edges: ", len(cur_node.outs()));
53
54 curr_dist = curr_dist + 1;
55
56 if (has_node(seen, n) == false) {
57     seen.enqueue(x);
58     queue.enqueue(n);
59     dist[n] = curr_dist;
60 }
61 cur_node = queue.peek();
62 queue.dequeue();
63 }
64
65 print(dist, "\n\n");

```

Listing 10: Breadth-First Search Algorithm

```

src -- bash -- 92x16

src ./exec
Searching
Graph Initialized
current node: N--1027589344("a")

current node: N--1027589472("y")

current node: N--1027589408("z")

{N--1027589344("a"): 3.000, N--1027589472("y"): 3.000, N--1027589408("z"): 3.000, }
src $

```

Figure 4: Output after running bst.dots.

See Appendix E.3 for the compiled C code.

### 5.3 User-Defined Functions Example

```

1 def string node_in_list (list<node> l, node x) {
2     for (n in l) {
3         if (n == x) {
4             return "yes";
5         }
6     }
7     return "no";
8 }
9
10 def string node_in_graph (graph g, node x) {
11     for (n in g) {

```

```

12     if (n == x) {
13         return "yes";
14     }
15 }
16 return "no";
17 }
18
19 def list<node> find_intersection(graph gg1, graph gg2) {
20     list<node> result;
21
22     for (n1 in gg1) {
23         for (n2 in gg2) {
24             if (n1 == n2) {
25                 result.enqueue(n1);
26             }
27         }
28     }
29     return result;
30 }
31
32 node x("chicago");
33 node y("bar");
34 node z("foo");
35 node w("blah");
36
37 /* NodeFunctions: */
38
39 list<node> node_list = [x, y, z];
40 string result;
41 print("list contains: \n", node_list, "\n", "\n");
42
43 result = node_in_list (node_list, x);
44 print(x, " in node_list?\n", "\t", result, "\n");
45
46 result = node_in_list (node_list, w);
47 print(w, " in node_list?\n", "\t", result, "\n");
48
49 print("\n\n");
50
51 /* regular graph declaration */
52 graph g1;
53 g1 = x + w;
54 g1 = g1 + y;
55
56 print("G1 contains:\n");
57 for (n1 in g1) {
58     print(n1, "\n");
59 }
60 print("\n");
61
62 /* fancy graph declaration */
63 graph g2 = {
64     x -- y,
65     y --> z,

```

```

66 z -->[22.3] x
67 };
68
69 print("G2 contains: \n");
70 for (n2 in g2) {
71   print(n2, "\n");
72 }
73 print("\n");
74
75 result = node_in_graph(g1, z);
76 print("z in g1? ", result, "\n");
77
78 result = node_in_graph(g2, z);
79 print("z in g2? ", result, "\n");
80
81 /* graph function */
82 list<node> union;
83 union = find_intersection(g1, g2);
84
85 print("\nSHARED NODES:\n");

```

Listing 11: Simple example showing off different features of the language including userdefined functions.

```

src ./exec
list contains:
[N-146815552("chicago"), N-146815616("bar"), N-146815680("foo"), ]

N-146815552("chicago") in node_list?
    yes
N-146815744("blah") in node_list?
    no

G1 contains:
N-146815552("chicago")
N-146815744("blah")
N-146815616("bar")

G2 contains:
N-146815552("chicago")
N-146815616("bar")
N-146815616("bar")
N-146815680("foo")
N-146815680("foo")

z in g1? no
z in g2? yes

SHARED NODES:
src $

```

Figure 5: Output after running sample01.dots.



See Appendix E.2 for the compiled C code.

## 5.4 Dijkstra's Algorithm

The Dijkstra's Algorithm example in Listing 12 was our end-goal for the d.o.t.s. compiler. Unfortunately, although pieces of the necessary elements have been implemented and are working, we didn't have enough time to implement all the pieces needed for this example to run.

```
1 /*
2  * Dijkstra's algorithm: calculates shortest paths starting
3  * from the source node and returns a dict of the lowest cost
4  * to destination nodes
5  */
6
7 def dict<node, num> relax (node u, dict<node, num> w) {
8     for (v in u.out()) {
9         if (w[v] > u.out()[v]) {
10            w[v] = u.out()[v];
11        }
12    }
13    return w;
14 }
15
16 def dict<node, num> dijkstra (graph G, node source){
17     dict<node, num> S;
18     dict<node, num> Q;
19
20     for (n in G) {
21         Q[n] = INF;
22     }
23     Q[source] = 0;
24
25     while (len(Q) != 0) {
26         node u;
27         u = min(Q);
28         num w;
29         w = Q[u];
30         Q.remove(u);
31         S[u] = w;
32         Q = relax(u, Q);
33     }
34     return S;
35 }
36
37 /* Graph set-up */
38 node x("dc");
39 node y("chicago");
40 node z("philly");
41 node q("nyc");
42 node r("boston");
43
44 graph g1 = {
45     x --[2] z,
46     z --[2] q,
47     q --[3] r,
```

```

48     z --[9] r,
49     x --[8] y,
50     y --[9] r
51 };
52
53 /* end Graph set-up */
54
55 /* find the min costs from "philly" to all other cities: */
56 dict<node, num> min_costs;
57 min_costs = dijkstra(g1, z);

```

Listing 12: Dijkstra's Algorithm

Now that we have defined the `bfs` and `dijkstra` functions, we can use them to find shortest paths in an actual instance of a graph.

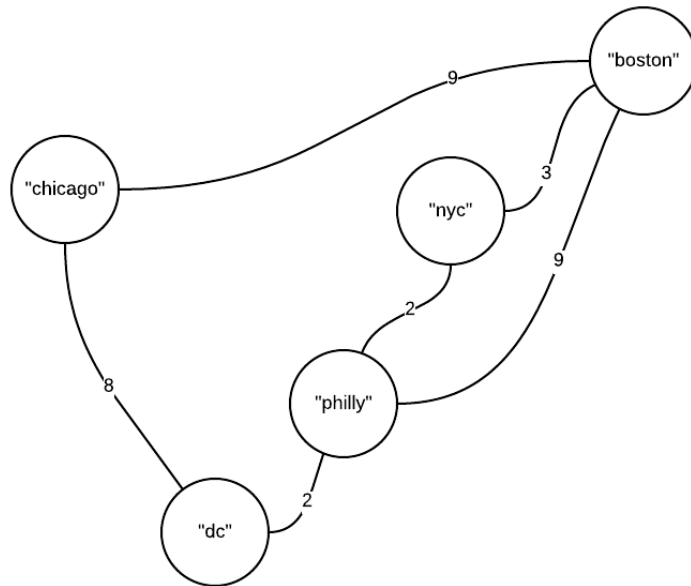


Figure 6: Visual representation of graph created in Listing 13.

```

1 /* Graph set-up */
2 node x("dc"), y("chicago"), z("philly"), q("nyc"), r("boston");
3
4 graph g1 = {
5     x --[2] z;
6     z --[2] q;
7     q --[3] r;
8     z --[9] r;
9     x --[8] y;
10    y --[9] r;
11 };
12 /* end Graph set-up */
13
14 # find the min costs from "philly" to all other cities:

```

```

15 dict<node, num> min_costs = dijkstra(g1, z);
16
17 # find path with minimum number of hops from "philly" to "chicago":
18 list<node> min_path = bfs(g1, z, y);

```

Listing 13: Using user-defined functions.

## 6 Project Development and Lifetime

### 6.1 Project Plan

Our team planned a weekly meeting usually on Sunday nights. During our in-person meeting time we would spend roughly ten minutes going over the checklist of things that needed to get done and then worked on those either individually or paired. We also kept each other up to date on work we'd done during the week with biweekly "standups" via Slack in which we updated each other on things we were working on, things that we needed to work on, and blockers. Each of those updates were posted to a respective Slack channel. For example, AST updates were posted on the AST channel, type converter issues to the Typeconverter channel, etc.

### 6.2 Planning and Synchronization

Once we had a better understanding of what we wanted to accomplish in terms of language features, we wrote an extensive Language Reference Manual including all of our goals, language features, and expected use cases. Aquila, our TA, met with us to go over the status of our LRM and gave us feedback as to which language to compile to and we ended up choosing C instead of C++ to avoid being considered a "java to java" language. From there we created a GitHub repository in order of us to handle version control and merge conflict issues.

### 6.3 Project Development

Initially we began by pushing all of our code to the `master` branch. However we quickly realized that this did not allow us to roll back changes, if necessary, to the last working version. Instead, we decided to create several branches but primarily the `clib` and `compile` branches which continuously got merged back into combined branches when they fit the following criteria:

1. dots source code compiles to `.dotc` file
2. dots source code compiles to `./exec` file (our C executable)
3. positive unit tests have been written for the particular language feature
4. negative unit tests have been written for the particular language feature
5. expected output file written to match the positive and negative tests
6. code passes when you run `runtest.py`

branches were only merged back to the feature branch when the entire file you were working on compile with no shift reduce, warnings, or error messages. This allowed us to test each new file we added incrementally so debugging was easier. Additionally, to allow testing run more smoothly we attempted to work on the `print` functions first for each stage of development in order to run the aforementioned test on each new language feature.

## 6.4 Development Environment

Dots Compiler Written Using:

- OCaml
- OCamllex
- OCaml yacc

Testing Performed Using:

- menhir
  - Python 2.7
- dependencies: subprocess32

C Library Written Using:

- C (on unix)
- gcc

Final Compilation Done Using:

- gcc

*Note:* On Mac (which was our development environment), `gcc` is really an alias for `clang`. So everywhere we mention `gcc`, behind the scenes it's actually using `clang`.

## 6.5 Style Guide

1. Comment particularly esoteric sections of code
2. Indent code hierarchically
3. For long parameter lists, put each parameter on a separate line
4. For long lines of code, put sub-pieces on a separate line

## 6.6 Project Log

Figure 7 shows an excerpt from our git commit history. For the full git log, see Appendix F.

```
gitlog.txt

commit cf233e03b463ffad062fd243462b92c26bdcf470
Author: Hosanna <miramonte23@gmail.com>
Date:   Wed Nov 4 14:36:04 2015 -0500

    added a baseline interpreter

commit bf58fd6c1e116697951f3bea33880f1daf0afdab
Merge: c7de03b f94390c
Author: Hosanna <miramonte23@gmail.com>
Date:   Wed Nov 4 14:29:18 2015 -0500

    Merge branch 'parser' of https://github.com/adamincera/dots into parser

commit c7de03b76bdc1b0d029249c8f6db856386d6f56f
Author: Hosanna <miramonte23@gmail.com>
Date:   Wed Nov 4 14:29:00 2015 -0500

    added microC version of our code

commit f94390c2063798a81039588751208dcf44bfed93
Author: Yumeng Liao <yl2908@columbia.edu>
Date:   Mon Nov 2 22:34:44 2015 -0500

    added method to write tests that should fail and print that it should have failed but passed

commit df650eeddb0244037cb4291fa1bed7edeb8406e7
Author: Yumeng Liao <yl2908@columbia.edu>
Date:   Mon Nov 2 21:57:54 2015 -0500

    worked out a few list and dict tests

commit 29021b3f4c78356aa75b0634681295904022e597
Merge: a2de0d7 aa14b69
Author: rrgordon <rcgordon@umass.edu>
Date:   Mon Nov 2 21:50:46 2015 -0500

    Merge branch 'parser' of https://github.com/adamincera/dots into parser

    merge

commit a2de0d72728744b73412f9edbea98de147490761
Author: rrgordon <rcgordon@umass.edu>
Date:   Mon Nov 2 21:50:32 2015 -0500

    changed list decl assignment to require a actuals_list instead of formals_list, meaning that
    expressions can be assigned to lists.
```

Figure 7: Excerpt from our git commit history.

## 6.7 Roles and Responsibilities

Hosanna Fuller	Project Manager – In charge of scheduling, divying out tasks, keeping us on track, sprint goals.  OCaml Developer – develop OCaml portion of the d.o.t.s. compiler
Adam Incera	C Guru – Sole author of C Libraries that underlie the d.o.t.s. compiler  OCaml Assistant – Helped with development of the analyzer.ml file.
Yumeng Liao	Tester – In charge of writing the test suites and test cases. Debugger of OCaml Compiler.  OCaml Developer – develop OCaml portion of the d.o.t.s. compiler
Rachel Gordon	Dots Guru – In charge of d.o.t.s. language and feature design.  OCaml Guru – Develop and design OCaml Code, assist team members with OCaml development and understanding

Table 10: Roles and Responsibilities

## 7 Compiler Design

Conceptually, the d.o.t.s. compiler is split into four pieces:

1. Input Processing – Interpreting the `.dots` source file into a basic abstract syntax tree.  
Files: `scanner.mll`, `parser.mly`, `ast.ml`
2. DOTS Handling – Processing the DOTS syntax trees, including semantic checking.  
Files: `ast.ml`, `Sast.ml`, `typeConverter.ml`
3. C Handling – Converting DOTS abstract syntax trees into C abstract syntax trees, and converting that tree into actual C code.  
Files: `Sast.ml`, `analyzer.ml`, `translate.ml`
4. Binary Compilation – turning the C code into a binary executable.  
`gcc`

Figure 8 gives a high-level look at the compiler, while Sections 7.1 - 7.3 explain the first three stages in more depth. The final stage, Binary Compilation, consists of running `gcc` with the C Library on the compiled C code.

Hosanna Fuller, Yumeng Liao, and Rachel Gordon worked on all of the OCaml files. Adam Incera also worked on `analyzer.ml`. The C libraries were done entirely by Adam Incera; accordingly, he worked less with the OCaml files.

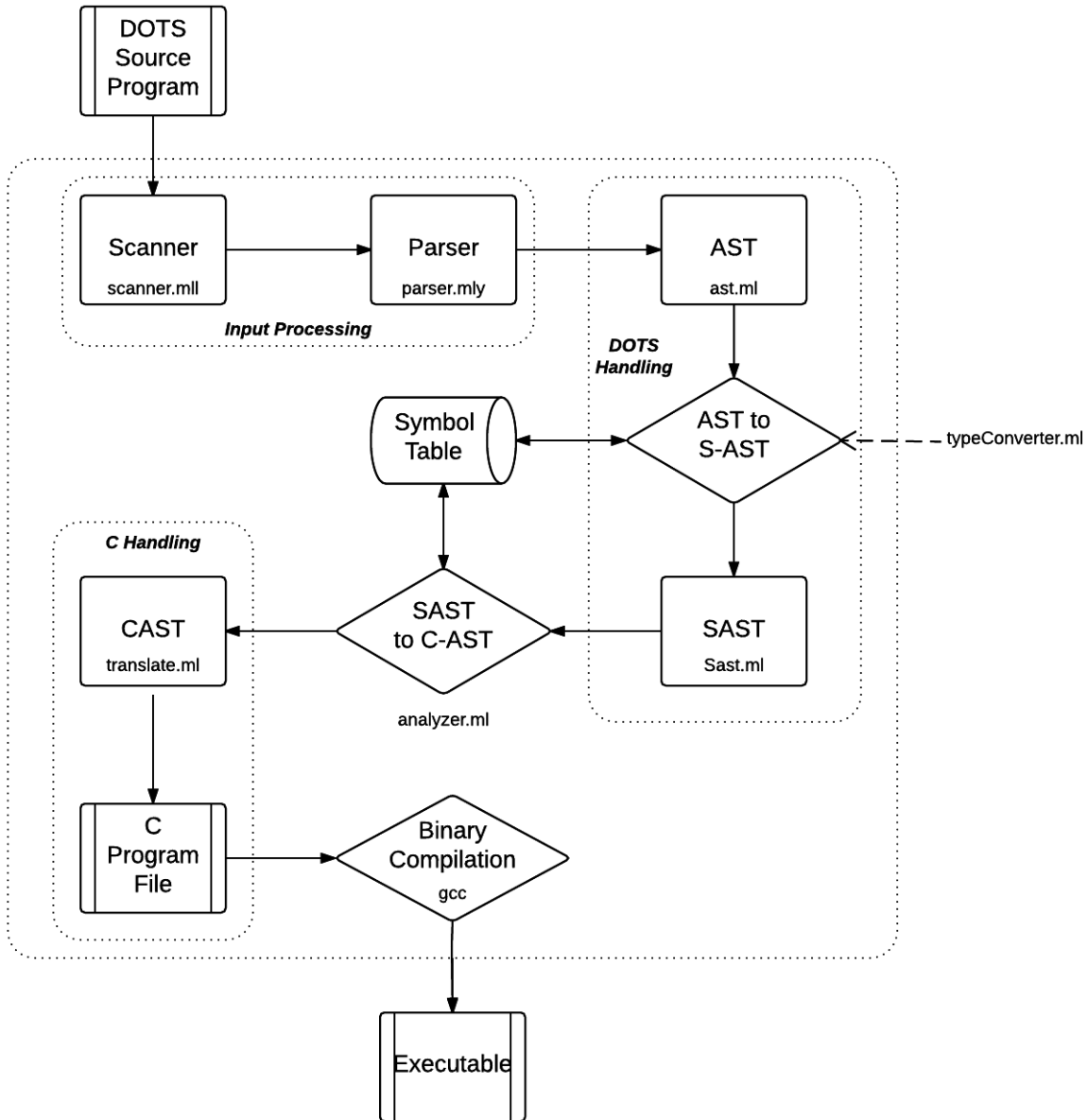


Figure 8: Visual representation of the d.o.t.s. compiler.

## 7.1 Input Processing

This stage consists of converting a source file written in DOTS into a series of tokens that can be easier processed by the compiler.

The scanner makes sure that all tokens in the input program are valid, while the parser checks on a basic level whether tokens have been used in an appropriate ordering.

## 7.2 DOTS Handling

This stage consists of semantically checking the basic AST and converting the fully-semantically checked into an SAST (aka a semantically typed AST).

The files `ast.ml` and `Sast.ml` define the elements in a basic AST and semantically typed AST (SAST) respectively. The `typeConverter.ml` file contains the functions which recursively walk an AST program and convert it into an SAST program. During the AST walk, the converter also semantically checks the d.o.t.s. program.

After parsing and scanning have finished, the d.o.t.s. compiler has a basic Abstract Syntax Tree. This basic AST is not ideal, however, for either translating into C or semantically checking the d.o.t.s. code, as the parser cannot fully type each element of the AST. This means that the elements in the AST also are not fully typed. Instead, we needed to convert the AST into a form of the DOTS syntax tree that did associate each element with its type.

The recursive definition of elements in an AST (`expr`, `stmt`, `program`) makes this type checking easier. For elements containing or referencing other objects, the converter can first semantically check and type the AST subelements into an SAST element and then easily extract the appropriate data type to check the parent element.

Using the types, the converter can that check elements of certain types are used in the appropriate places. For example, if the converter is checking an `assign` statement, it will check that the type of the object being assigned matches what is being assigned to it.

This stage also makes sure that all referenced variables have been declared and that the same variable name is not overwritten by a subsequent declaration. To keep track of what variables have already been assigned and which types they are, the converter keeps track of several symbol tables in an environment variable. The first table maps variable names to types, the second maps variable names to indices, the third maps function names to types and the fourth maps function names to indices.

The two index tables are used in the C translation. This is because a d.o.t.s. program has few restrictions on variable names in terms of reserved words. Because of this, it is possible that a user might declare variables and functions which conflict with names in the underlying C libraries. To avoid this problem, we wanted to use generic variable and function names in our converted C file. This way, each function and variable name has a unique number identifying it and can be referenced simply as `v1`, `f1`, etc.

Once the semantic checking and conversion to SAST is finished, the compiler is guaranteed that the input DOTS program is valid.

## 7.3 C Handling

This stage consists of converting the SAST into an equivalent C AST representation of a C program, and converting that C AST into actual C code.

Because the structure of a d.o.t.s. program differs from that of a C program, the SAST required some special handling before it could be converted into a C AST. The main issue was that in d.o.t.s., users can execute any number of statements outside functions, in the manner of a scripting language. C, however, only allows global variable declarations outside of functions. Most scripting-style commands can simply be put into the main function in C. And because order is preserved, and the main function is run first, the executing C program's order will also match. But if all of the script-style commands were dumped into the main function in C, then variables which were treated like globals in d.o.t.s. would become local in the C program, making references to them in user-defined functions fail.

To solve this issue, we first walked through a completed SAST and separated the program into three pieces: variable declarations outside of functions, function declarations and their bodies, and all other statements. The compiler then runs the C conversion on each of the three pieces sequentially. Declaring all globally



in the C program that were declared outside of functions in the `d.o.t.s.` program allows the converted C functions to use them.

After splitting the SAST program into those three programs, the compiler can then convert it into a C AST. The `analyzer.ml` file walks down the SAST and outputs corresponding elements of a C AST. The end result is full C AST program object. The `translate.ml` file can then walk through its C AST and essentially pretty-print a C program.

## 8 C Library

In order to make it feasible to generate C code that creates and manipulates graphs and other complex data structures, we wrote a graph API in C from which the compiler can generate function calls. The library contains implementations for data types for graphs, nodes, lists, and dictionaries, as well as all the standard operations associated with those data structures.

One of the most difficult portions to implement in C was the `dict` data type. The solution we chose is a simplified version of Python's underlying implementation of its dictionaries, where the dictionary is essentially an array of linked lists of key-value pairs. In order to accomplish hashing, we implemented simple datatype-specific hash algorithms.

The `get ()` function followed these steps:

- Hash the key using the appropriate hash function to obtain the hashed key  $k$ .
- Iterate through the list beginning at the  $k$ th element in the array, checking the key of each key-value pair against the unhashed key.
- If a match is found, return the corresponding value.
- If the end of the list is reached, return `NULL`.

The `put ()` function followed these steps:

- Hash the key using the appropriate hash function to obtain the hashed key  $k$ .
- Iterate through the list beginning at the  $k$ th element in the array, checking the key of each key-value pair against the unhashed key.
- If a match is found, copy the value into the value field of the entry.
- If the end of the list is reached, insert a new node containing the key-value pair.

The following hashing algorithms were used (`TABLE_SIZE` is a constant defining the size of the array inside the dictionary):

- `string`: sum the characters and reduce mod `TABLE_SIZE`.
- `num`: reduce mod `TABLE_SIZE`.
- `node`: bit shift the memory address of the node three to the right (because pointers are eight-byte-aligned, so the rightmost bits were always 0), then reduce mod `TABLE_SIZE`. The memory address was used because node equality in `d.o.t.s.` is defined as physical equality.
- `graph`: sum the memory addresses of the nodes it contains, bit shift three to the right, and reduce mod `TABLE_SIZE`.
- `other`: anything else could be hashed by reducing its memory address mod `TABLE_SIZE`.

The other major difficulty encountered was emulating `d.o.t.s.`' generics (`dict` and `list`) in C. Because C doesn't have built-in templates, we used the next best thing, which was `void *s` and function pointers. Each `list` and `dict` operation was written as a generalized static function that was datatype-agnostic and took in a function pointer to perform comparison, copying, or whatever type-specific operation was required.

This function was then accessed through type-specific wrapper functions that cast the datatype as a `void *` and passed in the appropriate function pointer.

## 9 Testing

### 9.1 Test Suites

To help with writing the parser, we wrote tests in `.txt` files that consisted of a string of tokens as defined by our Ocamllex parser file. These were piped into our Menhir testing script to ensure that while adding new rules to the parser nothing that was working would be broken again.

Shortcut to Menhir testing script: D.2

After we started to generate C code, we wrote unit tests in the form of small individual snippets of d.o.t.s. code to test that a particular feature was working. Such features included: operators, function and variable declarations, looping, etc. These snippets were organized in sub-directories according to what core functionality they were testing. Test scripts in the "complex" directory contain programs that are more practical and combine multiple features.

Shortcut to main testing script: D.1

*Note:* to run tests, make sure to use Python's pip package manager to install the required packages, using "pip install -r requirements.txt" from the src directory.

### 9.2 Test Automation

Both testing scripts made use of Python's subprocess module to open the appropriate processes. The Menhir script runs all tests in the "menhir-tests" directory, reading each line of each `.txt` file so as to see which tests are supposed to pass and which are supposed to fail. The test script ignores lines beginning with "\*\*\*\*" and looks for failure or a pass condition on lines beginning with "f\*\*". Different flags in running the test script provide different forms of output such as: suppressing errors, printing the full tree, etc.

The main testing script looks through the `dtest` directory, which is dedicated to tests that should pass (aka positive tests), and the `ntest` directory, which is dedicated to tests that should fail (aka negative tests). For both, it walks through all subdirectories and builds a `.c` file, executable, and processes the `gdc` (the shell script that ties together all parts of our compiler) output. It then runs the executable (if there is one) and compares the output to a pre-written `.out` file. A difference between expected and actual output means failure. Such differences are recorded in `.dif` files. Specifying certain flags will clean up intermediate files and will only print the full list of results. It's as simple as running "python runtest.py".

### 9.3 Source to Target

Source program (bst.dots):

```
1 #breadth first search
2
3 print("Searching\n");
4
5 def bool has_node (list<node> l, node x) {
6     for (n in l) {
7         if (n == x) {
8             return true;
9         }
10    }
```

```

10 }
11 return false;
12 }
13
14 graph g;
15
16 node x("x");
17 node y("y");
18 node z("z");
19
20 node a("a");
21 node b("b");
22 node c("c");
23
24 g = x + y;
25 g = g + z;
26
27 x -->[2] y;
28 x -->[1.5] z;
29 z -->[4] y;
30 y -->[2] c;
31 z -->[2.5] b;
32 c -->[.5] b;
33 x -->[333] a;
34 z -->[15] a;
35
36 print("Graph Initialized\n");
37
38 list<node> queue;
39 list<node> seen;
40 dict<node, num> dist;
41
42
43 node cur_node;
44 cur_node = x;
45
46 num curr_dist;
47 curr_dist = 0;
48
49 for (n in cur_node.outs()){
50     print("current node: ", n, "\n", "\n");
51
52     #print("Number of outgoing edges: ", len(cur_node.outs()));
53
54     curr_dist = curr_dist + 1;
55
56     if (has_node(seen, n) == false) {
57         seen.enqueue(x);
58         queue.enqueue(n);
59         dist[n] = curr_dist;
60     }
61     cur_node = queue.peek();
62     queue.dequeue();
63 }

```

```
64
65 print(dist, "\n\n");
```

Listing 14: Breadth-First Search Algorithm

Target program: bst.dots.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <dict.h>
5 graph_t * v1 = NULL;
6 node_t * v2;
7 node_t * v3;
8 node_t * v4;
9 node_t * v5;
10 node_t * v6;
11 node_t * v7;
12 list_t * v8 = NULL;
13 list_t * v9 = NULL;
14 entry_t** v10 = NULL;
15 node_t * v11;
16 float v12;
17 int f6 (list_t * v111, node_t * v112)
18 {
19     list_t ** v113;
20     v113 = &(v111);
21     node_t * v115;
22     list_t * v114 = NULL;
23     for (v114 = *(v113); v114; v114 = (v114)->next) {
24         v115 = *((node_t **) ((v114)->data));
25         node_t ** v116;
26         v116 = &(v115);
27         node_t ** v117;
28         v117 = &(v112);
29         if (*(v116) == *(v117)) {
30             int* v118;
31             v118 = malloc(sizeof(int));
32             *(v118) = 1;
33             return *(v118);
34         } else {
35
36         }
37     }
38     int* v119;
39     v119 = malloc(sizeof(int));
40     *(v119) = 0;
41     return *(v119);
42 }
43
44 int main (int argc, char ** argv)
45 {
46     char ** v13;
```

```

47 v13 = malloc(sizeof(char *));
48 *(v13) = malloc(strlen("Searching\n") + 1);
49 strcpy(*(v13), "Searching\n");
50 printf("%s", *(v13));;
51 char ** v14;
52 v14 = malloc(sizeof(char *));
53 *(v14) = malloc(strlen("x") + 1);
54 strcpy(*(v14), "x");
55 v2 = init_node("");
56 (v2)->data = *(v14);
57 char ** v15;
58 v15 = malloc(sizeof(char *));
59 *(v15) = malloc(strlen("y") + 1);
60 strcpy(*(v15), "y");
61 v3 = init_node("");
62 (v3)->data = *(v15);
63 char ** v16;
64 v16 = malloc(sizeof(char *));
65 *(v16) = malloc(strlen("z") + 1);
66 strcpy(*(v16), "z");
67 v4 = init_node("");
68 (v4)->data = *(v16);
69 char ** v17;
70 v17 = malloc(sizeof(char *));
71 *(v17) = malloc(strlen("a") + 1);
72 strcpy(*(v17), "a");
73 v5 = init_node("");
74 (v5)->data = *(v17);
75 char ** v18;
76 v18 = malloc(sizeof(char *));
77 *(v18) = malloc(strlen("b") + 1);
78 strcpy(*(v18), "b");
79 v6 = init_node("");
80 (v6)->data = *(v18);
81 char ** v19;
82 v19 = malloc(sizeof(char *));
83 *(v19) = malloc(strlen("c") + 1);
84 strcpy(*(v19), "c");
85 v7 = init_node("");
86 (v7)->data = *(v19);
87 node_t ** v20;
88 v20 = &(v2);
89 node_t ** v21;
90 v21 = &(v3);
91 graph_t ** v22;
92 v22 = malloc(sizeof(graph_t *));
93 *(v22) = (node_plus_node(*(v20), *(v21)));
94 graph_t ** v23;
95 v23 = &(v1);
96 *(v23) = *(v22);
97 graph_t ** v24;
98 v24 = &(v1);
99 node_t ** v25;
100 v25 = &(v4);

```

```

101 graph_t ** v26;
102 v26 = malloc(sizeof(graph_t *));
103 *(v26) = (graph_plus_node(*(v24), *(v25)));
104 graph_t ** v27;
105 v27 = &(v1);
106 *(v27) = *(v26);
107 float* v28;
108 v28 = malloc(sizeof(float));
109 *(v28) = 2;
110 connect_dir_weighted (v2, v3, *(v28));
111 float* v29;
112 v29 = malloc(sizeof(float));
113 *(v29) = 1.5;
114 connect_dir_weighted (v2, v4, *(v29));
115 float* v30;
116 v30 = malloc(sizeof(float));
117 *(v30) = 4;
118 connect_dir_weighted (v4, v3, *(v30));
119 float* v31;
120 v31 = malloc(sizeof(float));
121 *(v31) = 2;
122 connect_dir_weighted (v3, v7, *(v31));
123 float* v32;
124 v32 = malloc(sizeof(float));
125 *(v32) = 2.5;
126 connect_dir_weighted (v4, v6, *(v32));
127 float* v33;
128 v33 = malloc(sizeof(float));
129 *(v33) = .5;
130 connect_dir_weighted (v7, v6, *(v33));
131 float* v34;
132 v34 = malloc(sizeof(float));
133 *(v34) = 333;
134 connect_dir_weighted (v2, v5, *(v34));
135 float* v35;
136 v35 = malloc(sizeof(float));
137 *(v35) = 15;
138 connect_dir_weighted (v4, v5, *(v35));
139 char ** v36;
140 v36 = malloc(sizeof(char *));
141 *(v36) = malloc(strlen("Graph Initialized\n") + 1);
142 strcpy(*(v36), "Graph Initialized\n");
143 printf("%s", *(v36));;
144
145 v11 = init_node("");
146 (v11)->data = "";
147 node_t ** v37;
148 v37 = &(v2);
149 node_t ** v38;
150 v38 = &(v11);
151 *(v38) = *(v37);
152 float* v39;
153 v39 = malloc(sizeof(float));
154 *(v39) = 0;

```

```

155 float* v40;
156 v40 = &(v12);
157 *(v40) = *(v39);
158 node_t ** v41;
159 v41 = &(v11);
160 entry_t*** v42;
161 v42 = &((*v41)->out);
162 int v69;
163 entry_t* v43;
164 void* v44;
165 if (*(v42)) {
166 for (v69 = 0; v69 < TABLE_SIZE; v69 = v69 + 1) {
167 for (v43 = (*(v42))[v69]; v43; v43 = (v43)->next) {
168 v44 = (v43)->key;
169 char ** v70;
170 v70 = malloc(sizeof(char *));
171 *(v70) = malloc(strlen("current node: ") + 1);
172 strcpy(*(v70), "current node: ");
173 printf("%s", *(v70));
174 node_t ** v71;
175 v71 = &(v44);
176 printf("%s", "N-");
177 printf("%d", (int) (*(v71)));
178 printf("%s", "\\");
179 printf("%s", (char *) ((*(v71))->data));
180 printf("\\");
181 char ** v72;
182 v72 = malloc(sizeof(char *));
183 *(v72) = malloc(strlen("\\n") + 1);
184 strcpy(*(v72), "\\n");
185 printf("%s", *(v72));
186 char ** v73;
187 v73 = malloc(sizeof(char *));
188 *(v73) = malloc(strlen("\\n") + 1);
189 strcpy(*(v73), "\\n");
190 printf("%s", *(v73));
191 float* v74;
192 v74 = &(v12);
193 float* v75;
194 v75 = malloc(sizeof(float));
195 *(v75) = 1;
196 float* v76;
197 v76 = malloc(sizeof(float));
198 *(v76) = (*(v74) + *(v75));
199 float* v77;
200 v77 = &(v12);
201 *(v77) = *(v76);
202
203 list_t ** v78;
204 v78 = &(v9);
205 node_t ** v79;
206 v79 = &(v2);
207 list_t ** v80;
208 v80 = malloc(sizeof(void));

```

```

209 *(v78) = node_add_back(*(v78), *(v79));;
210 list_t ** v81;
211 v81 = &(v8);
212 node_t ** v82;
213 v82 = &(v44);
214 list_t ** v83;
215 v83 = malloc(sizeof(void));
216 *(v81) = node_add_back(*(v81), *(v82));;
217 entry_t*** v84;
218 v84 = &(v10);
219 node_t ** v85;
220 v85 = &(v44);
221 float* v86;
222 v86 = &(v12);
223 node_t * v88;
224 v88 = *(v85);
225 float v87;
226 v87 = *(v86);
227 *(v84) = put_node(*(v84), (node_t *) (v88), (void*) (&(v87)));;
228 list_t ** v89;
229 v89 = &(v8);
230 node_t ** v90;
231 v90 = peek(*(v89));;
232 node_t ** v91;
233 v91 = &(v11);
234 *(v91) = *(v90);
235 list_t ** v92;
236 v92 = &(v8);
237 void* v93;
238 *(v92) = pop(*(v92));;
239 }
240 }
241 } else {
242
243 }
244 printf("{");
245 entry_t*** v95;
246 v95 = &(v10);
247 int v104;
248 entry_t* v96;
249 void* v97;
250 if (*(v95)) {
251 for (v104 = 0; v104 < TABLE_SIZE; v104 = v104 + 1) {
252 for (v96 = (*(v95))[v104]; v96; v96 = (v96)->next) {
253 v97 = (v96)->key;
254 node_t ** v105;
255 v105 = &(v97);
256 printf("%s", "N-");
257 printf("%d", (int) (*(v105)));;
258 printf("%s", "\\");
259 printf("%s", (char *) ((*(v105))->data));
260 printf("\\");;
261 char ** v106;
262 v106 = malloc(sizeof(char *));;

```



```

263 *(v106) = malloc(strlen(": ") + 1);
264 strcpy(*(v106), ": ");
265 printf("%s", *(v106));;
266 entry_t*** v107;
267 v107 = &(v10);
268 node_t ** v108;
269 v108 = &(v97);
270 float* v109;
271 v109 = (float*)(get_node(*(v107), *(v108)));
272 printf("%.3f", *(v109));;
273 char ** v110;
274 v110 = malloc(sizeof(char *));
275 *(v110) = malloc(strlen(", ") + 1);
276 strcpy(*(v110), ", ");
277 printf("%s", *(v110));;
278 }
279 }
280 } else {
281
282 }
283 printf("}");
284 char ** v111;
285 v111 = malloc(sizeof(char *));
286 *(v111) = malloc(strlen("\n\n") + 1);
287 strcpy(*(v111), "\n\n");
288 printf("%s", *(v111));;
289 }

```

Listing 15: bst.dots.c

## 10 Lessons Learned

### 10.1 Hosanna's Advice

The sooner you dig into OCaml, the better off you are going to be. Even if you have no idea how to write the compiler or some other assignment you have. Start building small programs because they will most likely come in handy some time in the future.

Similarly, try to truncate problems out into smaller sub-problems and try to get those parts to compile. It's easy to be ambitious in the beginning and try to get your entire parser working in one fell swoop, but it's better to test incrementally. Especially with OCaml's error messages, you need to be careful and build your code in a modular manner.

### 10.2 Adam's Advice

Keep up with what the rest of your team is doing! I spent a large part of the semester implementing the C library, and by the time I finished it, the code base was big enough that I had to spend a bunch more time just getting up to speed on what everyone else had written.

Do the calculator assignment. Get comfortable with OCaml as soon as you possibly can, so that when the time comes you can focus on actually writing your compiler instead of trying to decipher a language you don't understand.

## 10.3 Rachel's Advice

If you only ever do one homework assignment, whatever you do make it the calculator assignment. Doing that assignment teaches you the basics of how to think in OCaml, and if you don't do it, then the learning curve when you go to do your project is going to be huge.

Prioritize implementation of your language based on your end-goal example code. If your language design includes W, X, Y, Z, but the program you want to be able to demo only uses X and Y, start with X and Y, don't waste time implementing Z.

Sit down and design your compiler on paper before you start coding. Think about specific examples of what you want to be able to write in your language and what that's going to mean in terms of what information you need encoded in your ASTs. If you only design it as you write it, you're going to end up having to rewrite portions of the compiler multiple times.

Don't develop your underlying libraries and compiler in isolation. They're interdependent, so if you don't keep that in mind when you're designing your system, the different components won't work well together and might not be able to express what you need them to.

## 10.4 Yumeng's Advice

Don't try to stay within the strict definitions of the role that you are assigned within the team, because each role must know everything about what everyone else is doing in order to put out quality work. As the tester, early on I fell into the trap of writing tests that weren't immediately relevant to what the rest of the team was doing because I didn't understand how the compiler was working. As a result, my first few tests weren't useful to the team. I was only able to write quality tests only after I knew the intricacies of what everyone else was doing and after I'd put work into the compiler.

Try to find people you like, and more importantly, respect tremendously. It'll be much easier to treat each other with kindness, and trust that your teammates will pull their weight.

# Appendices

## A OCaml Compiler Files

### A.1 scanner.mll

```
1 { open Parser }
2
3 let num = ['0'-'9']+
4 let num_regex = '-'? (num* '.' num+) | (num+ ('.' num*)?)
5
6 rule token = parse
7   [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
8 | "/"* { multicomment lexbuf } (* Multi-Line Comments *)
9 | "#" { comment lexbuf } (* Single-Line Comments *)
10 | '(' { LPAREN }
11 | ')' { RPAREN }
12 | '{' { LBRACE }
13 | '}' { RBRACE }
14 | '[' { LBRACKET }
```

```

15 | ']' { RBRACKET }
16 | ';' { SEMI }
17 | ':' { COLON }
18 | ',' { COMMA }
19 | '+' { PLUS }
20 | '-' { MINUS }
21 | '*' { TIMES }
22 | '/' { DIVIDE }
23 | '=' { ASSIGN }
24 | '.' { DOT }
25 | "&&" { LOGAND }
26 | "||" { LOGOR }
27 | "--" { UEDGE }
28 | "-->" { REDGE }
29 | "==" { EQ }
30 | "!=" { NEQ }
31 | "<" { LT }
32 | "<=" { LEQ }
33 | ">" { GT }
34 | ">=" { GEQ }
35 | "def" { DEF }
36 | "if" { IF }
37 | "else" { ELSE }
38 | "true" { TRUE }
39 | "INF" { INF }
40 | "false" { FALSE }
41 | "in" { IN }
42 | "for" { FOR }
43 | "while" { WHILE }
44 | "return" { RETURN }
45 | "bool" { BOOL }
46 | "num" { NUM }
47 | "string" { STRING }
48 | "node" { NODE }
49 | "graph" { GRAPH }
50 | "list" { LIST }
51 | "dict" { DICT }
52 | num_regex as lxm { NUM_LIT(lxm) } (* num literal *)
53 | ''' ([^']* as lxm) ''' { STR_LIT(lxm) } (* string literals *)
54 | ['a'-'z' 'A'-'Z'] ['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
55 | eof { EOF }
56 | _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
57
58 and multicomment = parse
59   "*/" { token lexbuf }
60 | _ { multicomment lexbuf }
61
62 and comment = parse
63   "\n" { token lexbuf }
64 | _ { comment lexbuf }

```

Listing 16: scanner.mll

## A.2 parser.mly

```

1 %{ open Ast %}
2
3 /* Punctuation Tokens */
4 %token SEMI COLON LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA DOT
5 /* Arithmetic Operation Tokens */
6 %token PLUS MINUS TIMES DIVIDE
7 /* Assignment Operator */
8 %token ASSIGN
9 /* Comparative Operators */
10 %token EQ NEQ LT LEQ GT GEQ
11 /* Logical Operators */
12 %token LOGAND LOGOR
13 /* Node Operators */
14 %token UEDGE REDGE
15 /* Function Keyword Tokens */
16 %token RETURN IF ELSE FOR WHILE DEF IN
17 /* Punctuation Tokens */
18 %token BOOL NUM STRING NODE GRAPH LIST DICT
19 /* Boolean Operations */
20 %token TRUE FALSE INF
21
22 %token <string> NUM_LIT
23 %token <string> STR_LIT
24 %token <string> ID
25 %token EOF
26
27 /* Order of Operation */
28 %nonassoc NOCALL
29 %nonassoc NOELSE
30 %nonassoc ELSE
31 %right ASSIGN
32 %left LOGOR
33 %left LOGAND
34 %left EQ NEQ
35 %left LT GT LEQ GEQ
36 %left PLUS MINUS
37 %left TIMES DIVIDE
38
39 %start program          /* start symbol */
40 %type <Ast.program> program /* return type program object */
41
42 %%
43
44 /* START PROGRAM */
45
46 program:
47   decls EOF { $1 }
48
49 decls:
50 | /* nothing */ { { cmds = [] } }
51 | decls stmt { { cmds = $2 :: $1.cmds } }
52
53
54 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

55             /* FUNCTIONS */
56 ///////////////////////////////////////////////////////////////////
57
58 /* (1)def (2)func (3)<funcName> ( (5)arg1,...argN ) { (8) <local variables>
   (9) <body> } */
59 fdecl:
60   DEF f_data_type ID LPAREN formals_opt RPAREN LBRACE non_func_stmt_list
   RBRACE
61   { Fdecl({
62     rtype = $2;
63     fname = $3;
64     formals = $5;
65     body = List.rev $8
66   }) }
67
68 /* Optional Formal Args */
69 formals_opt:
70   /* nothing */ { [] }
71   | formal_list { List.rev $1 }
72
73 formal_list:
74   f_data_type ID { [($1, $2)] }
75   | formal_list COMMA f_data_type ID { ($3, $4) :: $1 }
76
77 /* comma separated list of operations on nodes
78 * for use with graph declarations
79 */
80 /* Edge Operations */
81 edge_op_list:
82 | edge_op { [$1] }
83 | edge_op_list COMMA edge_op { $3 :: $1 }
84
85 edge_op:
86 /* ID { NoOp($1) }*/
87 | ID UEDGE ID { Undir($1, $3) } /* x -- y */
88 | ID REDGE ID { Dir($1, $3) } /* x --> y */
89 | ID REDGE LBRACKET expr RBRACKET ID { DirVal($1, $6, $4) } /* x -->[5] y */
90 | ID UEDGE LBRACKET expr RBRACKET ID { UndirVal($1, $6, $4) } /* x --[5] y */
91 /* bug with bidirectional weighted edges: */
92 /*| ID LBRACKET expr RBRACKET UEDGE LBRACKET expr RBRACKET ID */ /* x [3]--[5]
   y */
93 /*{ BidirVal($3, $1, $9, $7) } */
94
95 ///////////////////////////////////////////////////////////////////
96             /* VARIABLES */
97 ///////////////////////////////////////////////////////////////////
98
99 /* Literals */
100 literal:
101 | NUM_LIT { NumLiteral($1) }
102 | STR_LIT { StrLiteral($1) }
103 | list_literal {$1}
104 | dict_literal {$1}
105

```

```

106 list_literal:
107 | LBRACKET actuals_opt RBRACKET { ListLiteral($2) }
108
109 dict_literal:
110 | LBRACE tuples_opt RBRACE { DictLiteral($2)}
111
112
113 /* Primitive Typenames */
114 prim_type:
115 | BOOL { "bool" }
116 | NUM { "num" }
117 | STRING { "string" }
118
119 data_type:
120 | prim_type { $1 }
121 | DICT LT data_type COMMA data_type GT { "dict" }
122 | LIST LT data_type GT { "list" }
123 | NODE { "node" }
124 | GRAPH { "graph" }
125
126 f_data_type:
127 | prim_type { Basic($1) }
128 | DICT LT data_type COMMA data_type GT { Dict($3,$5) }
129 | LIST LT data_type GT { List($3) }
130 | NODE { Basic("node") }
131 | GRAPH { Basic("graph") }
132
133 vdecl:
134 | prim_decl_prefix SEMI { $1 }
135 | node_decl_prefix SEMI { $1 }
136 | graph_decl_prefix SEMI { $1 }
137 | list_decl_prefix SEMI { $1 }
138 | dict_decl_prefix SEMI { $1 }
139
140 /* PRIMITIVE INITIALIZERS */
141 prim_decl_prefix:
142 | prim_type ID { Vdecl($1, $2) } /* num x */
143 | prim_type ID ASSIGN expr { Block([Vdecl($1, $2); Assign(Id($2), $4)]) } /*
    MOVE THESE */
144
145 /* NODE INITIALIZERS */
146 node_decl_prefix:
147 | NODE ID { Block[Vdecl("node", $2); NodeDef($2, Noexpr)] }
    /* node x; */
148 | NODE ID LPAREN expr RPAREN { Block([Vdecl("node", $2); NodeDef($2, $4)]) }
    /* node x("chicago") */ /* node x("Chicago") */
149
150 /* GRAPH INITIALIZERS */
151 graph_decl_prefix:
152 | GRAPH ID { Vdecl("graph", $2) } /* graph g;
    */
153 /*| GRAPH ID ASSIGN LBRACE edge_op_list RBRACE { Block([Vdecl("graph", $2);
    AssignList($2, $5)]) } */ /* graph g = { x --[5] y; y -->[3] z; } */
154 | GRAPH ID ASSIGN LBRACE edge_op_list RBRACE { Block([Vdecl("graph", $2);

```

```

155 GraphDef($2,$5)]) } /* graph g = { x --[5] y; y -->[3] z; } */
156
157 list_decl_prefix:
158 | LIST LT data_type GT ID { ListDecl($3, $5) }
159 | LIST LT data_type GT ID ASSIGN expr { Block([ListDecl($3, $5); Assign(Id($5)
160 | $7)]) } /* list<node> min_path = { x, y, z; }; */
161 dict_decl_prefix:
162 | DICT LT data_type COMMA data_type GT ID { DictDecl($3, $5, $7) }
163 | DICT LT data_type COMMA data_type GT ID ASSIGN expr { Block([DictDecl($3, $5
164 | $7); Assign(Id($7), $9)]) } /* dict<node, num> parents = { x; y; z; };
165 */
166 ////////////////////////////////////////////////////
167 ///* STATEMENTS */
168 ////////////////////////////////////////////////////
169
170 /* statements are defined inside functions or executed like a script */
171 /* a statement is just an action. ex. x = 5; */
172
173 stmt:
174 | non_func_stmt {$1}
175 | func_stmt {$1}
176
177 non_func_stmt:
178 | expr SEMI { Expr($1) }
179 | log_expr SEMI { Expr($1) }
180 | edge_op SEMI { Expr($1) }
181 | ID ASSIGN expr SEMI { Assign(Id($1), $3) }
182 /*| access_expr ASSIGN expr SEMI { AccessAssign($1, $3) } */
183 | expr LBRACKET expr RBRACKET ASSIGN expr SEMI { AccessAssign($1, $3, $6) }
184 | RETURN expr SEMI { Return($2) }
185 /* | LBRACE stmt_list RBRACE { Block(List.rev $2) } */
186 | IF LPAREN log_expr RPAREN LBRACE stmt_list RBRACE %prec NOELSE { If($3,
187 | Block($6), Block([])) }
188 | IF LPAREN log_expr RPAREN LBRACE stmt_list RBRACE ELSE LBRACE stmt_list
189 | RBRACE { If($3, Block($6), Block($10)) }
190 | FOR LPAREN ID IN expr RPAREN LBRACE stmt_list RBRACE
191 | { For($3, $5, $8) }
192 | WHILE LPAREN log_expr RPAREN LBRACE stmt_list RBRACE { While($3, $6) }
193 | vdecl { $1 }
194
195 func_stmt:
196 | fdecl {$1}
197
198 /* list of statements */
199 stmt_list:
200 | /* nothing */ { [] }
201 | stmt_list stmt { $2 :: $1 }

```

```

201 non_func_stmt_list:
202   /* nothing */ { [] }
203   | non_func_stmt_list non_func_stmt { $2 :: $1 }
204
205   //////////////////////////////////////
206   /* EXPRESSIONS */
207   //////////////////////////////////////
208
209
210 log_expr:
211   | expr EQ expr { Binop($1, Equal, $3) }
212   | expr NEQ expr { Binop($1, Neq, $3) }
213   | expr LT expr { Binop($1, Less, $3) }
214   | expr LEQ expr { Binop($1, Leq, $3) }
215   | expr GT expr { Binop($1, Greater, $3) }
216   | expr GEQ expr { Binop($1, Geq, $3) }
217   | log_expr LOGAND log_expr { Binop($1, LogAnd, $3) }
218   | log_expr LOGOR log_expr { Binop($1, LogOr, $3) }
219
220 expr:
221   | access_expr { $1 }
222   | nacc_expr { $1 }
223
224 nacc_expr: /* non access exprs */
225   | expr DOT ID LPAREN actuals_opt RPAREN { MemberCall($1, $3, $5) }
226   | LPAREN expr RPAREN { $2 }
227   | term { $1 }
228
229 access_expr:
230   | expr LBRACKET expr RBRACKET { Access($1, $3) }
231
232 term :
233   | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
234   | term PLUS term { Binop($1, Add, $3) }
235   | term MINUS term { Binop($1, Sub, $3) }
236   | term TIMES term { Binop($1, Mult, $3) }
237   | term DIVIDE term { Binop($1, Div, $3) }
238   | atom { $1 }
239
240 atom:
241   literal { $1 }
242   | INF { NumLiteral("INF") }
243   | TRUE { Boolean(True) }
244   | FALSE { Boolean(False) }
245   | ID { Id($1) }
246
247
248   //////////////////////////////////////
249   /* actuals */
250   //////////////////////////////////////
251 actuals_opt:
252   /* nothing */ { [] }
253   | actuals_list { List.rev $1 }
254

```



```

255 tuples_opt:
256 /* nothing*/ {[]}
257 | tuples_list {List.rev $1}
258
259 /* for dictionary assignment */
260 tuples_list:
261   expr COLON expr { [($1, $3)] }
262 | tuples_list COMMA expr COLON expr { ($3, $5) :: $1 }
263
264 /* arguments to a function */
265 actuals_list:
266   expr { [$1] }
267 | actuals_list COMMA expr { $3 :: $1 }

```

Listing 17: parser.mly

### A.3 ast.ml

```

1 type op = | Add | Sub | Mult | Div
2           | Equal | Neq | Less | Leq | Greater | Geq
3           | LogAnd (* && *)
4           | LogOr (* || *)
5
6 type bool = True | False
7
8 type fun_dt =
9   | Basic of (string)
10  | List of (string)
11  | Dict of (string * string)
12
13 type expr =
14   NumLiteral of string
15   | StrLiteral of string
16   | ListLiteral of expr list (* ex. [1, 3, 42.33] *)
17   | DictLiteral of (expr * expr) list (* ex. [(key, value)] *)
18   | Boolean of bool
19   | Id of string
20   | Binop of expr * op * expr
21   | Call of string * expr list
22   | Access of expr * expr (* for dict and list element access, node.in[node2]
23     *)
24   | MemberCall of expr * string * expr list (* expr that evaluates to parent
25     variable, accessed funct, parameters *)
26   | Undir of string * string (* id, id *)
27   | Dir of string * string (* id, id *)
28   | UndirVal of string * string * expr (* id, id, weight *)
29   | DirVal of string * string * expr (* id, id, weight *)
30   | BidirVal of expr * string * string * expr (* weight, id, id, weight *)
31   | NoOp of string
32   | Noexpr
33
34 type stmt =
35   Block of stmt list

```

```

35 | Expr of expr
36 | Vdecl of string * string (* (type, id) *)
37 | ListDecl of string * string (* elem_type, id *)
38 | DictDecl of string * string * string (* key_type, elem_type, id *)
39 | Assign of expr * expr
40 | AccessAssign of expr * expr * expr (* a[5] = 10 where first expr is an
    access expr *)
41 | NodeDef of string * expr (* (node id, what goes inside parens) of item *)
42 (* | AssignList of string * expr *) (* when a list of expressions is assigned
    to a variable *)
43 | GraphDef of string * expr list (* id EdgeOp list - in form of undir dir -
    *)
44 | Return of expr
45 | If of expr * stmt * stmt
46 | For of string * expr * stmt list (* temp var, iterable var, var decls,
    stmts *)
47 | While of expr * stmt list (* condition, var decls, stmt list *)
48 | Fdecl of func_decl and
49
50 func_decl = {
51   rtype : fun_dt;
52   fname : string;
53   formals : (fun_dt * string) list;
54   (*locals : string list;*)
55   body : stmt list;
56 }
57
58 type program = { cmds: stmt list }
59
60 (*
    ////////////////////////////////////////////////////
61
62 ////////////////////////////////////////////////////
    *)
63
64 (* prepends prelist at the head of postlst *)
65 let rec base_concat postlst = function
66 | [] -> postlst
67 | hd :: tl -> base_concat (hd :: postlst) tl
68
69 let concat prelst postlst = base_concat postlst (List.rev prelst)
70
71 let rec string_of_expr = function
72   NumLiteral(l) -> l
73 | StrLiteral(l) -> "\"" ^ l ^ "\""
74 | ListLiteral(el) -> "[" ^ String.concat "," (List.map string_of_expr el) ^
    "]"
75 | DictLiteral(el) -> "[" ^ String.concat "," (List.map (fun f -> "(" ^ (
    string_of_expr (fst f)) ^ " : " ^ (string_of_expr (snd f)) ^ ")" ) el)
76 | Boolean(b) -> if b = True then "true" else "false"
77 | Id(s) -> s
78 | Binop(e1, o, e2) ->
79   string_of_expr e1 ^ " " ^

```

```

80     (match o with
81       Add -> "+"
82       | Sub -> "-"
83       | Mult -> "*"
84       | Div -> "/"
85       | Equal -> "=="
86       | Neq -> "!="
87       | Less -> "<"
88       | Leq -> "<="
89       | Greater -> ">"
90       | Geq -> ">="
91       | LogAnd -> "&&"
92       | LogOr -> "||"
93     ) ^ " " ^
94
95     string_of_expr e2
96 | Undir (s1, s2) -> s1 ^ " -- " ^ s2
97 | Dir (s1, s2) -> s1 ^ " --> " ^ s2
98 | UndirVal (s1, s2, w) -> s1 ^ " --[" ^ string_of_expr w ^ "]" ^ s2
99 | DirVal (s1, s2, w) -> s1 ^ " -->[" ^ string_of_expr w ^ "]" ^ s2
100 | BidirVal (w1, s1, s2, w2) -> s1 ^ " [" ^ string_of_expr w1 ^ "]" --[" ^
    string_of_expr w2 ^ "]" ^ s2
101 | NoOp (s) -> s
102 | Call(f, el) ->
103   f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
104 | Access (e, el) -> string_of_expr e ^ "[" ^ string_of_expr el ^ "]"
105 | MemberCall (el, s2, el) -> string_of_expr el ^ "." ^ s2 ^ "(" ^ String.
    concat ", " (List.map string_of_expr el) ^ ")"
106 | Noexpr -> ""
107
108 let rec f_type_to_string = function
109 | Basic(t) -> t
110 | List(t) -> "list <" ^ t ^ ">"
111 | Dict(kt,vt) -> "dict <" ^ kt ^ ", " ^ vt ^ ">"
112
113 let rec string_of_stmt = function
114   Block(stmts) ->
115     "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
116 | Expr(expr) -> string_of_expr expr ^ ";\n";
117 | Vdecl(dt, id) -> dt ^ " " ^ id ^ ";\n";
118 | ListDecl(dt, id) -> "list <" ^ dt ^ "> " ^ id ^ ";\n"
119 | DictDecl(kdt, vdt, id) -> "dict <" ^ kdt ^ ", " ^ vdt ^ "> " ^ id ^ ";\n"
120 | Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e ^ ";";
121 | AccessAssign(e1, e2, e3) -> string_of_expr e1 ^ "[" ^ string_of_expr e2 ^
    "]" = " ^ string_of_expr e3 ^ ";\n"
122 | NodeDef(v, e) -> v ^ "(" ^ string_of_expr e ^ ")" (* (node id, what goes
    inside parens) of item *)
123 | GraphDef(v, el) -> v ^ " = { " ^ String.concat ", " (List.map
    string_of_expr el) ^ "};"
124 | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
125 | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt
    s
126 | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")\n" ^
    string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
127

```

```

128 | For(e1, e2, sl) ->
129   "for (" ^ e1 ^ " in " ^ string_of_expr e2
130   ^ ") { " ^ String.concat "\n" (List.map string_of_stmt sl) ^ " }"
131 | While(e, sl) -> "while (" ^ string_of_expr e ^ ") {" ^ String.concat "\n"
132   (List.map string_of_stmt sl) ^ " }"
133 | Fdecl(f) -> string_of_fdecl f and
134
135 string_of_fdecl fdecl =
136 "def " ^ (f_type_to_string fdecl.rtype) ^ " " ^ fdecl.fname ^ "(" ^
137 (String.concat ", " (List.map (fun f ->(f_type_to_string (fst f)) ^ " " ^
138   snd f) fdecl.formals)) ^
139   ")\n{\n" ^
140 String.concat "\n" (List.map string_of_stmt fdecl.body) ^
141   "}\n"
142
143 let string_of_vdecl id = "type " ^ id ^ ";\n"
144
145 let string_of_program (funcs, cmds) =
146   String.concat "\n" (List.map string_of_fdecl funcs) ^
147   String.concat "\n" (List.map string_of_stmt cmds)

```

Listing 18: ast.ml

## A.4 Sast.ml

```

1 (* this defines semantically typed dots ast *)
2 module StringMap = Map.Make(String)
3
4 type dataType = | Num | String | Bool
5                 | Graph | Node
6                 | List of dataType (* val type *)
7                 | Dict of dataType * dataType (* key type, val type *)
8                 | Void
9
10
11
12 type s_expr =
13   NumLiteral of string * dataType (* 5 *)
14 | StrLiteral of string * dataType (* "Hello" *)
15 | ListLiteral of s_expr list * dataType (* [2.5, 3, x] *)
16 | DictLiteral of (s_expr * s_expr) list * dataType (* [(Hello, 15)] *)
17 | Boolean of Ast.bool * dataType (* True *)
18 | Id of string * dataType (* x *)
19 | Binop of s_expr * Ast.op * s_expr * dataType (* x + y *)
20 | Call of string * s_expr list * dataType
21 | Access of s_expr * s_expr * dataType (* for dict and list element
22   access *)
23 | MemberCall of s_expr * string * s_expr list * dataType (* parent variable,
24   accessed funct, parameters *)
25 | Undir of string * string * dataType (* id, id *)
26 | Dir of string * string * dataType (* id, id *)
27 | UndirVal of string * string * s_expr * dataType (* id, id, weight *)
28 | DirVal of string * string * s_expr * dataType (* id, id, weight *)

```

```

27 | BidirVal of s_expr * string * string * s_expr * dataType (* weight, id, id
    , weight *)
28 | NoOp of string * dataType
29 | Noexpr
30
31 type s_stmt =
32   Block of s_stmt list
33 | Expr of s_expr
34 | Vdecl of dataType * string
35 | NodeDef of string * s_expr * dataType (* (node id, item id, datatype) *)
36 | GraphDef of string * s_expr list
37 | Assign of s_expr * s_expr * dataType (* x = 5; *)
38 | AccessAssign of s_expr * s_expr * s_expr * dataType (* a[5] = 10 where
    first thing is an access expr *)
39 | Return of s_expr * dataType (* return x (dataType) *)
40 | If of s_expr * s_stmt * s_stmt (* if (boolean) stmt; *)
41 | For of string * s_expr * s_stmt list (* temp var, iterable var, stmts *)
42 | While of s_expr * s_stmt list (* condition, var decls, stmt list *)
43 | Fdecl of s_fdecl and
44
45 s_fdecl = {
46   s_fname : string;
47   s_rtype : dataType;
48   s_formals : (dataType * string) list;
49   s_body : s_stmt list;
50 }
51
52 type program = { s_cmds : s_stmt list }
53
54 let rec dt_to_str = function
55 | Num -> "num"
56 | String -> "string"
57 | Bool -> "bool"
58 | List(dt) -> "list<" ^ (dt_to_str dt) ^ ">"
59 | Dict(dtk, dtv) -> "dict<" ^ (dt_to_str dtk) ^ ", " ^ (dt_to_str dtv) ^ ">"
60 | Graph -> "graph"
61 | Node -> "node"
62 | Void -> "void"
63
64 (* end Sast *)

```

Listing 19: Sast.ml

## A.5 typeConverter.ml

```

1 (*****
2 (* CONVERTS AN AST TO AN SAST *)
3 (*****
4 open Ast
5 open Sast
6 open Analyzer
7
8 (* extract dt from list *)
9 let get_list_type = function

```

```

10 | Sast.List(dt) -> dt
11 | _ -> raise (Failure("wrong type: not a list"))
12
13 (* extract key type, val type from dict *)
14 let get_dict_type = function
15 | Sast.Dict(dt1, dt2) -> (dt1, dt2)
16 | _ -> raise (Failure("wrong type: not a dict "))
17
18 (* make sure each element in a list is the right type *)
19 let rec check_list env v_e = function
20 | [] -> ""
21 | hd::tl ->
22   if not(v_e = (get_sexpr_type hd)) then
23     raise (Failure ("list element not of type: " ^ type_to_str ( v_e ) ) )
24   else
25     check_list env v_e tl
26
27
28 let rec check_graph_list env = function
29 | [] -> ""
30 | hd::tl ->
31   (match hd with
32   | Sast.Id(v, dt) ->
33     let e_dt = get_sexpr_type hd in
34     (if not (e_dt = Sast.Node || e_dt = Sast.Graph) then
35       raise (Failure ("you can not have a graph def with type other than
36         Node or Graph"))
37     else
38       check_graph_list env tl)
39 | Sast.Undir(v1,v2,dt) | Sast.Dir(v1,v2,dt) ->
40   check_graph_list env tl
41 | Sast.UndirVal(v1,v2,e1,dt) | Sast.DirVal(v1,v2,e1,dt) ->
42   check_graph_list env tl
43 | Sast.BidirVal(e1,v1,v2,e2,dt) ->
44   check_graph_list env tl
45 | _ -> raise (Failure ("type not expected in Graph Def"))
46
47 (* make sure each pair in dict assignment is right type *)
48 let rec check_dict env v_e = function
49 | [] -> ""
50 | hd::tl ->
51   if not((fst v_e = get_sexpr_type (fst hd)) && (snd v_e = get_sexpr_type (
52     snd hd))) then
53     raise (Failure ("assignment expression not of type: " ) )
54   else
55     check_dict env v_e tl
56
57 (* match arguments to a function call to that func's definition *)
58 let rec formal_check s_formal_list s_el =
59   match s_formal_list, s_el with
60   | [], [] -> true
61   | [], _ -> raise (Failure ("incorrect arguments" ) )
62   | _, [] -> raise (Failure ("incorrect arguments" ) )
63   | hd1::tl1, hd2::tl2 -> ((fst hd1) = (get_sexpr_type hd2)) && (formal_check

```

```

    t11 t12)
62
63
64 (* converts Ast.program to Sast.program *)
65 let convert_ast prog env =
66
67 (* convert an Ast.expr object to Sast.expr object *)
68 let rec expr env = function
69 | Ast.NumLiteral(v) -> Sast.NumLiteral(v, Sast.Num)
70 | Ast.StrLiteral(v) -> Sast.StrLiteral(v, Sast.String)
71 | Ast.ListLiteral(el) ->
72   let s_el = List.map (expr env) el in
73   (match el with
74   | [] -> ListLiteral([], List(Void))
75   | x -> let dt = get_sexpr_type (List.hd s_el)
76           in
77           ignore (check_list env dt s_el);
78           ListLiteral(s_el, List(dt))
79   )
80 | Ast.DictLiteral(el) ->
81 (* key_type, elem_type, id [(Hello, 15)] *)
82 let s_el = List.map (fun f -> (expr env (fst f), expr env (snd f))) el in
83 (match s_el with
84 | [] -> DictLiteral([], Dict(Void,Void))
85 | x ->
86   let dt = (get_sexpr_type (fst(List.hd s_el)), get_sexpr_type (snd(List.hd
87     s_el)))
88   in
89   ignore (check_dict env dt s_el);
90   DictLiteral(s_el, Sast.Dict(fst dt, snd dt))
91 )
92 | Ast.Boolean(b) -> Sast.Boolean(b, Sast.Bool)
93 | Ast.Id(v) ->
94   (try
95     Sast.Id(v, find_var v env.var_types) (* uses find_var to determine the
96       type of id *)
97   with
98   | Not_found -> raise (Failure ("undeclared variable: " ^ v))
99   )
100 | Ast.Binop(e1, op, e2) ->
101   let s_e1 = expr env e1 in
102   let s_e2 = expr env e2 in
103   let e1_dt = get_sexpr_type s_e1 in
104   let e2_dt = get_sexpr_type s_e2 in
105   (match op with
106   | Add ->
107     (match e1_dt with
108     | Num ->
109       (match e2_dt with
110       | Num -> Sast.Binop(s_e1, op, s_e2, Sast.Num)
111       | String -> Sast.Binop(s_e1, op, s_e2, Sast.String)
112       | _ -> raise (Failure("wrong type: Num + ? "))
113       )
114     )
115   | String ->

```

```

113     (match e2_dt with
114     | Num -> Sast.Binop(s_e1, op, s_e2, Sast.String)
115     | String -> Sast.Binop(s_e1, op, s_e2, Sast.String)
116     | _ -> raise (Failure("wrong type: String + ? "))
117     )
118 | Graph ->
119     (match e2_dt with
120     | Node -> Sast.Binop(s_e1, op, s_e2, Sast.Graph)
121     | Graph -> Sast.Binop(s_e1, op, s_e2, Sast.Graph)
122     | _ -> raise (Failure("wrong type: Graph + ? "))
123     )
124 | Node ->
125     (match e2_dt with
126     | Node -> Sast.Binop(s_e1, op, s_e2, Sast.Graph)
127     | Graph -> Sast.Binop(s_e1, op, s_e2, Sast.Graph)
128     | _ -> raise (Failure("wrong type: Node + ? "))
129     )
130 | List(dt) ->
131     (match e2_dt with
132     | List(dt) ->
133         if (e1_dt = e2_dt) then
134             Sast.Binop(s_e1, op, s_e2, e1_dt)
135         else
136             raise (Failure("wrong type: List + List<?> "))
137     | _ -> raise (Failure("wrong type: List + ? "))
138     )
139 | _ -> raise (Failure("Expr using + has incompatible types"))
140 )
141 | Sub ->
142     (match e1_dt with
143     | Num ->
144         (match e2_dt with
145         | Num -> Sast.Binop(s_e1, op, s_e2, Sast.Num)
146         | _ -> raise (Failure("wrong type: Num - ? "))
147         )
148     | Graph ->
149         (match e2_dt with
150         | Node -> Sast.Binop(s_e1, op, s_e2, Sast.Graph)
151         | Graph -> Sast.Binop(s_e1, op, s_e2, Sast.Graph)
152         | _ -> raise (Failure("wrong type: Graph - ? "))
153         )
154     | _ -> raise (Failure("Expr using - has incompatible types"))
155     )
156 | Mult | Div ->
157     (match e1_dt with
158     | Num ->
159         (match e2_dt with
160         | Num -> Sast.Binop(s_e1, op, s_e2, Sast.Num)
161         | _ -> raise (Failure("wrong type: Num * or / ? "))
162         )
163     | _ -> raise (Failure("Expr using / has incompatible types"))
164     )
165 | Equal | Neq ->
166     (match e1_dt with

```



```

167 | Num ->
168   (match e2_dt with
169     | Num -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
170     | _ -> raise (Failure("wrong type: Num ==/!= ? "))
171   )
172 | String ->
173   (match e2_dt with
174     | String -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
175     | _ -> raise (Failure("wrong type: String ==/!= ? "))
176   )
177 | Bool ->
178   (match e2_dt with
179     | Bool -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
180     | _ -> raise (Failure("wrong type: Bool ==/!= ? "))
181   )
182 | Void ->
183   (match e2_dt with
184     | Void -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
185     | _ -> raise (Failure("wrong type: Void ==/!= ? "))
186   )
187 | Graph ->
188   (match e2_dt with
189     | Node -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
190     | Graph -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
191     | _ -> raise (Failure("wrong type: Graph ==/!= ? "))
192   )
193 | Node ->
194   (match e2_dt with
195     | Node -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
196     | Graph -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
197     | _ -> raise (Failure("wrong type: Node ==/!= ? "))
198   )
199 | List(dt) ->
200   (match e2_dt with
201     | List(dt) ->
202       if (e1_dt = e2_dt) then
203         Sast.Binop(s_e1, op, s_e2, Sast.Bool)
204       else
205         raise (Failure("wrong type: List ==/!= List<?> "))
206     | _ -> raise (Failure("wrong type: List ==/!= ? "))
207   )
208 | Dict(dtk, dtv) ->
209   (match e2_dt with
210     | Dict(dtk, dtv) ->
211       if (e1_dt = e2_dt) then
212         Sast.Binop(s_e1, op, s_e2, Sast.Bool)
213       else
214         raise (Failure("wrong type binop: Dict ==/!= Dict<?> "))
215     | _ -> raise (Failure("wrong type: Dict ==/!= ? "))
216   )
217 )
218
219 | Less | Leq | Greater | Geq ->
220   (match e1_dt with

```

```

221 | Num ->
222   (match e2_dt with
223     | Num -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
224     | _ -> raise (Failure("wrong type: Num </>/<=/>= ? "))
225   )
226 | String ->
227   (match e2_dt with
228     | String -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
229     | _ -> raise (Failure("wrong type: String </>/<=/>= ? "))
230   )
231 | _ -> raise (Failure("Expr using </>/<=/>= has incompatible types"))
232 )
233
234 | LogAnd | LogOr ->
235   (match e1_dt with
236     | Bool ->
237       (match e2_dt with
238         | Bool -> Sast.Binop(s_e1, op, s_e2, Sast.Bool)
239         | _ -> raise (Failure("wrong type: Bool &&/|| ? "))
240       )
241     | _ -> raise (Failure("Expr using &&/|| has incompatible types"))
242   )
243 )
244 | Ast.Call(f, el) ->
245   let s_el = List.map (expr env) el in
246   (*
247   match range can only take 1 or two args not 0 or 3+ and make sure nums...
248   instead of f try check with the func name
249   if it does exist put the value of the key function name for the map
250   func_types
251   s_formals : (dataType * string) list;
252   range(1,5)
253   [1,2,3,4,5]
254   *)
254   if f = "range" then
255     (let len = List.length el in
256     if (len = 1) then
257       let arg_types = [(Sast.Num, "foo")] in
258       ignore (formal_check arg_types s_el);
259       Sast.Call(f, s_el, Sast.List(Sast.Num))
260     else if (len = 2) then
261       let arg_types = [(Sast.Num, "foo"); (Sast.Num, "foo")] in
262       ignore (formal_check arg_types s_el);
263       Sast.Call(f, s_el, Sast.List(Sast.Num))
264     else
265       raise(Failure("range can only take 1 or 2 args"))
266     )
267   else if f = "print" then Sast.Call(f, s_el, Sast.Void)
268   else if (f = "min" || f = "max") then
269     (try
270       let s_el = List.map (expr env) el in
271       let data_type = get_sexpr_type (List.hd s_el) in
272       let len = List.length el in
273       if (len = 1) then

```

```

274     (match data_type with
275       | Sast.List(dt) -> Sast.Call(f, s_el, dt)
276       | Sast.Dict(dtk, dtv) -> Sast.Call(f, s_el, dtk)
277       | _ -> raise(Failure("member call failed")))
278   else
279     raise(Failure("member call failed"))
280   with
281     Not_found -> raise (Failure("undeclared variable: "))
282 else if f = "len" then
283   (try
284     let s_el = List.map (expr env) el in
285     let data_type = get_sexpr_type (List.hd s_el) in
286     let len = List.length el in
287     if (len = 1) then
288       (match data_type with
289         | Sast.List(dt) -> Sast.Call(f, s_el, Sast.Num)
290         | Sast.Dict(dtk, dtv) -> Sast.Call(f, s_el, Sast.Num)
291         | _ -> raise(Failure("member call failed")))
292     else
293       raise(Failure("member call failed"))
294     with
295       Not_found -> raise (Failure("undeclared variable: "))
296 else
297   (
298     let fdecl = find_var f env.func_obj in
299     ignore (formal_check fdecl.s_formals s_el);
300     let rtype = fdecl.s_rtype in
301     Sast.Call(f, s_el , rtype)
302   )
303 | Ast.Access(e1, e2) ->
304   let s_e1 = expr env e1 in (* func rec until it knows datatype -- sast
305     version of ast expr e *)
306   let e1_dt = get_sexpr_type s_e1 in
307   let s_e2 = expr env e2 in (* func rec until it knows datatype -- sast
308     version of ast expr e *)
309   let e2_dt = get_sexpr_type s_e2 in
310   (try (*sees if variable defined*)
311     (match e1_dt with
312       | List(dt) ->
313         (match e2_dt with
314           | Sast.Num -> Sast.Access(s_e1, s_e2, dt)
315           | _ -> raise (Failure "expr to access list should be Num")
316         )
317       | Dict(dk, dv) ->
318         if (e2_dt = dk) then
319           Sast.Access(s_e1, s_e2, dv)
320         else
321           raise (Failure("wrong type ast.access: sexpr type != Dict<?>,
322             expected " ^ (type_to_str dk) ^ " got " ^ (type_to_str e2_dt)
323             ^ " life sucks"))
324       | _ -> raise (Failure("must use Dict or List with access!"))
325     )
326   with
327     | Not_found -> raise (Failure("undeclared variable: "))

```

```

324 );
325 | Ast.MemberCall(e, m, el) ->
326 (
327   let s_e = expr env e in
328   let e_dt = get_sexpr_type s_e in
329   let num_args = List.length el in
330   let s_el = List.map (expr env) el in
331
332   match m with
333   | "enqueue" | "push" ->
334     if num_args != 1 then raise (Failure ("enqueue/push requires 1 arg"))
335     else
336       ignore((match e_dt with
337         | List(_) -> ignore()
338         | _ -> raise (Failure ("enqueue/push error: not a list"))
339       ));
340       ignore(check_list env (get_list_type e_dt) s_el); (* check that the
341         arg is the type in the list *)
342       Sast.MemberCall(s_e, m, s_el, Sast.Void)
343   | "dequeue" | "pop" ->
344     if num_args != 0 then raise (Failure ("dequeue/pop requires 0 args"))
345     else Sast.MemberCall(s_e, m, s_el, Sast.Void)
346   | "peek" ->
347     if num_args != 0 then raise (Failure ("peek requires 0 args"))
348     else Sast.MemberCall(s_e, m, s_el, get_list_type e_dt)
349   | "oute" | "ine" ->
350     if num_args != 0 then raise (Failure ("oute/ine requires 0 args"))
351     else Sast.MemberCall(s_e, m, s_el, Dict(Node, Num))
352   | "val" ->
353     if num_args != 0 then raise (Failure ("val requires 0 args"))
354     else Sast.MemberCall(s_e, m, s_el, String)
355   | "remove" ->
356     if num_args != 1 then raise (Failure ("remove requires 1 arg"))
357     else
358       ignore((match e_dt with
359         | Dict(_) -> ignore()
360         | _ -> raise (Failure ("remove error: not a dict"))
361       ));
362       ignore(check_list env (fst (get_dict_type e_dt)) s_el); (* check that
363         the arg is the type in the list *)
364       Sast.MemberCall(s_e, m, s_el, Sast.Void)
365   | _ -> raise (Failure ("no member function: " ^ m))
366 )
367 | Ast.Undir(v1, v2) ->
368   (*check if v1 and v2 exist *)
369   (try (*sees if variable defined*)
370     let v1_e = find_var v1 env.var_types in
371     if v1_e = Sast.Node then
372       let v2_e = find_var v2 env.var_types in
373       if v2_e = Sast.Node then
374         Sast.Undir(v1, v2, Sast.Void)
375       else raise (Failure("Wrong variable types"))
376     else
377       raise (Failure("Wrong variable types"))

```

```

376     with
377     | Not_found -> raise (Failure("undeclared variable: "))
378
379 | Ast.Dir(v1, v2) ->
380     (try (*sees if variable defined*)
381         let v1_e = find_var v1 env.var_types in
382         if v1_e = Sast.Node then
383             let v2_e = find_var v2 env.var_types in
384             if v2_e = Sast.Node then
385                 Sast.Dir(v1, v2, Sast.Void)
386             else raise (Failure("undeclared variable: "))
387         else raise (Failure("undeclared variable: "))
388     with
389     | Not_found -> raise (Failure("undeclared variable: "))
390 | Ast.BidirVal(w1, v1, v2, w2) ->
391     (try (*sees if variable defined*)
392         if find_var v1 env.var_types = Sast.Graph then
393             if find_var v2 env.var_types = Sast.Graph then
394                 let s_w1 = expr env w1 in
395                 if get_sexpr_type s_w1 = Sast.Num then
396                     let s_w2 = expr env w2 in
397                     if get_sexpr_type s_w2 = Sast.Num then
398                         Sast.BidirVal(expr env w1, v1, v2, expr env w2, Sast.Void)
399                     else
400                         raise (Failure("undeclared variable: "))
401                 else
402                     raise (Failure("undeclared variable: "))
403             else
404                 raise (Failure("undeclared variable: "))
405         else
406             raise (Failure("undeclared variable: "))
407     with
408     | Not_found -> raise (Failure("undeclared variable: "))
409 | Ast.UndirVal(v1, v2, w) ->
410     (try (*sees if variable defined*)
411         if find_var v1 env.var_types = Sast.Node then
412             if find_var v2 env.var_types = Sast.Node then
413                 let s_w = expr env w in
414                 if get_sexpr_type s_w = Sast.Num then
415                     Sast.UndirVal(v1, v2, expr env w, Sast.Void)
416                 else
417                     raise (Failure("undeclared variable: "))
418             else
419                 raise (Failure("undeclared variable: "))
420         else
421             raise (Failure("undeclared variable: "))
422     with
423     | Not_found -> raise (Failure("undeclared variable: "))
424 | Ast.DirVal(v1, v2, w) ->
425     (try
426         if find_var v1 env.var_types = Sast.Node then
427             if find_var v2 env.var_types = Sast.Node then
428                 let s_w = expr env w in
429                 if get_sexpr_type s_w = Sast.Num then

```

```

430     Sast.DirVal(v1, v2, expr env w, Sast.Void)
431   else
432     raise (Failure("undeclared variable: "))
433   else
434     raise (Failure("undeclared variable: "))
435   else
436     raise (Failure("undeclared variable: "))
437   with
438     | Not_found -> raise (Failure("undeclared variable: "))
439 | Ast.NoOp(v) -> Sast.NoOp(v, Sast.Void)
440 | Ast.Noexpr -> Sast.Noexpr
441 in
442
443 (* convert an Ast.stmt object to Sast.stmt object *)
444 let rec stmt env = function
445 | Ast.Block(sl) -> Sast.Block(List.map (fun s -> stmt env s) sl)
446 | Ast.Expr(e) -> Sast.Expr(expr env e)
447 | Ast.Vdecl(dt, id) ->
448   (try
449     ignore (StringMap.find id !(List.hd env.var_types));
450     raise (Failure ("variable already declared in local scope: " ^ id))
451   with | Not_found -> (List.hd env.var_types) := StringMap.add id (
452     str_to_type dt) !(List.hd env.var_types); (* add type map *)
453     (List.hd env.var_inds) := StringMap.add id (find_max_index !(List.
454       hd env.var_inds)+1) !(List.hd env.var_inds); (* add index
455       mapping *)
456   | Failure(f) -> raise (Failure (f) )
457   );
458   Sast.Vdecl(str_to_type dt, id)
459 | Ast.ListDecl(dt, id) -> (* Sast.ListDecl(str_to_type dt, v) *)
460   let vtype = Sast.List(str_to_type dt) in
461   (try
462     ignore (StringMap.find id !(List.hd env.var_types));
463     raise (Failure ("variable already declared in local scope: " ^ id))
464   with | Not_found -> (List.hd env.var_types) := StringMap.add id vtype !(
465     List.hd env.var_types); (* add type map *)
466     (List.hd env.var_inds) := StringMap.add id (find_max_index !(List.
467       hd env.var_inds)+1) !(List.hd env.var_inds); (* add index
468       mapping *)
469   | Failure(f) -> raise (Failure (f))
470   );
471   Sast.Vdecl(vtype, id)
472 | Ast.DictDecl(dtk, dtv, id) -> (*Sast.DictDecl(str_to_type dtk, str_to_type
473   dtv, v)*)
474   let vtype = Sast.Dict(str_to_type dtk, str_to_type dtv) in
475   (try
476     ignore (StringMap.find id !(List.hd env.var_types));
477     raise (Failure ("variable already declared in local scope: " ^ id))
478   with | Not_found -> (List.hd env.var_types) := StringMap.add id vtype !(
479     List.hd env.var_types);
480     (List.hd env.var_inds) := StringMap.add id (find_max_index !(List.
481       hd env.var_inds)+1) !(List.hd env.var_inds);
482   | Failure(f) -> raise (Failure (f) )
483   );

```

```

475   Sast.Vdecl(vtype, id)
476 | Ast.Assign(v, e) ->          (* checks that the var and expression are of
   the same type, then converts to Sast.Assign *)
477   let s_e = expr env e in      (* func rec until it knows datatype -- sast
   version of ast expr e *)
478   let s_v = expr env v in
479   let e_dt = get_sexpr_type s_e in (* data type of that sast expr with
   function get_sexpr_type*)
480   let v_dt = get_sexpr_type s_v in
481   if not (v_dt = e_dt)
482   then raise (Failure ("assignment expression not of type: " ^ (type_to_str
   v_dt) ))
483   else Sast.Assign(s_v, s_e, Sast.Void)
484 | Ast.AccessAssign(e1, e2, e3) ->
485   let s_e1 = expr env e1 in    (* func rec until it knows datatype -- sast
   version of ast expr e *)
486   let e1_dt = get_sexpr_type s_e1 in
487   let s_e2 = expr env e2 in    (* func rec until it knows datatype -- sast
   version of ast expr e *)
488   let e2_dt = get_sexpr_type s_e2 in
489   let s_e3 = expr env e3 in
490   let e3_dt = get_sexpr_type s_e3 in
491   (try                          (*sees if variable defined*)
492     (match e1_dt with
493       List(dt) ->
494         (match e2_dt with
495           | Sast.Num ->
496             if (e3_dt = dt) then
497               Sast.AccessAssign(s_e1, s_e2, s_e3, Sast.Void)
498             else
499               raise (Failure("AccessAssign: Assigning wrong type"))
500           | _ -> raise (Failure ("expr to access list should be Num")))
501         )
502     | Dict(dk,dv) ->
503       if (e2_dt = dk) then
504         if(e3_dt = dv) then
505           Sast.AccessAssign(s_e1, s_e2, s_e3, Sast.Void)
506         else
507           raise(Failure("AccessAssign: mismatched Value data type"))
508       else
509         raise (Failure("wrong type accessassign: Dict != Dict<?> "))
510     | _ -> raise (Failure("must use Dict or List with access!"))
511   )
512   )
513   with
514   | Not_found -> raise (Failure("undeclared variable: "))
515   );
516
517 | Ast.NodeDef(v, e) -> (* (node id, what goes inside parens) of item *)
518   (try
519     let v_e = (find_var v env.var_types) in
520     let s_e = expr env e in
521     let e_dt = get_sexpr_type s_e in
522     if v_e = Sast.Node then

```

```

523     Sast.NodeDef(v, s_e, e_dt)
524     else raise (Failure ("Node Def failure"))
525   with
526     | Not_found -> raise (Failure("Node Def failure"))
527 | Ast.GraphDef(v, el) ->
528   (try
529     let s_el = List.map (expr env) el in
530     ignore(check_graph_list env s_el);
531     Sast.GraphDef(v, s_el)
532   with
533     Not_found -> raise (Failure ("GraphDef issue")))
534
535 | Ast.Return(e) ->
536   (try
537     let s_e = expr env e in
538     (match get_sexpr_type s_e with
539     | Sast.Node -> Sast.Return(s_e, Sast.Node)
540     | Sast.Num -> Sast.Return(s_e, Sast.Num)
541     | Sast.String -> Sast.Return(s_e, Sast.String)
542     | Sast.Bool -> Sast.Return(s_e, Sast.Bool)
543     | Sast.Graph -> Sast.Return(s_e, Sast.Graph)
544     | Sast.List(s_dt) -> Sast.Return(s_e, Sast.List(s_dt))
545     | Sast.Dict(dtk, dtv) -> Sast.Return(s_e, Sast.Dict(dtk, dtv))
546     | Sast.Void -> Sast.Return(s_e, Sast.Void))
547   with
548     Not_found -> raise (Failure ("return issue")))
549 | Ast.If(cond, s1, s2) ->
550   (try
551     let s_cond = expr env cond in
552     if (get_sexpr_type s_cond) = Sast.Bool then
553       Sast.If(expr env cond, stmt env s1, stmt env s2)
554     else
555       raise (Failure ("if issue"))
556   with
557     Not_found -> raise (Failure ("return issue")))
558 | Ast.For(v, e, s1) ->
559   (* iterable expr must have var which already has been declared *)
560   let s_e = expr env e in
561   let e_dt = get_sexpr_type s_e in
562   (match e_dt with
563   | Sast.List(dt) ->
564     (try
565       ignore(find_var v env.var_inde);
566       ignore(raise (Failure ("'" ^ v ^ "' has already been declared")))
567     with
568       | Not_found -> ignore()
569       | Failure(f) -> raise (Failure f)
570     );
571     (* add the temp var to the symbol table *)
572     (List.hd env.var_types) := StringMap.add v dt !(List.hd env.var_types);
573     (* add type map *)
574     (List.hd env.var_inde) := StringMap.add v (find_max_index !(List.hd env.
575     var_inde)+1) !(List.hd env.var_inde); (* add index mapping *)
576     Sast.For(v, s_e, List.map (fun s -> stmt env s) s1)

```



```

575 | Dict(dtk, dtv) ->
576   (try
577     ignore(find_var v env.var_inds);
578     ignore(raise (Failure ("'" ^ v ^ "' has already been declared")))
579   with
580   | Not_found -> ignore()
581   | Failure(f) -> raise (Failure f)
582   );
583   (* add the temp var to the symbol table *)
584   (List.hd env.var_types) := StringMap.add v dtk !(List.hd env.var_types);
585   (* add type map *)
586   (List.hd env.var_inds) := StringMap.add v (find_max_index !(List.hd env.
587     var_inds)+1) !(List.hd env.var_inds); (* add index mapping *)
588   Sast.For(v, s_e, List.map (fun s -> stmt env s) sl)
589 | Graph ->
590   (try
591     ignore(find_var v env.var_inds);
592     ignore(raise (Failure ("'" ^ v ^ "' has already been declared")))
593   with
594   | Not_found -> ignore()
595   | Failure(f) -> raise (Failure f)
596   );
597   (* add the temp var to the symbol table *)
598   (List.hd env.var_types) := StringMap.add v Sast.Node !(List.hd env.
599     var_types); (* add type map *)
600   (List.hd env.var_inds) := StringMap.add v (find_max_index !(List.hd env.
601     var_inds)+1) !(List.hd env.var_inds); (* add index mapping *)
602   Sast.For(v, s_e, List.map (fun s -> stmt env s) sl)
603 | Node ->
604   (try
605     ignore(find_var v env.var_inds);
606     ignore(raise (Failure ("'" ^ v ^ "' has already been declared")))
607   with
608   | Not_found -> ignore()
609   | Failure(f) -> raise (Failure f)
610   );
611   (* add the temp var to the symbol table *)
612   (List.hd env.var_types) := StringMap.add v Sast.Node !(List.hd env.
613     var_types); (* add type map *)
614   (List.hd env.var_inds) := StringMap.add v (find_max_index !(List.hd env.
615     var_inds)+1) !(List.hd env.var_inds); (* add index mapping *)
616   Sast.For(v, s_e, List.map (fun s -> stmt env s) sl)
617 | _ -> raise(Failure("Trying to for loop an expr that doesnt return an
618   iterable"))
619 )
620 | Ast.While(cond, sl) ->
621   (try
622     let s_cond = expr env cond in
623     if (get_sexpr_type s_cond) = Sast.Bool then
624       Sast.While(expr env cond, List.map (fun s -> stmt env s) sl)
625     else
626       raise (Failure ("while issue"))
627   )

```

```

622   with
623     Not_found -> raise (Failure ("while issue"))
624 | Ast.Fdecl(func) -> (*Fdecl of func_decl and *)
625   (try
626     (* add formal variables to local scope variable maps *)
627     let fname = func.fname in
628     let formals = List.map (fun (dt, v) -> (f_dt_to_type dt, v)) func.formals
629       in
630     let rtype = f_dt_to_type func.rtype in
631
632     (* add this function to symbol table *)
633     let dummy_func_obj = {
634       Sast.s_fname = fname;
635       Sast.s_rtype = rtype;
636       Sast.s_formals = [];
637       Sast.s_body = []
638     }
639     in
640     (List.hd env.func_obj) := StringMap.add fname dummy_func_obj !(List.hd env.
641       func_obj);
642     (List.hd env.func_inds) := StringMap.add fname (find_max_index !(List.hd env
643       .func_inds)+1) !(List.hd env.func_inds); (* add index map *)
644
645     let func_env = {
646       var_inds = ref StringMap.empty :: env.var_inds; (* var names to
647         indices ex. x -> 1 so that we can just refer to it as v1 *)
648       var_types = ref StringMap.empty :: env.var_types; (* maps a var name
649         to its type ex. x -> num *)
650       func_inds = env.func_inds; (* func names to indices ex. x -> 1 so
651         that we can just refer to it as f1 *)
652       func_obj = env.func_obj;
653       return_type = rtype; (* what should the return type be of
654         the current scope *)
655     }
656     in
657
658     let rec fmls_adder env = function
659     | [] -> ignore()
660     | hd :: tl ->
661       (List.hd env.var_types) := StringMap.add (snd hd) (fst hd) !(List.hd env.
662         var_types);
663       (List.hd env.var_inds) := StringMap.add (snd hd) (find_max_index !(List.
664         hd env.var_inds)+1) !(List.hd env.var_inds); (* add index map *)
665       ignore(fmls_adder env tl)
666     in
667
668     let populated_fdecl = {
669       Sast.s_fname = func.fname;
670       Sast.s_rtype = rtype;
671       Sast.s_formals = formals;
672       Sast.s_body = List.map (fun s -> stmt func_env s) func.body
673     }
674     in

```

```

667 (List.hd env.func_obj) := StringMap.add fname populated_fdecl !(List.hd env.
    func_obj); (* replace the dummy fobj with the evaluated one *)
668 Sast.Fdecl(populated_fdecl)
669 with
670     Not_found -> raise (Failure ("fdecl issue"))
671 in
672
673 { Sast.s_cmds = List.map (fun s -> stmt env s) prog.cmds }
674
675 (* get printf fmt string for Sast.dataType types *)
676 let dt_fmt = function
677 | Sast.Num -> "%f"
678 | Sast.String -> "%s"
679 | Sast.Bool -> "%d"
680 | Sast.Graph -> "" (* TODO *)
681 | Sast.Node -> "" (* TODO *)
682 | Sast.Dict(dtk, dtv) -> "" (* TODO *)
683 | Sast.List(dt) -> "" (* TODO *)
684 | Sast.Void -> "" (* TODO *)

```

Listing 20: typeConverter.ml

## A.6 analyzer.ml

```

1 (* converts dots SAST to C AST *)
2 open Ast
3 open Sast
4 open Translate
5
6
7 module StringMap = Map.Make(String)
8 type s_program = { s_globals : s_stmt list; s_main: s_stmt list; s_funcs :
    s_fdecl list; }
9
10 (*
11 DEALING WITH AUTOMATIC RESULT VARS:
12
13 Step 1: After every "let x = ...." where "translate_expr env ..." is called,
14 create a variable to hold the name of the result variable from that
15 call
16 ex. let e1_result = "v" ^ string_of_int (find_max_index !(List.hd env.
17 var_inds)) in
18
19 Step 2: After the end of all "translate_expr env ..." calls (i.e. when that
20 function
21 is no longer called), create a new auto_var to hold the result of the
22 current
23 function's translation.
24 ex. let result_var = "v" ^ string_of_int(create_auto env "" (dt)) in
25
26 Step 3: Output a Block([]) that contains:
27 a. a Vdecl object for result_var
28 b. the C code that corresponds to the current expr's translation

```

```

26     c. an Assign call that assigns the result of part b. to result_var
27
28     ex. for if the result of e1 is the result of your expression
29         Block([Vdecl(..., result_var);
30             c_e1;
31             Assign(Id(..., result_var), Id(..., e1_result))
32         ])
33 *)
34
35 (* where symbol tables are stored *)
36 type translation_env = {
37     var_inds : int StringMap.t ref list; (* var names to indices ex. x
38         -> 1 so that we can just refer to it as v1 *)
39     var_types : Sast.dataType StringMap.t ref list; (* maps a var name to
40         its type ex. x -> num *)
41     func_inds : int StringMap.t ref list; (* func names to indices ex. x
42         -> 1 so that we can just refer to it as f1 *)
43     func_obj : Sast.s_fdecl StringMap.t ref list; (* maps a func name to
44         its return type *)
45     return_type : Sast.dataType; (* what should the return type
46         be of the current scope *)
47 }
48
49 (*
50 @param *implicit* := list of Sast.stmts to sort
51 @param sifted := a struct that contains the globals,
52     regular stmts, and func decls that have been sorted so far
53 *)
54 let get_list_type = function
55 | Sast.List(dt) -> dt
56 | _ -> raise (Failure("wrong type: not a list"))
57
58 let get_dict_type = function
59 | Sast.Dict(dtk, dtv) -> dtk
60 | _ -> raise (Failure("wrong type: not a list"))
61
62 (*
63 extract an SAST into a form more presentable to C AST
64 3 pieces: global variable declarations, function definitions,
65 other statements
66 *)
67 let rec stmt_sifter sifted = function
68 | [] -> sifted
69 | hd :: tl -> (match hd with
70 | Sast.Vdecl(dt, id) ->
71     stmt_sifter {s_globals = hd :: sifted.s_globals;
72                 s_main = sifted.s_main;
73                 s_funcs = sifted.s_funcs} tl
74 | Sast.Assign(v, e, dt) ->
75     stmt_sifter {s_globals = sifted.s_globals;
76                 s_main = hd :: sifted.s_main;
77                 s_funcs = sifted.s_funcs} tl
78 | Sast.Expr(e) ->
79     stmt_sifter {s_globals = sifted.s_globals;

```

```

75         s_main = hd :: sifted.s_main;
76         s_funcs = sifted.s_funcs} tl
77 | Sast.NodeDef(id, e, dt) ->
78     stmt_sifter {s_globals = sifted.s_globals;
79     s_main = hd :: sifted.s_main;
80     s_funcs = sifted.s_funcs} tl
81 | Sast.AccessAssign(sel, se2, se3, dt) ->
82     stmt_sifter {s_globals = sifted.s_globals;
83     s_main = hd :: sifted.s_main;
84     s_funcs = sifted.s_funcs} tl
85 | Sast.GraphDef(id, el) ->
86     stmt_sifter {s_globals = sifted.s_globals;
87     s_main = hd :: sifted.s_main;
88     s_funcs = sifted.s_funcs} tl
89 | Sast.Return(e, dt) ->
90     stmt_sifter {s_globals = sifted.s_globals;
91     s_main = hd :: sifted.s_main;
92     s_funcs = sifted.s_funcs} tl
93 | Sast.Block(sl) ->
94     let sifted_sl = stmt_sifter {s_globals = []; s_main = [];
95     s_funcs = []} sl in
96     stmt_sifter {s_globals = sifted_sl.s_globals @ sifted.s_globals;
97     s_main = Block(sifted_sl.s_main) :: sifted.s_main;
98     s_funcs = sifted_sl.s_funcs @ sifted.s_funcs} tl
99 | Sast.If(cond, sl, s2) ->
100     let sifted_sl = stmt_sifter {s_globals = []; s_main = [];
101     s_funcs = []} [sl] in
102     let sifted_s2 = stmt_sifter {s_globals = []; s_main = [];
103     s_funcs = []} [s2] in
104     let tmp = {s_globals = sifted_sl.s_globals @ sifted.s_globals;
105     s_main = sifted_sl.s_main @ sifted.s_main;
106     s_funcs = sifted_sl.s_funcs @ sifted.s_funcs} in
107     stmt_sifter {s_globals = sifted_s2.s_globals @ tmp.s_globals;
108     s_main = sifted_s2.s_main @ tmp.s_main;
109     s_funcs = sifted_s2.s_funcs @ sifted.s_funcs} tl
110 | Sast.For(tmp, iter, sl) ->
111     let sifted_sl = stmt_sifter {s_globals = []; s_main = [];
112     s_funcs = []} sl in
113     stmt_sifter {s_globals = sifted_sl.s_globals @ sifted.s_globals;
114     s_main = For(tmp, iter, sifted_sl.s_main) :: sifted.
115     s_main;
116     s_funcs = sifted_sl.s_funcs @ sifted.s_funcs} tl
117 | Sast.While(cond, sl) ->
118     let sifted_sl = stmt_sifter {s_globals = []; s_main = [];
119     s_funcs = []} sl in
120     stmt_sifter {s_globals = sifted_sl.s_globals @ sifted.s_globals;
121     s_main = While(cond, sifted_sl.s_main) :: sifted.s_main

```

```

122
123 let mappings = [("ine", Sast.Node); ("oute", Sast.Node); ("value", Sast.Node);
    ("nodes", Sast.Graph)]
124 let mem_vars = List.fold_left (fun m (k, v) -> StringMap.add k v m) StringMap.
    empty mappings
125
126 (* returns list of tuples mapping each elem of a list to consecutive
127 numbers starting from n and incrementing n by stride for each elem *)
128 let rec enum stride n = function
129     [] -> []
130     | hd::tl -> (n, hd) :: enum stride (n+stride) tl
131
132     (* val string_map_pairs StringMap 'a -> (int * 'a) list -> StringMap 'a *)
133     (* takes list of tuples (value, key) and adds them to the given map *)
134 let string_map_pairs map pairs =
135     List.fold_left (fun m (i, n) -> StringMap.add n i m) map pairs
136
137 let find_max_index map =
138     let bindings = StringMap.bindings map in
139     let rec max cur = function
140         | [] -> cur
141         | hd :: tl -> if snd hd > cur then max (snd hd) tl else max cur tl
142     in
143     max 0 bindings
144
145
146     (*
147     returns the value associated with a given key,
148     traversing through the list of maps until it finds the
149     first occurrence of the key, or raises an error if none of
150     the maps contain that key
151
152     value: type
153     key: variable name
154
155     intended for things like: finding the type of a variable
156     *)
157 let find_var var map_list =
158     let rec finder var = function
159         | m :: tl ->
160             (try StringMap.find var !m
161             with
162             | Not_found -> finder var tl)
163         | [] -> raise (Not_found )
164     in
165     finder var map_list
166
167 let str_to_type = function
168     | "num" -> Sast.Num
169     | "string" -> Sast.String
170     | "bool" -> Sast.Bool
171     | "graph" -> Sast.Graph
172     | "node" -> Sast.Node
173     | "dict" -> Sast.Dict(Sast.Void, Sast.Void)

```

```

174 | "list" -> Sast.List(Sast.Void)
175 | "void" -> Sast.Void
176 | x -> raise (Failure ("unknown type: " ^ x))
177
178 (* for function args only, pass in a special type *)
179 let f_dt_to_type = function
180 | Ast.Basic(dt) -> str_to_type dt
181 | Ast.List(dt) -> Sast.List(str_to_type dt)
182 | Ast.Dict(dtk, dtv) -> Sast.Dict(str_to_type dtk, str_to_type dtv)
183
184 (* converts a datatype to a string *)
185 let rec type_to_str = function
186 | Sast.Num -> "num"
187 | Sast.String -> "string"
188 | Sast.Bool -> "bool"
189 | Sast.Graph -> "graph"
190 | Sast.Node -> "node"
191 | Sast.Dict(dtk, dtv) -> "dict <" ^ type_to_str dtk ^ ", " ^ type_to_str
    dtv ^ ">"
192 | Sast.List(dt) -> "list <" ^ type_to_str dt ^ ">"
193 | Sast.Void -> "void"
194
195 let rec expr_type_str = function
196 | Sast.NumLiteral(v, dt) -> "NumLiteral"
197 | Sast.StrLiteral(v, dt) -> "StrLiteral"
198 | Sast.ListLiteral(el, dt) -> "ListLiteral"
199 | Sast.DictLiteral(kvl, dt) -> "DictLiteral"
200 | Sast.Boolean(v, dt) -> "Boolean"
201 | Sast.Id(v, dt) -> "Id"
202 | Sast.Binop(e1, op, e2, dt) -> "Binop"
203 | Sast.Call(v, el, dt) -> "Call"
204 | Sast.Access(v, e, dt) -> "Access"
205 | Sast.MemberCall(v, m, el, dt) -> "MemberCall"
206 | Sast.Undir(v1, v2, dt) -> "Undir"
207 | Sast.Dir(v1, v2, dt) -> "Dir"
208 | Sast.UndirVal(v1, v2, w, dt) -> "UndirVal"
209 | Sast.DirVal(v1, v2, w, dt) -> "DirVal"
210 | Sast.BidirVal(w1, v1, v2, w2, dt) -> "BidirVal"
211 | Sast.NoOp(v, dt) -> "NoOp"
212 | Sast.Noexpr -> "Noexpr"
213 (* returns the datatype of an Sast expressions *)
214 let get_sexpr_type = function
215 | Sast.NumLiteral(v, dt) -> dt
216 | Sast.StrLiteral(v, dt) -> dt
217 | Sast.ListLiteral(el, dt) -> dt
218 | Sast.DictLiteral(kvl, dt) -> dt
219 | Sast.Boolean(v, dt) -> dt
220 | Sast.Id(v, dt) -> dt
221 | Sast.Binop(e1, op, e2, dt) -> dt
222 | Sast.Call(v, el, dt) -> dt
223 | Sast.Access(v, e, dt) -> dt
224 | Sast.MemberCall(v, m, el, dt) -> dt
225 | Sast.Undir(v1, v2, dt) -> Sast.Void
226 | Sast.Dir(v1, v2, dt) -> Sast.Void

```

```

227 | Sast.UndirVal(v1, v2, w, dt) -> Sast.Void
228 | Sast.DirVal(v1, v2, w, dt) -> Sast.Void
229 | Sast.BidirVal(w1, v1, v2, w2, dt) -> Sast.Void
230 | Sast.NoOp(v, dt) -> Sast.Void
231 | Sast.Noexpr -> Sast.Void
232
233 (* ***** *)
234 (* TRANSLATES AN SAST *)
235 (* ***** *)
236 (* determines whether a num string is an Int or a Float *)
237 let num_type num_str =
238   let numregex = Str.regexp "-?[0-9]+$"
239   in
240   if Str.string_match numregex num_str 0 then Int else Float
241
242 let rec dt_to_ct = function
243 | Sast.Num -> Float
244 | Sast.String -> Cstring
245 | Sast.Bool -> Int
246 | Sast.Graph -> Graph (* TODO *)
247 | Sast.Node -> Node (* TODO *)
248 | Sast.List(dt) -> List(dt_to_ct dt) (* TODO *)
249 | Sast.Dict(dtk, dtv) -> Ptr(Ptr(Entry)) (* TODO *)
250 | Sast.Void -> Void
251
252 (* the meat of the compiler *)
253 (* actually converts Sast objects into strings of C code *)
254 let translate (env, sast_prg) =
255
256
257 (* Automatic Variables *)
258 (* certain translations require creating vars automatically
259 keep track of all auto vars created so far, so that we
260 don't repeat auto vars in C
261 *)
262
263 (* maps the given key to the next available int index
264 returns the index/number that the key was mapped to
265
266 Note: a key is a dots variable name
267 ex. "key" : 3 := means that var "key" represents auto var "a3"
268 *)
269 let create_auto env key dt =
270   let ind = (find_max_index !(List.hd env.var_inds)+1) in
271   let var_name = (match key with
272     | "" -> "v" ^ string_of_int(ind)
273     | _ -> key
274   ) in
275   (List.hd env.var_types) := StringMap.add var_name dt !(List.hd env.
276     var_types); (* add type map *)
277   (List.hd env.var_inds) := StringMap.add var_name ind !(List.hd env.
278     var_inds); (* add index map *)
279   ind
280 in

```



```

279
280 (*
281 C equivalent:
282 char str[50];
283 int len;
284
285 strcpy(str, "This is tutorialspoint.com");
286 len = strlen(str);
287 *)
288 let string_len c_v =
289 let cdt1 = Translate.get_cexpr_type c_v in
290 if cdt1 = Cstring then
291 let auto_var = "v" ^ string_of_int(create_auto env "" (Sast.Num)) in
292 (auto_var, Block([
293 Vdecl(Int, auto_var);
294 Expr(Assign(Id(Int, auto_var),
295 Call(Int, "strlen", [c_v]))]))
296 else
297 raise (Failure("only possible with string "))
298 in
299
300 (* C requires special handling of string concatenation *)
301 let string_concat c_v1 c_v2 =
302 let cdt2 = Translate.get_cexpr_type c_v2 in
303
304 let len_c1 = ((string_len c_v1)) in
305 let len_c2 = ((string_len c_v2)) in
306 let len_new = Assoc(Binop(Int, Id(Int, (fst len_c1)), Add, Id(Int, (fst
len_c2)))) in
307
308 let auto_var = "v" ^ string_of_int(create_auto env "" (Sast.String)) in
309 if cdt2 = Cstring then
310 (auto_var,
311 Block([
312 (snd len_c1);
313 (snd len_c2);
314 Vdecl(Cstring, auto_var);
315 Expr(Assign(
316 Id(Cstring, auto_var),
317 Call(Ptr(Void),
318 "malloc",
319 [Binop(Int, Call(Int, "sizeof", [Id(Void, "int")]), Mult, len_new
))]
))]
320 ));
321 Expr(Call(Void,
322 "strcpy",
323 [Id(Cstring, auto_var);
324 c_v2]));
325 Expr(Call(Void,
326 "strcat",
327 [Id(Cstring, auto_var);
328 c_v1]))
329 ]))
330 else

```

```

331   raise (Failure("only accesible for strings"))
332
333 in
334
335 let string_of_stmt c_v =
336   let cdt = Translate.get_cexpr_type c_v in
337   let s_dt = Translate.type_to_str cdt in
338   let auto_var = "v" ^ string_of_int(create_auto env "" (Sast.String)) in
339   (match cdt with
340     | Int ->
341       (auto_var, Block([
342         Vdecl(Cstring, auto_var);
343         Expr(Assign(Id(Cstring, auto_var),
344           Call(Ptr(Void),
345             "malloc",
346             [Binop(Int,
347               Call(Int, "sizeof", [Id(Void, "char")]),
348               Mult,
349               Literal(Int, "400"))
350             ] )
351           ));
352         Call(Void,
353           "itoa",
354           [c_v;
355             Id(Cstring, auto_var);
356             Literal(Int, "10")
357           ])
358       ]))
359     | Float ->
360       (auto_var, Block([
361         Vdecl(Cstring, auto_var);
362         Expr(Assign(Id(Cstring, auto_var),
363           Call(Ptr(Void),
364             "malloc",
365             [Binop(Int,
366               Call(Int, "sizeof", [Id(Void, "char")]),
367               Mult,
368               Literal(Int, "400"))
369             ] )
370           ));
371         Expr(Call(Void,
372           "sprintf",
373           [Id(Void, auto_var);
374             Literal(Cstring, "%d.%02u");
375             Cast(Int, c_v);
376             Cast(Int,
377               (Binop(Float,
378                 (Binop(Float, c_v, Sub, Cast(Int, c_v))),
379                 Mult,
380                 Literal(Int, "100"))
381             ))
382         ]))
383       ]))
384     | _ -> raise (Failure ("cannot convert type to cstring: " ^ s_dt))

```

```

385 in
386
387
388 let rec build_args args = function
389 | [] -> args
390 | hd :: tl ->
391   let arg_cstmts = translate_expr env hd in
392   let arg_result = "v" ^ string_of_int (find_max_index !(List.hd env.var_inds
393     )) in (* result of arg translation *)
394   let arg_type = dt_to_ct (get_sexpr_type hd) in
395   build_args ((arg_cstmts, Deref(arg_type, Id(Ptr(arg_type), arg_result))) ::
396     args) tl
397 and
398
399 translate_expr env = function
400 | Sast.NumLiteral(l, dt) ->
401   let result_var = "v" ^ string_of_int(create_auto env "" (dt)) in
402   Block([ Vdecl(Ptr(Float), result_var);
403     Expr(Assign(Id(Ptr(Float), result_var),
404       Call(Ptr(Void), "malloc", [ Call(Int, "sizeof", [Id(Void, "
405         float" ] ) ] )
406     ));
407     Expr(Assign( Deref(Float, Id( Ptr(Float), result_var)), Literal(
408       Float, l)))
409   ])
410 | Sast.Boolean(b, dt) ->
411   let bool_val = if b = Ast.True then Literal(Int, "1") else Literal(Int,
412     "0") in
413
414   let result_var = "v" ^ string_of_int(create_auto env "" (dt)) in
415   Block([ Vdecl(Ptr(Int), result_var);
416     Expr(Assign(Id(Ptr(Int), result_var),
417       Call(Ptr(Void), "malloc", [ Call(Int, "sizeof", [Id(Void, "
418         int" ] ) ] )
419     ));
420     Expr(Assign(Deref(Int, Id( Ptr(Int), result_var)),
421       bool_val
422     )
423   ])
424 | Sast.StrLiteral(l, dt) ->
425   let result_var = "v" ^ string_of_int(create_auto env "" (dt)) in
426   Block([
427     (*create string*)
428     Vdecl(Ptr(Cstring), result_var); (* char **result *)
429     Expr(Assign(Id(Ptr(Cstring), result_var),
430       Call(Ptr(Void), "malloc", [ Call(Int, "sizeof", [Id(Void, "
431         char *" ] ) ] )
432     )
433   ); (* *result = malloc(strlen(literal)) *)
434   Expr(Assign(Deref(Cstring, Id(Ptr(Cstring), result_var)),
435     Call(Ptr(Void), "malloc",
436       [ Binop(Int, Call(Int, "strlen", [Literal(Cstring,
437         l)]), Add, Literal(Int, "1")) ]
438   ))

```

```

432         )
433     )
434     ); (* strcpy( *result, literal) *)
435     Expr(Call(Ptr(Void),
436         "strcpy", [Deref(Cstring, Id(Ptr(Cstring), result_var));
437             Literal(Cstring, l)
438         ]
439     )
440 )
441 ])
442 | Sast.ListLiteral(el, dt) ->
443     let c_dt = dt_to_ct dt in
444     let elem_stype = get_list_type dt in
445     let elem_ctype = dt_to_ct elem_stype in
446     let enq_func = (match elem_stype with
447         | Num -> "num_add_back"
448         | String -> "string_add_back"
449         | Node -> "node_add_back"
450         | Graph -> "graph_add_back"
451         | _ -> raise (Failure("can not enqueue this datatype"))
452     ) in
453     let temp_list = "v" ^ string_of_int(create_auto env "" (dt)) in (* the
454         variable containing the list *)
455
456     let rec build_enqueue ops = function
457     | [] -> ops
458     | hd :: tl ->
459         let elem_c = translate_expr env hd in (* translate list element being
460             added *)
461         let elem_result = "v" ^ string_of_int (find_max_index !(List.hd env.
462             var_inds)) in (* get result of element translation *)
463         let enq_call = Call(c_dt, enq_func, [Deref(c_dt, Id(Ptr(c_dt),
464             temp_list));
465             Id(Ptr(elem_ctype), elem_result) (*
466                 element translation result *)
467         ]
468         ) in
469         build_enqueue ((elem_c, enq_call) :: ops) tl
470     in
471     let enq_calls = build_enqueue [] el in (* get calls for each element in
472         the list *)
473
474     let rec build_list stmts = function
475     | [] -> stmts
476     | hd :: tl ->
477         let en_stmt = Block([(fst hd);
478             Expr(Assign(Deref(c_dt, Id(Ptr(c_dt), temp_list)),
479                 (snd hd)
480             )
481         )
482         ] in
483         build_list (en_stmt :: stmts) tl
484     in
485     let c_stmts = build_list [] enq_calls in (* combines the element

```

```

translation and enqueue calls *)
480
481 let result_var = "v" ^ string_of_int(create_auto env "" (dt)) in (* will
equal the temporary list created earlier *)
482
483 Block([Vdecl(Ptr(c_dt), temp_list);
484 Expr(Assign(Id(Ptr(c_dt), temp_list),
485 Call(Ptr(Void), "malloc", [ Call(Int, "sizeof", [Id(Void,
"list_t *")])]) ]))
486 ));
487 Expr(Assign(Deref(c_dt, Id(Ptr(c_dt), temp_list)), Id(Void, "NULL"
)));
488 Vdecl(Ptr(c_dt), result_var)
489 ]
490 @
491 (List.rev (Expr(Assign(Id(Ptr(c_dt), result_var), Id(Ptr(c_dt),
temp_list))))
492 :: (List.rev c_stmts)
493 )))) (* add the assignment to the end of enqueue calls *)
494
495 | Sast.DictLiteral(kvl, dt) ->
496 DictLiteral(dt_to_ct dt,
497 List.map (fun f -> (translate_expr env (fst f), translate_expr
env (snd f))) kvl) (* TODO *)
498
499 | Sast.Id(v, dt) ->
500 let result_var = "v" ^ string_of_int(create_auto env "" (dt)) in
501 let index = "v" ^ string_of_int(find_var v env.var_inde) in (* see if
id exists, get the num index of the var *)
502 let v_type = dt_to_ct dt in
503 Block([
504 Vdecl(Ptr(v_type), result_var);
505 Expr(Assign(Id(Ptr(v_type), result_var), Ref(Ptr(v_type), Id(
v_type, index) )))
506 ])
507
508 | Sast.Binop(e1, op, e2, dt) ->
509 let c_dt = dt_to_ct dt in
510
511 let ce1 = translate_expr env e1 in
512 let result_e1 = "v" ^ string_of_int (find_max_index !(List.hd env.
var_inde)) in (* get result var of e1's translation *)
513 let e1_cdt = dt_to_ct (get_sexpr_type e1) in (*gets is the c data type
of the expression*)
514
515 let ce2 = translate_expr env e2 in
516 let result_e2 = "v" ^ string_of_int (find_max_index !(List.hd env.
var_inde)) in (* get result var of e1's translation *)
517 let e2_cdt = dt_to_ct (get_sexpr_type e2) in
518
519 let cdt1 = Translate.get_cexpr_type ce1 in
520 let cdt2 = Translate.get_cexpr_type ce2 in
521
522 let result_var = "v" ^ string_of_int(create_auto env "" (dt)) in (*
create a new auto_var to store THIS EXPR'S result *)
523 let result_decl = Vdecl(Ptr(c_dt), result_var) in (* declare this expr's

```

```

    result var *)
522
523 let binop_func =
524 (match op with
525 | Add ->
526 (match e1_cdt with
527 | Float ->
528 (match e2_cdt with
529 | Float -> Translate.Binop(Float,
530 Deref(e1_cdt, Id(Ptr(e1_cdt), result_e1)
531 ),
532 op, Deref(e2_cdt, Id(Ptr(e2_cdt), result_e2
533 )))
534 | Cstring ->
535 let float_convert = string_of_stmt ce1 in
536 Block(
537 [(snd float_convert) ;
538 translate_expr env (Sast.Binop(Id((fst float_convert),
539 String), Add, e2, String))]
540 | _ -> raise(Failure("With the type checking in Sast, this
541 should never be reached..."))
542 )
543 | Cstring ->
544 (match e2_cdt with
545 | Float ->
546 let float_convert = string_of_stmt ce2 in
547 Block([(snd float_convert) ;
548 translate_expr env (Sast.Binop(Id((fst float_convert),
549 String), Add, e1, String))]
550 | Cstring ->
551 let c_string = string_concat ce1 ce2 in
552 Block([(snd c_string)])
553 | Int ->
554 let int_convert = string_of_stmt ce2 in
555 Block([(snd int_convert) ;
556 translate_expr env (Sast.Binop(Id((fst int_convert),
557 String), Add, e1, String))]
558 | _ -> raise(Failure("With the type checking in Sast, this
559 should never be reached..."))
560 )
561 | Graph ->
562 (match e2_cdt with
563 | Node ->
564 Call(Graph, "graph_plus_node", [Deref(Graph, Id(e1_cdt,
565 result_e1));
566 Deref(Node, Id(e2_cdt, result_e2))
567 ])
568 | Graph ->
569 Nostmt
570 | _ -> raise(Failure("With the type checking in Sast, this
571 should never be reached..."))
572 )
573 | Node ->
574 (match e2_cdt with

```

```

565     | Node ->
566         Call(Graph, "node_plus_node", [Deref(Node, Id(e1_cdt,
567         result_e1));
568         Deref(Node, Id(e2_cdt, result_e2))])
569
570     | Graph -> Translate.Binop(cdt1, ce1, op, ce2) (*TODO*)
571     | _ -> raise(Failure("With the type checking in Sast, this
572         should never be reached..."))
573
574 )
575 | List(dt) -> Translate.Binop(cdt1, ce1, op, ce2) (*TODO*)
576 | Int ->
577     (match e2_cdt with
578     | Cstring ->
579         let int_convert = string_of_stmt ce1 in
580         Block(
581             [(snd int_convert) ;
582             translate_expr env (Sast.Binop(Id((fst int_convert),
583             String), Add, e2, String))]
584     | Int -> Translate.Binop(Int,
585         Deref(e1_cdt, Id(Ptr(e1_cdt), result_e1)
586         ),
587         op, Deref(e2_cdt, Id(Ptr(e2_cdt), result_e2
588         )))
589     | _ -> raise (Failure("invalid operation"))
590     )
591 | _ -> raise (Failure("Invalid c type for + binop " ^ (Translate
592     .type_to_str cdt2)))
593 )
594 | Sub ->
595     (match e1_cdt with
596     | Float -> Translate.Binop(Float,
597         Deref(e1_cdt, Id(Ptr(e1_cdt), result_e1)
598         ),
599         op, Deref(e2_cdt, Id(Ptr(e2_cdt), result_e2
600         )))
601     | Graph ->
602         (match cdt2 with
603         | Node -> Translate.Binop(cdt1, ce1, op, ce2) (* TODO *)
604         | Graph -> Translate.Binop(cdt1, ce1, op, ce2) (* TODO *)
605         | _ -> raise(Failure("With the type checking in Sast, this
606             should never be reached..."))
607         )
608     | _ -> raise(Failure("With the type checking in Sast, this
609         should never be reached..."))
610     )
611 | Mult | Div -> Translate.Binop(Float, ce1, op, ce2)
612 | Equal | Neq ->
613     (* This one isn't complete, dict maps to what c type? confusion *)
614     (match e1_cdt with
615     | Float -> Call(Int, "float_equals", [Id(e1_cdt, result_e1);
616         Id(e2_cdt, result_e2)])
617     | Int -> Translate.Binop(Int, Deref(e1_cdt, Id(Ptr(e1_cdt),
618         result_e1)),
619         op, Deref(e2_cdt, Id(Ptr(e2_cdt), result_e2

```

```

609         )))
610     | Cstring ->
611         (* (strcmp(check,input) = 0) *)
612         let auto_var = "v" ^ string_of_int(create_auto env "" (Sast.
        Num)) in
613         Assign(Id(Int, auto_var), (Call(Int, "strcmp", [ce1;ce2])))
614         ;
615     | Graph ->
616         (match e2_cdt with
617         | Graph ->
618             (match op with
619             | Equal -> Call(Int, "graph_equals", [Deref(Ptr(Graph)
620                 , Id(e1_cdt, result_e1)); Deref(Ptr(Graph), Id(
621                 e2_cdt, result_e2))])
622             | Neq -> Translate.Binop(Int, Call(Int, "graph_equals"
623                 ,
624                 [Deref(Ptr(Graph), Id(e1_cdt
625                 , result_e1));
626                 Deref(Ptr(Graph), Id(e2_cdt
627                 , result_e2))
628                 ]),
629                 op, Literal(Int, "1"))
630             | _ -> raise(Failure("With the type checking in Sast,
631                 this should never be reached..."))
632         )
633         | _ -> raise(Failure("With the type checking in Sast, this
634                 should never be reached..."))
635     )
636     | Node ->
637         (match e2_cdt with
638         | Node -> Translate.Binop(cdt1, ce1, op, ce2)
639         | _ -> raise(Failure("With the type checking in Sast, this
640                 should never be reached..."))
641         )
642     | List(dt) -> Translate.Binop(cdt1, ce1, op, ce2) (*TODO*)
643     | Void -> Translate.Binop(cdt1, ce1, op, ce2) (*TODO*)
644     | _ -> raise (Failure("Invalid c type for ==/!= binop"))
645 )
646 | Less | Leq | Greater | Geq ->
647     (match e1_cdt with
648     | Float -> Translate.Binop(Float,
649         Deref(e1_cdt, Id(Ptr(e1_cdt), result_e1)
650         ),
651         op, Deref(e2_cdt, Id(Ptr(e2_cdt), result_e2
652         )))
653     | Int ->
654         Translate.Binop(Int,
655         Deref(e1_cdt, Id(Ptr(e1_cdt), result_e1)
656         ),
657         op, Deref(e2_cdt, Id(Ptr(e2_cdt), result_e2
658         )))
659     | Long -> Translate.Binop(Long,
660         Deref(e1_cdt, Id(Ptr(e1_cdt), result_e1)

```



```

648         ),
        op, Deref(e2_cdt, Id(Ptr(e2_cdt), result_e2
        )))
649     | Cstring ->
650         let auto_var = "v" ^ string_of_int(create_auto env "" (Sast.
        Num)) in
651         Block([Vdecl(Int, auto_var);
652             Assign(Id(Int, auto_var), (Call(Int, "strcmp", [ce1;
        ce2])))
653         ]);
654     | _ -> raise(Failure("With the type checking in Sast, this
        should never be reached..."))
655 )
656 | LogAnd | LogOr -> Translate.Binop(Int, ce1, op, ce2)
657 ) in
658 Block([
659     ce1;
660     ce2;
661     result_decl;
662     Expr(Assign(Id(Ptr(c_dt), result_var),
663         Call(Ptr(Void), "malloc", [ Call(Int, "sizeof",
664             [Id(Void, Translate.type_to_str c_dt)] ) ] )
665     ));
666     Expr(Assign(Deref(c_dt, Id(Ptr(c_dt), result_var)), Assoc(
        binop_func)
667     )) (* store the result of Access in our result_var *)
668 ])
669 | Sast.Call(func_name, e1, dt) ->
670     let return_type = dt_to_ct dt in
671     (match func_name with
672     | "print" ->
673         let rec print_builder elems = function
674         | [] -> List.rev elems
675         | hd :: tl ->
676             let hd_type = get_sexpr_type hd in
677             let print_expr = translate_expr env hd in (* elem to print *)
678             let print_result = "v" ^ string_of_int (find_max_index !(List.
        hd env.var_inds)) in (* result of elem translation *)
679             let deref_print_var = Deref(Node, Id(Ptr(Node), print_result))
        in
680             let print_type = dt_to_ct hd_type in (* type of elem *)
681
682             (match hd_type with
683             | Num | String | Bool ->
684                 print_builder (Block([print_expr;
685                     Expr(Call(Void, "printf", [Literal(Cstring
        , get_fmt_str print_type);
686                         Deref(print_type, Id(
        Ptr(print_type),
        print_result))
687                     ]))
688                 ]) :: elems)
689             | Node ->

```

```

691     print_builder (Block([ print_expr;
692         Expr(Call(Void, "printf", [Literal(
        Cstring, "%s"); Literal(Cstring, "N-"
        )] ));
693     Expr(Call(Void, "printf", [Literal(
        Cstring, "%d"); Cast(Int,
        deref_print_var) ] ));
694     Expr(Call(Void, "printf", [Literal(
        Cstring, "%s"); Literal(Cstring, "
        (\\\\"))]);
695     Expr(Call(Void, "printf", [Literal(
        Cstring, "%s"); Cast(Cstring, Member(
        Ptr(Void), deref_print_var, "data")]
696     ));
697     Expr(Call(Void, "printf", [Literal(
        Cstring, "\\\"))]);
698     ]) :: elems)
699         tl
700 | List(dt) ->
701     let print_loop = translate_stmt env
702         (Sast.For("elem", hd,
703             [Expr(Call("print", [Id("elem", dt)
704                 ], Sast.Void));
705             Expr(Call("print", [StrLiteral(", "
706                 , String)], Sast.Void)
707             ])) in
708     print_builder
709     (
710         (* b/c of building the list up backwards,
711            this list must be declared in reverse order
712
713            // c translation:
714            // print(num_list)
715            list_t* auto;
716            for (auto = num_list; auto; auto = auto->next) {
717                print( *auto );
718            }
719            *)
720     [
721         Expr(Call(Void, "printf", [Literal(Cstring, "]" )]);
722         print_loop;
723         Expr(Call(Void, "printf", [Literal(Cstring, "[" )])
724
725
726     ]
727     @ elems
728     )
729     tl
730
731 | Dict(dtk, dtv) ->
732
733     (* build the print value statement for the specific key

```

```

    type *)
734 (*
735 // C code:
736 int i;
737 entry_t *temp;
738 void *key;
739 /* print "{"; */
740 int first = 1;
741 for(i = 0; i < TABLE_SIZE; i = i + 1) {
742     for(temp = d[i]; temp; temp = temp->next) {
743         key = temp->key;
744         if(first) {
745             first = 0;
746             /* print key, ": ", value */
747         } else {
748             /* print ", " , key, ": ", value */
749         }
750     }
751 }
752 *)
753 let print_loop = translate_stmt env
754     (Sast.For("$key", hd,
755         [Expr(Call("print",
756             [Id("$key", dtk)],
757             Sast.Void));
758         Expr(Call("print",
759             [StrLiteral(": ", String)],
760             Sast.Void));
761         Expr(Call("print",
762             [Sast.Access(hd, Id("$key",
763                 dtk), dtv)],
764             Sast.Void));
765         Expr(Call("print",
766             [StrLiteral(", ", String)],
767             Sast.Void));
768     ])) in
769
770 print_builder
771 (
772     (* b/c of building the list up backwards,
773     this list must be declared in reverse order
774
775     // c translation:
776     // print(num_list)
777     list_t* auto;
778     for (auto = num_list; auto; auto = auto->next) {
779         print( *auto );
780     }
781     *)
782 [
783     Expr(Call(Void, "printf", [Literal(Cstring, "}")]);
784     print_loop;
785     Expr(Call(Void, "printf", [Literal(Cstring, "{")]))

```

```

786
787
788         ]
789         @ elems
790     )
791     tl
792     | Graph -> print_builder (Nostmt :: elems) tl (*TODO*)
793     | Void -> raise (Failure "stop trying to print Void -- it's
794         not gonna happen")
795 )
796 in
797 Block( print_builder [] el (* TODO *) )
798 | "len" ->
799 (*func_name, el, dt*)
800 let elem_c = (translate_expr env (List.hd el)) in
801 let arg_e = "v" ^ string_of_int (find_max_index !(List.hd env.
802     var_inds)) in
803 let arg_dt = get_sexpr_type (List.hd el) in
804 let arg_id = Id((dt_to_ct arg_dt), arg_e) in
805
806 let result_var = "v" ^ string_of_int(create_auto env "" (dt))
807     in (* create a new auto_var to store THIS EXPR'S result *)
808 let result_decl = Vdecl(Ptr(Float), result_var) in
809 let final_result = Id(Ptr(Float), result_var) in
810 (match arg_dt with
811 | List(dt) ->
812     Block([
813         elem_c;
814         result_decl;
815         Expr(Assign(Id(Ptr(Float), result_var),
816             Call(Ptr(Void), "malloc", [ Call(Int, "sizeof", [Id
817                 (Void, "float")] ) ] )
818         )
819     );
820     Expr(Assign( Deref(Float, final_result),
821         Cast((Float),
822             Call(Int, "list_len", [ Deref((dt_to_ct
823                 arg_dt), arg_id) ]))
824     ));
825 ])
826 | Dict(dtk, dtv) ->
827     Block([
828         elem_c;
829         result_decl;
830         Expr(Assign(Id(Ptr(Float), result_var),
831             Call(Ptr(Void), "malloc", [ Call(Int, "sizeof", [Id
832                 (Void, "float")] ) ] )
833         )
834     );
835     Expr(Assign( Deref(Float, final_result),
836         Cast((Float),
837             Call(Int, "dict_len", [ Deref((dt_to_ct

```

```

834         arg_dt), arg_id) ]))
835     ));
836
837     | _ -> raise (Failure "len not implemented for this type")
838     )
839 | "min" | "max" ->
840     let elem_c = (translate_expr env (List.hd el)) in
841     let arg_e = "v" ^ string_of_int (find_max_index !(List.hd env.
842     var_inds)) in
843     let arg_dt = get_sexpr_type (List.hd el) in
844     let arg_id = Id((dt_to_ct arg_dt), arg_e) in
845
846     (match arg_dt with
847     | List(dt) ->
848         let result_var = "v" ^ string_of_int(create_auto env "" (
849         dt)) in (* create a new auto_var to store THIS EXPR'S
850         result *)
851         let e_list_type = (get_list_type arg_dt) in
852         let result_decl = Vdecl(Ptr(Float), result_var) in
853         let final_result = Id(Ptr(Float), result_var) in
854
855         (match e_list_type with
856         | Num ->
857             let fname = "num_list_" ^ func_name in
858             Block([
859                 elem_c;
860                 result_decl;
861                 Expr(Assign(Id(Ptr(Float), result_var),
862                 Call(Ptr(Void), "malloc", [ Call(Int, "
863                 sizeof", [Id(Void, "float")] ) ] )
864                 ));
865                 Expr(Assign( Deref(Float, final_result),
866                 Cast((Float),
867                 Call(Float, fname, [ Deref((
868                 dt_to_ct arg_dt), arg_id)
869                 ]))
870                 ));
871             ])
872         | _ -> raise (Failure ("cannot do min max ")))
873     | Dict(dtk, dtv) ->
874         (match dtv with
875         | Num ->
876             let fname = "num_dict_" ^ func_name in
877             let d_k = dt_to_ct dtk in
878             let result_var = "v" ^ string_of_int(create_auto
879             env "" (dtk)) in (* create a new auto_var
880             to store THIS EXPR'S result *)
881             let result_decl = Vdecl(Ptr(d_k), result_var) in
882             let final_result = Id(Ptr(d_k), result_var) in
883
884             Block([

```

```

879         elem_c;
880         result_decl;
881         Expr(Assign(Id(Ptr(d_k), result_var),
882             Call(Ptr(Void), "malloc", [ Call(Int, "
sizeof", [Id(Void, Translate.
type_to_str(d_k))] ) ]
883         )
884     );
885
886     Expr(Assign(Deref(d_k, final_result),
887         Cast((d_k),
888             Call(Ptr(Void), fname, [ Deref((
dt_to_ct arg_dt), arg_id) ]))
889     ));
890     ])
891     | _ -> raise (Failure ("cannot do min max "))
892     | _ -> raise (Failure("can not enqueue this datatype"))
893 | _ ->
894     let c_args = List.rev (build_args [] el) in
895     let c_stmts = List.map (fun t -> (fst t)) c_args in
896     let result_params = List.map (fun t -> (snd t)) c_args in
897     let func_index = "f" ^ string_of_int(find_var func_name env.
func_inds) in
898     let call_result = "v" ^ string_of_int(create_auto env "" (dt)) in
899     Block(c_stmts
900         @
901         [
902             Vdecl(Ptr(return_type), call_result);
903             Expr(Assign(Id(Ptr(return_type), call_result),
904                 Call(Ptr(Void), "malloc", [ Call(Int, "sizeof", [Id
(Void, Translate.type_to_str return_type)] ) ]
905             )
906         );
907             Expr(Assign(Deref(return_type, Id(Ptr(return_type),
call_result)),
908                 Call(dt_to_ct dt, func_index, result_params)
909             )
910         )
911     ]
912     )
913 )
914
915 | Sast.Access(e1, e2, dt) ->
916     let c_dt = dt_to_ct dt in
917     let e1_dt = get_sexpr_type e1 in
918     let e2_dt = get_sexpr_type e2 in
919
920     let c_e1 = translate_expr env e1 in (* translate e1 to c *)
921     let result_e1 = "v" ^ string_of_int (find_max_index !(List.hd env.
var_inds)) in (* get result var of e1's translation *)
922     let e1_deref = Deref(dt_to_ct e1_dt, Id(Ptr(dt_to_ct e1_dt), result_e1
)) in
923
924     let c_e2 = translate_expr env e2 in (* translate e2 to c *)

```

```

925     let result_e2 = "v" ^ string_of_int (find_max_index !(List.hd env.
926         var_inds)) in (* get result var of e2's translation *)
927
928     let e2_deref = Deref(dt_to_ct e2_dt, Id(Ptr(dt_to_ct e2_dt), result_e2
929         )) in
930
931     let result_var = "v" ^ string_of_int(create_auto env "" (dt)) in (*
932         create a new auto_var to store THIS EXPR'S result *)
933     let result_decl = Vdecl(Ptr(c_dt), result_var) in (* declare this expr's
934         result var *)
935     let args = [e1_deref; e2_deref] in (* specific to access function calls
936         *)
937     let access_call = (match e1_dt with
938         | List(dt) ->
939             Call(Ptr(Void), "list_access", args)
940         | Dict(dtk, dtv) ->
941             let c_dtk = dt_to_ct dtk in
942             (match c_dtk with
943                 | Float -> Call(Ptr(Void), "get_num", args)
944                 | Cstring -> Call(Ptr(Void), "get_string", args)
945                 | Graph -> Call(Ptr(Void), "get_graph", args)
946                 | Node -> Call(Ptr(Void), "get_node", args)
947                 | _ -> raise(Failure("unsupported dict type"))
948             )
949         | _ -> raise(Failure("unsupported access"))
950     ) in (* evaluate an Access expression *)
951     Block([
952         c_e1;
953         c_e2;
954         result_decl;
955         Assign(Id(Ptr(c_dt), result_var), Expr(Cast(Ptr(c_dt),
956             access_call))) (* store the result of Access in our result_var
957             *)
958     ])
959 | Sast.MemberCall(e, f, e1, dt) ->
960     let c_dt = dt_to_ct dt in
961
962     let c_e = translate_expr env e in (* translate e1 to c *)
963     let result_e = "v" ^ string_of_int (find_max_index !(List.hd env.
964         var_inds)) in (* get result var of e1's translation *)
965     let e_dt = get_sexpr_type e in
966     let c_id = Id(dt_to_ct e_dt, result_e) in
967
968     (match f with
969         | "enqueue" | "push" ->
970             let e_list_type = (get_list_type e_dt) in
971             let suffix = (if f = "enqueue" then "back" else "front") in
972             let func_name =
973                 (match e_list_type with
974                     | Num -> "num_add_" ^ suffix
975                     | String -> "string_add_" ^ suffix
976                     | Node -> "node_add_" ^ suffix
977                     | Graph -> "graph_add_" ^ suffix
978                     | _ -> raise (Failure("can not enqueue this datatype"))) in

```

```

971 let elem_c = (translate_expr env (List.hd e1)) in
972 let arg_e = "v" ^ string_of_int (find_max_index !(List.hd env
    .var_inds)) in
973 let arg_dt = get_cexpr_type elem_c in
974 let arg_id = Id(arg_dt, arg_e) in
975
976 let result_var = "v" ^ string_of_int(create_auto env "" (dt))
    in (* create a new auto_var to store THIS EXPR'S result
    *)
977 let result_decl = Vdecl(Ptr(dt_to_ct e_dt), result_var) in (*
    declare this expr's result var *)
978 (*let e1_cdt = dt_to_ct (get_sexpr_type e1) in *)
979 let final_result = Id(dt_to_ct e_dt, result_var) in
980
981 Block([
982   c_e;
983   elem_c;
984   result_decl;
985   Expr(Assign(final_result,
986             Call(Ptr(Void), "malloc", [ Call(Int, "sizeof", [Id
    (Void, Translate.type_to_str c_dt)] ) ]
987         )
988   );
989   Expr(Assign(Deref((dt_to_ct e_dt), c_id),
990             Call(c_dt, func_name,
991                 [Deref((dt_to_ct e_dt), c_id);
992                  Deref((arg_dt), arg_id)]
993             ))
994   (* store the result of Access in our result_var *)
995   ])
996 | "dequeue" | "pop" ->
997 let result_var = "v" ^ string_of_int(create_auto env "" (dt))
    in (* create a new auto_var to store THIS EXPR'S result
    *)
998 let result_decl = Vdecl(Ptr(c_dt), result_var) in (* declare
    this expr's result var *)
999
1000 Block([
1001
1002   c_e;
1003   result_decl;
1004   Expr(Assign(Deref((dt_to_ct e_dt), c_id),
1005             Call(c_dt, "pop",
1006                 [Deref((dt_to_ct e_dt), c_id)]
1007             ))
1008   (* store the result of Access in our result_var *)
1009   ])
1010 | "peek" ->
1011 let result_var = "v" ^ string_of_int(create_auto env "" (dt))
    in (* create a new auto_var to store THIS EXPR'S result *)
1012 let result_decl = Vdecl(Ptr(c_dt), result_var) in (* declare
    this expr's result var *)
1013 let final_result = Id(dt_to_ct e_dt, result_var) in
1014

```



```

1015     Block([
1016
1017         c_e;
1018         result_decl;
1019         Expr(Assign(final_result,
1020                   Call(c_dt, "peek",
1021                       [Deref((dt_to_ct e_dt), c_id)]))
1022         ))
1023         (* store the result of Access in our result_var *)
1024     ])
1025
1026 | "oute" | "ine" ->
1027     let func_name =
1028         (match f with
1029         | "oute" -> "out"
1030         | "ine" -> "in"
1031         | _ -> raise (Failure ("unexpected out/in func name")))
1032         ) in
1033     let result_var = "v" ^ string_of_int(create_auto env "" (dt))
1034         in (* create a new auto_var to store THIS EXPR'S result
1035            *)
1036     let result_decl = Vdecl(Ptr(Ptr(Ptr(Entry))), result_var) in
1037         (* declare this expr's result var *)
1038     let final_result = Id(Ptr(Entry), result_var) in
1039     Block([
1040         c_e;
1041         result_decl;
1042
1043         Expr(Assign(
1044             final_result,
1045             Ref(Ptr(Ptr(Ptr(Entry))), Member(Ptr(Entry), Deref((
1046                 dt_to_ct e_dt), c_id), func_name))
1047         ))
1048     ])
1049 | "remove" ->
1050     (match e_dt with
1051     | Dict(dtk, dtv) ->
1052         let func_name =
1053             (match dtk with
1054             | Num -> "num_dict_remove"
1055             | String -> "string_dict_remove"
1056             | Node -> "node_dict_remove"
1057             | Graph -> "graph_dict_remove"
1058             | _ -> raise (Failure("can not enqueue this datatype")))
1059             in
1060
1061         let key_c = (translate_expr env (List.hd el)) in
1062         let key_result = "v" ^ string_of_int (find_max_index !(
1063             List.hd env.var_inds)) in
1064         let key_dt = get_cexpr_type key_c in
1065         let arg_id = Id(key_dt, key_result) in
1066
1067         let result_var = "v" ^ string_of_int(create_auto env "" (
1068             dt)) in (* create a new auto_var to store THIS EXPR'S

```

```

1062         result *)
1063         let result_decl = Vdecl(Ptr(Void), result_var) in (*
1064             declare this expr's result var *)
1065         Block([
1066             c_e;
1067             key_c;
1068             result_decl;
1069             Expr(Call(Void, func_name,
1070                 [Deref((dt_to_ct e_dt), c_id);
1071                 Deref(key_dt, arg_id)]))
1072             );
1073             (* store the result of Access in our result_var *)
1074         ])
1075     | _ -> raise (Failure ("not a dict"))
1076 )
1077
1078 | "val" ->
1079     let result_var = "v" ^ string_of_int(create_auto env "" (dt))
1080     in (* create a new auto_var to store THIS EXPR'S result
1081         *)
1082     Block([
1083         c_e;
1084         Vdecl(Ptr(Cstring), result_var);
1085         Expr(Assign(
1086             Id(Cstring, result_var),
1087             Call(Ptr(Void),
1088                 "malloc",
1089                 [Call(Int, "sizeof", [Id(Void, "char *")])
1090                 ])
1091             ));
1092         Expr(Assign(
1093             Deref(Cstring, Id(Ptr(Cstring), result_var)),
1094             Member(Cstring, Deref((dt_to_ct e_dt), c_id), "data"
1095             )))
1096     ])
1097     | _ -> raise (Failure("not enqueue"))
1098 | Sast.Undir(v1, v2, dt) ->
1099     let v1_index = "v" ^ string_of_int (find_var v1 env.var_inds) in
1100     let v2_index = "v" ^ string_of_int (find_var v2 env.var_inds) in
1101     Call(Void, "connect_undir", [Id(Ptr(Node), v1_index); Id(Ptr(Node),
1102         v2_index)])
1103 | Sast.Dir(v1, v2, dt) ->
1104     let v1_index = "v" ^ string_of_int (find_var v1 env.var_inds) in
1105     let v2_index = "v" ^ string_of_int (find_var v2 env.var_inds) in
1106     Call(Void, "connect_dir", [Id(Ptr(Node), v1_index); Id(Ptr(Node),
1107         v2_index)])
1108 | Sast.UndirVal(v1, v2, w, dt) ->
1109     let v1_index = "v" ^ string_of_int (find_var v1 env.var_inds) in
1110     let v2_index = "v" ^ string_of_int (find_var v2 env.var_inds) in
1111     let w_c = translate_expr env w in
1112     let w_result = "v" ^ string_of_int (find_max_index !(List.hd env.
1113         var_inds)) in

```

```

1108     let w_deref = Deref(dt_to_ct (get_sexpr_type w),
1109                       Id(Ptr(dt_to_ct (get_sexpr_type w)),
1110                          w_result)) in
1111
1112     Block([
1113         w_c;
1114         Call(Void, "connect_undir_weighted ", [Id(Ptr(Node), v1_index); Id
1115           (Ptr(Node),
1116              v2_index); w_deref])
1117     ])
1118 | Sast.DirVal(v1, v2, w, dt) ->
1119     let v1_index = "v" ^ string_of_int (find_var v1 env.var_inde) in
1120     let v2_index = "v" ^ string_of_int (find_var v2 env.var_inde) in
1121     let w_c = translate_expr env w in
1122     let w_result = "v" ^ string_of_int (find_max_index !(List.hd env.
1123       var_inde)) in
1124     let w_deref = Deref(dt_to_ct (get_sexpr_type w), Id(Ptr(dt_to_ct
1125       (get_sexpr_type w)), w_result)) in
1126
1127     Block([
1128         w_c;
1129         Call(Void, "connect_dir_weighted ", [Id(Ptr(Node), v1_index); Id(
1130           Ptr(Node),
1131              v2_index); w_deref])
1132     ])
1133 | Sast.BidirVal(w1, v1, v2, w2, dt) -> Nostmt (* TODO *)
1134 | Sast.NoOp(s, dt) -> Nostmt (* TODO *)
1135 | Sast.Noexpr -> (Nostmt)
1136 and
1137 translate_stmt env = function
1138 | Sast.Block(sl) ->
1139     let csl = List.map (translate_stmt env) sl in
1140     Block(csl)
1141 | Sast.Expr(e) -> Expr(translate_expr env e)
1142 | Sast.Vdecl(dt, id) ->
1143     (List.hd env.var_types) := StringMap.add id dt !(List.hd env.
1144       var_types); (* add type map *)
1145     (List.hd env.var_inde) := StringMap.add id (find_max_index !(List.hd
1146       env.var_inde)+1) !(List.hd env.var_inde); (* add index map *)
1147     let index = "v" ^ string_of_int(find_var id env.var_inde) in
1148     (match dt with
1149     | Num -> Vdecl(Float, index)
1150     | String -> Vdecl(Cstring, index)
1151     | Bool -> Vdecl(Int, index)
1152     | Graph -> Block([Vdecl(Graph, index);
1153       ]) (* C: graph_t *g1 = init_graph(); *)
1154     | Node -> Block([Vdecl((Node), index);]) (* C: node_t *x = init_node
1155       (""); *)
1156     | List(dt) -> Vdecl(List(dt_to_ct dt), index) (* C: list_t *x; *)
1157     | Dict(dtk, dtv) -> Vdecl(Ptr(Ptr(Entry)), index) (* TODO *)
1158     | Void -> raise (Failure ("should not be using Void as a datatype"))
1159     )
1160 )
1161 | Sast.Assign(v, e, dt) ->

```

```

1155     let ce = translate_expr env e in
1156     let e_result = "v" ^ string_of_int (find_max_index !(List.hd env.
      var_inds)) in
1157     let cv = translate_expr env v in
1158     let v_result = "v" ^ string_of_int (find_max_index !(List.hd env.
      var_inds)) in
1159     let v_type = get_sexpr_type v in
1160     let var_type = dt_to_ct v_type in
1161
1162     Block([
1163         ce; (* translation of assignment *)
1164         cv; (* translation of assignee *)
1165         Expr(Assign(Deref(var_type, Id(Ptr(var_type), v_result)), Deref
      (var_type, Id(Ptr(var_type), e_result))))
1166     ])
1167 | Sast.AccessAssign(e1, e2, e3, dt) -> (* check access first *)
1168     let e1_dt = get_sexpr_type e1 in
1169     let e2_dt = get_sexpr_type e2 in
1170     let e3_dt = get_sexpr_type e3 in
1171     let c_e1 = translate_expr env e1 in
1172     let result_e1 = "v" ^ string_of_int (find_max_index !(List.hd env.
      var_inds)) in (* get result var of e1's translation *)
1173     let e1_deref = Deref(dt_to_ct e1_dt, Id(Ptr(dt_to_ct e1_dt), result_e1
      )) in
1174     let c_e2 = translate_expr env e2 in
1175     let result_e2 = "v" ^ string_of_int (find_max_index !(List.hd env.
      var_inds)) in (* get result var of e1's translation *)
1176     let e2_deref = Deref(dt_to_ct e2_dt, Id(Ptr(dt_to_ct e2_dt), result_e2
      )) in
1177     let c_e3 = translate_expr env e3 in
1178     let result_e3 = "v" ^ string_of_int (find_max_index !(List.hd env.
      var_inds)) in (* get result var of e1's translation *)
1179     let e3_deref = Deref(dt_to_ct e3_dt, Id(Ptr(dt_to_ct e3_dt), result_e3
      )) in
1180     let args = [e1_deref; e2_deref; Id(dt_to_ct e3_dt, result_e3)] in
1181     let call = (match e1_dt with
1182     | List(dt) ->
1183         if (e2_dt = Sast.Num) then
1184             if (e3_dt = dt) then
1185                 let c_dt = dt_to_ct dt in
1186                 (match c_dt with
1187                 | Float -> Expr(Call(Ptr(Void), "num_index_insert",
      args))
1188                 | Cstring -> Expr(Call(Ptr(Void), "string_index_insert
      ", args))
1189                 | Graph -> Expr(Call(Ptr(Void), "graph_index_insert",
      args))
1190                 | Node -> Expr(Call(Ptr(Void), "node_index_insert",
      args))
1191                 | _ -> raise(Failure("unsupported list type"))
1192                 )
1193             else
1194                 raise(Failure("accessassign: expr right of = is not
      same type as list"))

```

```

1195     else
1196         raise(Failure("AccessAssign: assign expr on left is wrong
                        for list"))
1197     | Dict(dtk, dtv) ->
1198     if (e2_dt = dtk) then
1199         if (e3_dt = dtv) then
1200             let c_dtk = dt_to_ct dtk in
1201             let c_dtv = dt_to_ct dtv in
1202             (* try to get rid of problem with & *)
1203             let auto_var = "v" ^ string_of_int(create_auto env ""
                                                dtv) in
1204             (match c_dtk with
1205             | Float -> Block([Vdecl(c_dtv, auto_var);
1206                             Expr(Assign(Id(c_dtv, auto_var), e3_deref));
1207                             Expr(Assign(e1_deref,
1208                                         Call(Ptr(Void), "put_num",
1209                                               [e1_deref;
1210                                               e2_deref;
1211                                               Cast(Ptr(Void), Ref(c_dtv, Id(
1212                                                         c_dtv, auto_var)))
1213                                               ]))
1214                             )
1215             ])
1216             | CString -> Block([Vdecl(c_dtv, auto_var);
1217                                 Expr(Assign(Id(c_dtv, auto_var),
1218                                             e3_deref));
1219                                 Expr(Assign(e1_deref, Call(Ptr(Void), "
1220                                                         put_string",
1221                                                         [e1_deref;
1222                                                         e2_deref;
1223                                                         Cast(Ptr(Void), Ref(
1224                                                                 c_dtv, Id(c_dtv,
1225                                                                 auto_var)))
1226                                                         ]))
1227                                 )
1228             ])
1229             | Node -> let auto_var2 = "v" ^ string_of_int(
1230                       create_auto env "" dtk) in
1231                       Block([Vdecl(c_dtk, auto_var2);
1232                               Expr(Assign(Id(c_dtk, auto_var2),
1233                                           Deref(dt_to_ct e2_dt,
1234                                                  Id(Ptr(dt_to_ct e2_dt),
1235                                                       result_e2))));
1236                               Vdecl(c_dtv, auto_var);
1237                               Expr(Assign(Id(c_dtv, auto_var), e3_deref));
1238                               Expr(Assign(e1_deref,
1239                                           Call(Ptr(Void), "put_node",
1240                                                 [e1_deref;
1241                                                 Cast(Node, Id(c_dtk, auto_var2));
1242                                                 Cast(Ptr(Void), Ref(c_dtv, Id(
1243                                                         c_dtv, auto_var)))
1244                                                 ]))
1245                               )
1246                       ])
1247             )
1248             )
1249             )
1250             )
1251             )
1252             )
1253             )
1254             )
1255             )
1256             )
1257             )
1258             )
1259             )
1260             )
1261             )
1262             )
1263             )
1264             )
1265             )
1266             )
1267             )
1268             )
1269             )
1270             )
1271             )
1272             )
1273             )
1274             )
1275             )
1276             )
1277             )
1278             )
1279             )
1280             )
1281             )
1282             )
1283             )
1284             )
1285             )
1286             )
1287             )
1288             )
1289             )
1290             )
1291             )
1292             )
1293             )
1294             )
1295             )
1296             )
1297             )
1298             )
1299             )
1300             )
1301             )
1302             )
1303             )
1304             )
1305             )
1306             )
1307             )
1308             )
1309             )
1310             )
1311             )
1312             )
1313             )
1314             )
1315             )
1316             )
1317             )
1318             )
1319             )
1320             )
1321             )
1322             )
1323             )
1324             )
1325             )
1326             )
1327             )
1328             )
1329             )
1330             )
1331             )
1332             )
1333             )
1334             )
1335             )
1336             )
1337             )
1338             )
1339             )
1340             )
1341             )
1342             )
1343             )
1344             )
1345             )
1346             )
1347             )
1348             )
1349             )
1350             )
1351             )
1352             )
1353             )
1354             )
1355             )
1356             )
1357             )
1358             )
1359             )
1360             )
1361             )
1362             )
1363             )
1364             )
1365             )
1366             )
1367             )
1368             )
1369             )
1370             )
1371             )
1372             )
1373             )
1374             )
1375             )
1376             )
1377             )
1378             )
1379             )
1380             )
1381             )
1382             )
1383             )
1384             )
1385             )
1386             )
1387             )
1388             )
1389             )
1390             )
1391             )
1392             )
1393             )
1394             )
1395             )
1396             )
1397             )
1398             )
1399             )
1400             )
1401             )
1402             )
1403             )
1404             )
1405             )
1406             )
1407             )
1408             )
1409             )
1410             )
1411             )
1412             )
1413             )
1414             )
1415             )
1416             )
1417             )
1418             )
1419             )
1420             )
1421             )
1422             )
1423             )
1424             )
1425             )
1426             )
1427             )
1428             )
1429             )
1430             )
1431             )
1432             )
1433             )
1434             )
1435             )
1436             )
1437             )
1438             )
1439             )
1440             )
1441             )
1442             )
1443             )
1444             )
1445             )
1446             )
1447             )
1448             )
1449             )
1450             )
1451             )
1452             )
1453             )
1454             )
1455             )
1456             )
1457             )
1458             )
1459             )
1460             )
1461             )
1462             )
1463             )
1464             )
1465             )
1466             )
1467             )
1468             )
1469             )
1470             )
1471             )
1472             )
1473             )
1474             )
1475             )
1476             )
1477             )
1478             )
1479             )
1480             )
1481             )
1482             )
1483             )
1484             )
1485             )
1486             )
1487             )
1488             )
1489             )
1490             )
1491             )
1492             )
1493             )
1494             )
1495             )
1496             )
1497             )
1498             )
1499             )
1500             )
1501             )
1502             )
1503             )
1504             )
1505             )
1506             )
1507             )
1508             )
1509             )
1510             )
1511             )
1512             )
1513             )
1514             )
1515             )
1516             )
1517             )
1518             )
1519             )
1520             )
1521             )
1522             )
1523             )
1524             )
1525             )
1526             )
1527             )
1528             )
1529             )
1530             )
1531             )
1532             )
1533             )
1534             )
1535             )
1536             )
1537             )
1538             )
1539             )
1540             )
1541             )
1542             )
1543             )
1544             )
1545             )
1546             )
1547             )
1548             )
1549             )
1550             )
1551             )
1552             )
1553             )
1554             )
1555             )
1556             )
1557             )
1558             )
1559             )
1560             )
1561             )
1562             )
1563             )
1564             )
1565             )
1566             )
1567             )
1568             )
1569             )
1570             )
1571             )
1572             )
1573             )
1574             )
1575             )
1576             )
1577             )
1578             )
1579             )
1580             )
1581             )
1582             )
1583             )
1584             )
1585             )
1586             )
1587             )
1588             )
1589             )
1590             )
1591             )
1592             )
1593             )
1594             )
1595             )
1596             )
1597             )
1598             )
1599             )
1600             )
1601             )
1602             )
1603             )
1604             )
1605             )
1606             )
1607             )
1608             )
1609             )
1610             )
1611             )
1612             )
1613             )
1614             )
1615             )
1616             )
1617             )
1618             )
1619             )
1620             )
1621             )
1622             )
1623             )
1624             )
1625             )
1626             )
1627             )
1628             )
1629             )
1630             )
1631             )
1632             )
1633             )
1634             )
1635             )
1636             )
1637             )
1638             )
1639             )
1640             )
1641             )
1642             )
1643             )
1644             )
1645             )
1646             )
1647             )
1648             )
1649             )
1650             )
1651             )
1652             )
1653             )
1654             )
1655             )
1656             )
1657             )
1658             )
1659             )
1660             )
1661             )
1662             )
1663             )
1664             )
1665             )
1666             )
1667             )
1668             )
1669             )
1670             )
1671             )
1672             )
1673             )
1674             )
1675             )
1676             )
1677             )
1678             )
1679             )
1680             )
1681             )
1682             )
1683             )
1684             )
1685             )
1686             )
1687             )
1688             )
1689             )
1690             )
1691             )
1692             )
1693             )
1694             )
1695             )
1696             )
1697             )
1698             )
1699             )
1700             )
1701             )
1702             )
1703             )
1704             )
1705             )
1706             )
1707             )
1708             )
1709             )
1710             )
1711             )
1712             )
1713             )
1714             )
1715             )
1716             )
1717             )
1718             )
1719             )
1720             )
1721             )
1722             )
1723             )
1724             )
1725             )
1726             )
1727             )
1728             )
1729             )
1730             )
1731             )
1732             )
1733             )
1734             )
1735             )
1736             )
1737             )
1738             )
1739             )
1740             )
1741             )
1742             )
1743             )
1744             )
1745             )
1746             )
1747             )
1748             )
1749             )
1750             )
1751             )
1752             )
1753             )
1754             )
1755             )
1756             )
1757             )
1758             )
1759             )
1760             )
1761             )
1762             )
1763             )
1764             )
1765             )
1766             )
1767             )
1768             )
1769             )
1770             )
1771             )
1772             )
1773             )
1774             )
1775             )
1776             )
1777             )
1778             )
1779             )
1780             )
1781             )
1782             )
1783             )
1784             )
1785             )
1786             )
1787             )
1788             )
1789             )
1790             )
1791             )
1792             )
1793             )
1794             )
1795             )
1796             )
1797             )
1798             )
1799             )
1800             )
1801             )
1802             )
1803             )
1804             )
1805             )
1806             )
1807             )
1808             )
1809             )
1810             )
1811             )
1812             )
1813             )
1814             )
1815             )
1816             )
1817             )
1818             )
1819             )
1820             )
1821             )
1822             )
1823             )
1824             )
1825             )
1826             )
1827             )
1828             )
1829             )
1830             )
1831             )
1832             )
1833             )
1834             )
1835             )
1836             )
1837             )
1838             )
1839             )
1840             )
1841             )
1842             )
1843             )
1844             )
1845             )
1846             )
1847             )
1848             )
1849             )
1850             )
1851             )
1852             )
1853             )
1854             )
1855             )
1856             )
1857             )
1858             )
1859             )
1860             )
1861             )
1862             )
1863             )
1864             )
1865             )
1866             )
1867             )
1868             )
1869             )
1870             )
1871             )
1872             )
1873             )
1874             )
1875             )
1876             )
1877             )
1878             )
1879             )
1880             )
1881             )
1882             )
1883             )
1884             )
1885             )
1886             )
1887             )
1888             )
1889             )
1890             )
1891             )
1892             )
1893             )
1894             )
1895             )
1896             )
1897             )
1898             )
1899             )
1900             )
1901             )
1902             )
1903             )
1904             )
1905             )
1906             )
1907             )
1908             )
1909             )
1910             )
1911             )
1912             )
1913             )
1914             )
1915             )
1916             )
1917             )
1918             )
1919             )
1920             )
1921             )
1922             )
1923             )
1924             )
1925             )
1926             )
1927             )
1928             )
1929             )
1930             )
1931             )
1932             )
1933             )
1934             )
1935             )
1936             )
1937             )
1938             )
1939             )
1940             )
1941             )
1942             )
1943             )
1944             )
1945             )
1946             )
1947             )
1948             )
1949             )
1950             )
1951             )
1952             )
1953             )
1954             )
1955             )
1956             )
1957             )
1958             )
1959             )
1960             )
1961             )
1962             )
1963             )
1964             )
1965             )
1966             )
1967             )
1968             )
1969             )
1970             )
1971             )
1972             )
1973             )
1974             )
1975             )
1976             )
1977             )
1978             )
1979             )
1980             )
1981             )
1982             )
1983             )
1984             )
1985             )
1986             )
1987             )
1988             )
1989             )
1990             )
1991             )
1992             )
1993             )
1994             )
1995             )
1996             )
1997             )
1998             )
1999             )
2000             )

```

```

1239         ])
1240         | _ -> raise (Failure ("unsupported dict type"))
1241     )
1242     else
1243         raise (Failure ("accessassign: expr right of = is not
1244                        dict value type"))
1245     else
1246         raise (Failure ("accessassign: assign expr on left for dict
1247                        is wrong"))
1248     | _ -> raise (Failure ("unsupported access"))
1249 ) in
1250 Block([
1251     c_e1;
1252     c_e2;
1253     c_e3;
1254     call;
1255 ])
1256 | Sast.Return(e, dt) ->
1257 let c_e = translate_expr env e in
1258 let e_result = "v" ^ string_of_int (find_max_index !(List.hd env.
1259 var_inds)) in
1260 Block([
1261     c_e;
1262     Translate.Return(Deref(dt_to_ct dt, Id(Ptr(dt_to_ct dt), e_result)))
1263 ])
1264 | Sast.NodeDef (id, s, dt) ->
1265 let index = "v" ^ string_of_int (find_var id env.var_inds) in
1266 let c_s = translate_expr env s in
1267 let result_s = "v" ^ string_of_int (find_max_index !(List.hd env.
1268 var_inds)) in
1269 (match s with
1270 | Sast.Noexpr ->
1271 Block([
1272     c_s;
1273     Expr (Assign (Id (Node, index), Call (Void, "init_node", [Literal (
1274 Cstring, "")]));
1275     Expr (Assign (Member (Ptr (Void), Id (Void, index), "data"), Literal (
1276 Cstring, ""))))])
1277 | _ ->
1278 Block([
1279     c_s;
1280     Expr (Assign (Id (Node, index), Call (Void, "init_node", [Literal (
1281 Cstring, "")]));
1282     Expr (Assign (Member (Ptr (Void), Id (Void, index), "data"), Deref (
1283 Cstring, Id (Ptr (Cstring), result_s)) ))
1284 ])
1285 )
1286 | Sast.GraphDef (id, sl) ->
1287 let index = "v" ^ string_of_int (find_var id env.var_inds) in
1288 let edge_ops = List.map (fun f -> Expr (translate_expr env f)) sl in
1289

```

```

1285 let rec find_vars stmts = function
1286 | [] -> stmts
1287 | hd :: tl ->
1288   let calls =
1289     (match hd with
1290     | Undir(n1, n2, dt) | Dir(n1, n2, dt)->
1291       let n1_index = "v" ^ string_of_int(find_var n1 env.var_inds) in
1292       let n2_index = "v" ^ string_of_int(find_var n2 env.var_inds) in
1293       [
1294         Expr(Call(Void, "add_node", [Id(Graph, index); Id(Node,
1295           n1_index)]));
1296         Expr(Call(Void, "add_node", [Id(Graph, index); Id(Node,
1297           n2_index)]))
1298       ]
1299     | DirVal(n1, n2, w, dt) | UndirVal(n1, n2, w, dt) ->
1300       let n1_index = "v" ^ string_of_int(find_var n1 env.var_inds) in
1301       let n2_index = "v" ^ string_of_int(find_var n2 env.var_inds) in
1302       [
1303         Expr(Call(Void, "add_node", [Id(Graph, index); Id(Node,
1304           n1_index)]));
1305         Expr(Call(Void, "add_node", [Id(Graph, index); Id(Node,
1306           n2_index)]))
1307       ]
1308     | _ -> raise (Failure ("unexpected type"))
1309   )
1310   in
1311   find_vars ( calls @ stmts) tl
1312 in
1313 let node_add_calls = find_vars [] sl in
1314 Block( (Expr(Assign(Id(Graph, index), Call(Void, "init_graph", []))) ::
1315   edge_ops)
1316   @ node_add_calls
1317 )
1318 | Sast.While (cond, sl) ->
1319   (match cond with
1320   | Binop(e1, op, e2, dt) ->
1321     let c_e1 = translate_expr env e1 in
1322     let e1_cdt = dt_to_ct (get_sexpr_type e1) in
1323     let e1_result = "v" ^ string_of_int (find_max_index !(List.hd env.
1324       var_inds)) in
1325
1326     let c_e2 = translate_expr env e2 in
1327     let e2_cdt = dt_to_ct (get_sexpr_type e2) in
1328     let e2_result = "v" ^ string_of_int (find_max_index !(List.hd env.
1329       var_inds)) in
1330
1331     let c_stmts = List.map (translate_stmt env) sl in
1332     Block([c_e1; c_e2;
1333       While(Translate.Binop(Int,
1334         Deref(e1_cdt, Id(Ptr(e1_cdt), e1_result)),
1335         op,
1336         Deref(e2_cdt, Id(Ptr(e2_cdt), e2_result))
1337       ),

```

```

1332         c_stmts)
1333     ])
1334     | _ -> raise (Failure ("not a while loop condition expression"))
1335 )
1336
1337     (* convert body *)
1338
1339 | Sast.If (cond, s1, s2) ->
1340
1341     (match cond with
1342     | Binop(e1, op, e2, dt) ->
1343         let c_e1 = translate_expr env e1 in
1344         let e1_cdt = dt_to_ct (get_sexpr_type e1) in
1345         let e1_result = "v" ^ string_of_int (find_max_index !(List.hd env.
1346             var_inds)) in
1347
1348         let c_e2 = translate_expr env e2 in
1349         let e2_cdt = dt_to_ct (get_sexpr_type e2) in
1350         let e2_result = "v" ^ string_of_int (find_max_index !(List.hd env.
1351             var_inds)) in
1352
1353         let then_stmts = translate_stmt env s1 in
1354         let else_stmts = translate_stmt env s2 in
1355         Block([c_e1; c_e2;
1356             If(Translate.Binop(Int,
1357                 Deref(e1_cdt, Id(Ptr(e1_cdt), e1_result)),
1358                 op,
1359                 Deref(e2_cdt, Id(Ptr(e2_cdt), e2_result))
1360             ),
1361             [then_stmts],
1362             [else_stmts])
1363         ])
1364     | _ -> raise (Failure ("not a while loop condition expression"))
1365 )
1366 | Sast.For (key, iter, s1) ->
1367     let iter_stype = get_sexpr_type iter in
1368     let iter_ctype = dt_to_ct iter_stype in
1369     let c_iter = translate_expr env iter in (* evaluate the iterable *)
1370     let iter_result = "v" ^ string_of_int (find_max_index !(List.hd env.
1371         var_inds)) in (* result of iterable *)
1372
1373     let loop_var = "v" ^ string_of_int(create_auto env key (Sast.Void))
1374         in
1375     let key_var = "v" ^ string_of_int(create_auto env key iter_stype) in
1376         (* the temp var in "for x in iterable " *)
1377     let csl = List.map (translate_stmt env) s1 in
1378     Block(
1379     c_iter ::
1380     [
1381         (match iter_stype with
1382         | List(dt) ->
1383             let key_var_type = dt_to_ct dt in
1384             let iter_deref = Deref(iter_ctype, Id(Ptr(iter_ctype),
1385                 iter_result)) in

```



```

1380     Block([Vdecl(key_var_type, key_var); (* *)
1381           Vdecl(iter_ctype, loop_var);
1382           For(Assign(Id(iter_ctype, loop_var), iter_deref),
1383               Id(iter_ctype, loop_var),
1384               Assign(Id(iter_ctype, loop_var), Member(iter_ctype,
1385                                                           Id(Void, loop_var), "next")),
1386
1387               Expr(Assign(Id(key_var_type, key_var),
1388                           Deref(key_var_type, Cast(Ptr(key_var_type),
1389                                                       Member(Ptr(Void), Id(Void, loop_var), "
1389                                                           data"))))
1388           )
1389         )
1390         :: csl
1391     )
1392   ])
1393 | Dict(dtk, dtv) ->
1394   let iter_deref = Deref(Ptr(Entry), Id(Ptr(Ptr(Entry)),
1395   iter_result)) in
1396   let int_var = "v" ^ string_of_int(create_auto env "" (Sast
1397   .Num)) in
1398   let for_loop =
1399     For(Assign(Id(Int, int_var), Literal(Int, "0")),
1400         Binop(Int, Id(Int, int_var), Ast.Less, Id(Int, "
1401             TABLE_SIZE")),
1402         Assign(Id(Int, int_var),
1403               Binop(Int, Id(Int, int_var), Ast.Add,
1404                   Literal(Int, "1"))),
1405         [For(Assign(Id(Ptr(Entry), loop_var), Access(Entry,
1406             Assoc(iter_deref), Id(Int, int_var))),
1407             Id(Entry, loop_var),
1408             Assign(Id(Ptr(Entry), loop_var), Member(Entry, Id
1409                 (Void, loop_var), "next")),
1410             ( Expr(Assign(Id(Ptr(Void), key_var),
1411                         Member(Ptr(Void), Id(Void, loop_var), "
1412                             key"))
1413             )
1414           )
1415         :: List.map (translate_stmt env) sl
1416       )
1417     ])
1418   ) in
1419   Block([Vdecl(Int, int_var);
1420         Vdecl(Ptr(Entry), loop_var);
1421         Vdecl(Ptr(Void), key_var);
1422         If(iter_deref, [for_loop], [])
1423       ])
1424 | Node ->
1425   let iter_deref = Deref(Node, Id(Ptr(Node), iter_result)) in
1426   Block([Vdecl(Ptr(Node), key_var);
1427         Expr(Assign(Id(Ptr(Node), key_var), iter_deref))
1428       ] @ csl)
1429 | Graph ->
1430   Block([Vdecl(Node, key_var);

```

```

1425     Vdecl(List(Node), loop_var);
1426     For(Assign(Id(List(Node), loop_var),
1427         Member(List(Node), Id(Void, "*" ^ iter_result), "
            nodes")),
1428         Id(List(Node), loop_var),
1429         Assign(Id(List(Node), loop_var),
1430             Member(List(Node), Id(Void, loop_var), "next")),
1431             Expr(Assign(Id(Node, key_var), Cast(Node, Member(
                Node,
1432                 Id(List(Node), loop_var), "data")))) :: csl
1433             )
1434         ])
1435     | _ -> raise (Failure("for loop iter is not iterable"))
1436     )
1437 ])
1438
1439 | Sast.Fdecl (func) -> Nostmt
1440
1441     (* ***** TODO ***** *)
1442     (*
1443     { crtype = Translate.Void;
1444     cname = "empty";
1445     cformals = [(Int, "argc"); (Ptr(Cstring), "argv)];
1446     cbody = []}
1447     *)
1448
1449 and
1450 translate_fdecl env func = (
1451     (* add the parameter names to the env *)
1452     let formals = func.s_formals in
1453     let rtype = func.s_rtype in
1454
1455     (* add formal variables to local scope variable maps *)
1456     let map_builder fmls m = (List.map (fun f -> m := (StringMap.add (snd f) (
1457         fst f) !m); "") formals) in
1458     let types_map = ref StringMap.empty in
1459     ignore (map_builder formals types_map);
1460     let fml_inds = enum 1 (find_max_index !(List.hd env.var_inds)) (List.map (
1461         fun f -> (snd f)) formals) in
1462     let inds_map = ref (string_map_pairs StringMap.empty fml_inds) in
1463
1464     let func_env = {
1465         var_inds = inds_map :: env.var_inds; (* var names to indices ex. x -> 1
1466         so that we can just refer to it as v1 *)
1467         var_types = types_map :: env.var_types; (* maps a var name to its type ex
1468         . x -> num *)
1469         func_inds = ref StringMap.empty :: env.func_inds; (* func names to
1470         indices ex. x -> 1 so that we can just refer to it as f1 *)
1471         func_obj = ref StringMap.empty :: env.func_obj;
1472         return_type = rtype; (* what should the return type be of the
1473         current scope *)
1474     } in
1475     {crtype = dt_to_ct func.s_rtype;

```

```

1471   cfname = "f" ^ string_of_int(find_var func.s_fname env.func_inde);
1472   cformals = List.map (fun f -> (dt_to_ct (fst f), "v" ^ string_of_int(
      find_var (snd f) func_env.var_inde))) func.s_formals;
1473   cbody = (List.map (fun f -> translate_stmt func_env f) func.s_body)
1474 )
1475 in
1476
1477 (* convert the sast_prg to cast_prg *)
1478 let global_vars = List.map (fun f -> translate_stmt env f) sast_prg.s_globals
      in
1479 let main_func = {crtype = Translate.Int;
1480                 cfname = "main";
1481                 cformals = [(Int, "argc"); (Ptr(Cstring), "argv")];
1482                 cbody = List.map (fun f -> translate_stmt env f) sast_prg.s_main}
1483 in
1484 let cfunc_list = List.map (fun f -> translate_fdecl env f) sast_prg.s_funcs in
1485 {globals = global_vars; cfuncs = List.rev (main_func :: List.rev cfunc_list)}
1486
1487 let print_bindings m =
1488   let bindings = StringMap.bindings m in
1489   let rec printer = function
1490     | [] -> print_endline("")
1491     | (k, v)::tl -> print_endline(k ^ string_of_int(v)) ; printer tl
1492   in
1493   printer bindings

```

Listing 21: analyzer.ml

## A.7 translate.ml

```

1 (* c AST is a library that handles c Asts pretty prints a c file *)
2 module StringMap = Map.Make(String)
3
4 type ctype = | Float | Int | Long | Cstring
5             | Array of ctype
6             | List of ctype
7             | Graph
8             | Node
9             | Ptr of ctype (* pointer to a data type *)
10            | Void
11            | Entry
12
13 type cstmt =
14 | Literal of ctype * string
15 | DictLiteral of ctype * (cstmt * cstmt) list
16 | Id of ctype * string (* ids arget_cexpr_typee ints ex. Id(2)
      -> v2 *)
17 | Binop of ctype * cstmt * Ast.op * cstmt
18 | Assign of cstmt * cstmt (* ex. Assign(2, 5) -> v2 = 5 *)
19 | Call of ctype * string * cstmt list (* return type of the function, function
      name, arguments *) (* Call(3, [Literal(5), Id(3)]) -> f3(5, v3) *)
20 | Access of ctype * cstmt * cstmt (* array access: id[cexpr] *)
21 | Member of ctype * cstmt * string (* id, member *)
22 | Cast of ctype * cstmt (* ex. Cast(Int, Id(f1)) -> (int)(f1) *)

```

```

23 | Deref of ctype * cstmt                                (* ex. *var *)
24 | Ref of ctype * cstmt                                (* ex. &var *)
25 | Block of cstmt list
26 | Expr of cstmt
27 | Vdecl of ctype * string                            (* (type, id) ex. Vdecl(Int, 2) -> int v2
    ; *)
28 | Return of cstmt
29 | If of cstmt * cstmt list * cstmt list
30 | For of cstmt * cstmt * cstmt * cstmt list (* assign, condition, incr, body
    -> ex. for (v1 = 3, v1 < 10; v1 = v1 + 1 *)
31 | While of cstmt * cstmt list
32 | Assoc of cstmt (* wrap the expression in parentheses *)
33 | Nostmt
34
35 type c_func = { ctype : ctype; (* c return type *)
36                 cname : string; (* function name *)
37                 cformals : (ctype * string) list; (* (data type, id) list *)
38                 cbody : cstmt list;
39             }
40
41 type cprogram = {
42     globals : cstmt list; (* global variables -- Note: should ONLY
        be Vdecl list *)
43     cfuncs : c_func list;
44 }
45
46 let rec type_to_str = function
47 | Float -> "float"
48 | Int -> "int"
49 | Long -> "long"
50 | Cstring -> "char *"
51 | Array(dt) -> type_to_str dt ^ "[]"
52 | List(dt) -> "list_t *"
53 | Node -> "node_t *"
54 | Graph -> "graph_t *"
55 | Ptr(dt) -> type_to_str dt ^ "*"
56 | Void -> "void"
57 | Entry -> "entry_t"
58
59 let fmt_str = function
60 | Float -> "%f"
61 | Int -> "%d"
62 | Cstring -> "%s"
63 | _ -> raise (Failure ("can't print other types directly"))
64
65 let rec get_cexpr_type = function
66 | Literal(dt, str) -> dt
67 (* | ListLiteral(dt, el) -> dt *)
68 | DictLiteral(dt, tl) -> dt
69 | Id(dt, id) -> dt
70 | Binop(dt, e1, op, e2) -> dt
71 | Assign(id, e1) -> Void
72 | Call(dt, id, el) -> dt
73 | Access(dt, id, e) -> dt

```

```

74 | Member(dt, stmt, m) -> dt
75 | Cast(dt, e) -> dt
76 | Ref(dt, e) -> dt
77 | Deref(dt, e) -> dt
78 | Assoc(e) -> get_cexpr_type e
79 | _ -> Void
80
81 let rec stmt_type_to_str = function
82 | Literal(dt, str) -> "Literal<" ^ type_to_str dt ^ ">"
83 | DictLiteral(dt, tl) -> "DictLiteral<" ^ type_to_str dt ^ ">"
84 | Id(dt, id) -> "Id<" ^ type_to_str dt ^ ">"
85 | Binop(dt, e1, op, e2) -> "Binop<" ^ type_to_str dt ^ ">"
86 | Assign(id, e1) -> "Assign<" ^ stmt_type_to_str id ^ ">"
87 | Call(dt, id, el) -> "Call<" ^ type_to_str dt ^ ">"
88 | Access(dt, id, e) -> "Access<" ^ type_to_str dt ^ ">"
89 | Member(dt, stmt, m) -> "Member<" ^ type_to_str dt ^ ">"
90 | Cast(dt, e) -> "Cast<" ^ type_to_str dt ^ ">"
91 | Ref(dt, e) -> "Ref<" ^ type_to_str dt ^ ">"
92 | Deref(dt, e) -> "Deref<" ^ type_to_str dt ^ ">"
93 | Block(sl) -> "Block"
94 | Expr(e) -> "Expr:" ^ stmt_type_to_str e
95 | Vdecl(dt, id) -> "Vdecl<" ^ type_to_str dt ^ ">"
96 | Return(e) -> "Return:" ^ stmt_type_to_str e
97 | If(cond, ell, el2) -> "If-then-Else"
98 | For(assign, cond, incr, sl) -> "For"
99 | While(cond, sl) -> "While"
100 | Assoc(e) -> stmt_type_to_str e
101 | Nostmt -> "Nostmt"
102
103 let op_to_str = function
104 | Ast.Add -> "+"
105 | Ast.Sub -> "-"
106 | Ast.Mult -> "*"
107 | Ast.Div -> "/"
108 | Ast.Equal -> "=="
109 | Ast.Neq -> "!="
110 | Ast.Less -> "<"
111 | Ast.Leq -> "<="
112 | Ast.Greater -> ">"
113 | Ast.Geq -> ">="
114 | Ast.LogAnd -> "&&"
115 | Ast.LogOr -> "||"
116
117 (* takes a c datatype and returns the print format string *)
118 let get_fmt_str = function
119 | Float -> "%.3f"
120 | Int -> "%d"
121 | Long -> "%l"
122 | Cstring -> "%s"
123 | Node -> raise (Failure "type node can't be directly printed")
124 | Entry | Graph -> raise (Failure "type requires iterable print handling")
125 | List(dt) | Array(dt) -> raise (Failure "type requires iterable print
    handling")
126 | Void -> raise (Failure ("can't directly print Void"))

```

```

127 | Ptr(dt) -> raise (Failure "can't directly print pointer")
128
129 let cvar_cnt = ref 0
130
131 let rec translate_stmt = function
132 | Literal(dt, v) ->
133   (match dt with
134   | Float -> v
135   | Int -> v
136   | Cstring -> "\"" ^ v ^ "\""
137   | Array(adt) -> v
138   | Void -> if v = "NULL" then v else raise (Failure "Void lit should only be
139     'NULL'")
139   | _ -> raise (Failure "invalid C literal type")
140   )
141 | DictLiteral(dt, el) -> translate_stmt (Literal(Cstring, "TODO: dict literal"
142   ))
143 | Id(dt, id) -> id
144 | Binop(dt, e1, op, e2) ->
145   (* check if either e1 is a string or e2 is a string:
146     different operation: concatenation
147     *)
148   translate_stmt e1 ^ " " ^ op_to_str(op) ^ " " ^ translate_stmt e2
149 | Assign(target, e) -> (translate_stmt target) ^ " = " ^ translate_stmt e
150 | Call(dt, id, el) ->
151   id ^ "(" ^ (String.concat ", " (List.map translate_stmt el)) ^ ")"
152 | Access(dt, id, e) -> (translate_stmt id) ^ "[" ^ (translate_stmt e) ^ "]"
153 | Member(dt, id, m) -> (translate_stmt (Assoc(id))) ^ "->" ^ m
154 | Cast(dt, e) -> "(" ^ type_to_str dt ^ ")" ^ translate_stmt e ^ ")"
155 | Ref(dt, e) -> "&(" ^ translate_stmt e ^ ")"
156 | Deref(dt, e) -> "*(" ^ translate_stmt e ^ ")"
157 | Block(sl) -> String.concat "\n" (List.map translate_stmt sl)
158 | Expr(e) -> translate_stmt e ^ ";"
159 | Vdecl(dt, id) ->
160   (match dt with
161   | Ptr(Ptr(Entry)) -> type_to_str dt ^ " " ^ id ^ " = NULL;"
162   | List(vdt) -> type_to_str dt ^ " " ^ id ^ " = NULL;"
163   | Graph -> type_to_str dt ^ " " ^ id ^ " = NULL;"
164   | _ -> type_to_str dt ^ " " ^ id ^ ";"
165   )
166
167 | Return(e) -> "return " ^ translate_stmt e ^ ";"
168 | If(cond, sl1, sl2) -> "if (" ^ translate_stmt cond ^ ") {\n" ^
169   String.concat "\n" (List.map translate_stmt sl1) ^
170   "\n} else {\n" ^
171   String.concat "\n" (List.map translate_stmt sl2) ^
172   "\n}"
173 | For(init, cond, incr, sl) -> "for (" ^ translate_stmt init ^ "; " ^
174   translate_stmt cond ^ "; " ^
175   translate_stmt incr ^ ") {\n" ^
176   String.concat "\n" (List.map translate_stmt sl) ^
177   "\n}"
178 | While(cond, sl) -> "while (" ^ translate_stmt cond ^ ") {\n" ^

```

```

179   String.concat "\n" (List.map translate_stmt sl) ^
180   "\n}"
181 | Assoc(e) -> "(" ^ (translate_stmt e) ^ ")"
182 | Nostmt -> ""
183
184
185 let translate_func func =
186   (type_to_str func.crtype) ^ " " ^ func.cfname ^ " (" ^
187   String.concat ", " (List.map (fun f -> (type_to_str (fst f)) ^ " " ^ snd f)
188     func.cformals) ^
189   "\n{\n" ^
190   String.concat "\n" (List.map translate_stmt func.cbody) ^
191   "\n}\n"
192
193 (* eventually won't be used by analyzer.ml *)
194 let string_of_cfunc func =
195   (type_to_str func.crtype) ^ " " ^ func.cfname ^ " (" ^
196   String.concat ", " (List.map (fun f -> (type_to_str (fst f)) ^ " " ^ snd f)
197     func.cformals) ^
198   "\n{\n" ^
199   (String.concat "\n" (List.map translate_stmt func.cbody)) ^
200   "\n}\n"
201
202 let translate_c (globals, cfuncs) =
203   (* "\"graph.h\"" *)
204   let libs = ["<stdio.h>"; "<stdlib.h>"; "<string.h>";
205     "<dict.h>"]
206   in
207   (* now we are going to translate a program *)
208   (String.concat "\n" (List.map (fun f -> "#include " ^ f) libs)) ^
209   "\n" ^
210   (String.concat "\n" (List.map translate_stmt globals)) ^
211   "\n" ^
212   (String.concat "\n" (List.map translate_func cfuncs))

```

Listing 22: translate.ml

## A.8 compile.ml

```

1 (* ties together parsing scanning ast sast the whole fucking thing *)
2 open Ast
3 open Sast
4 open TypeConverter
5 open Translate
6 open Analyzer
7
8 (* translate version *)
9 let _ =
10 let lexbuf = Lexing.from_channel stdin in
11 let ast_prg =
12   (try
13     (Parser.program Scanner.token lexbuf) (* outputs the Ast from parsing
14       and scanning *)

```

```

14   with exn ->
15       let curr = lexbuf.Lexing.lex_curr_p in
16       let line = curr.Lexing.pos_lnum in
17       let cnum = curr.Lexing.pos_cnum - curr.Lexing.pos_bol in
18       let tok = Lexing.lexeme lexbuf in
19       raise (Failure ("Parsing error: line " ^ string_of_int(line) ^ ", char
20                       " ^ string_of_int(cnum) ^ ", token " ^ tok))
21   )
22   in
23
24   let print_decl = {
25       s_fname = "print";
26       s_rtype = Sast.Void;
27       s_formals = [];
28       s_body = [];
29   } in
30   let range_decl = {
31       s_fname = "range";
32       s_rtype = Sast.List(Sast.Num);
33       s_formals = [];
34       s_body = [];
35   } in
36
37   (* set up default environment *)
38   let sast_env =
39       (* built-in function set-up *)
40       let bf_names = [ "print"; "range"; ] in
41       let bf_inds = enum 1 1 bf_names in
42       let bf_ind_map = ref (string_map_pairs StringMap.empty bf_inds) in
43       let bf_fdecl_map = ref (string_map_pairs StringMap.empty [(print_decl, "
44                               print"); (range_decl, "range")]) in
45       (* build default symbol tables: *)
46       {var_types = [ref StringMap.empty];
47        var_inds = [ref StringMap.empty];
48        func_obj = [bf_fdecl_map];
49        func_inds = [bf_ind_map];
50        return_type = Sast.Void} in
51
52   (* convert Ast to Sast *)
53   let sast_prg = convert_ast { cmds = List.rev ast_prg.cmds } sast_env in
54
55   (* massage Sast into a form more suitable for C Ast *)
56   (* i.e. split up variable declarations, function definitions, and other *)
57   let sifted_prg = stmt_sifter {s_globals = []; s_main = []; s_funcs = []}
58       sast_prg.s_cmds in
59
60   (* construct Sast program object from sifted *)
61   let sifted_prg = {s_globals = List.rev sifted_prg.s_globals;
62                   s_main = List.rev sifted_prg.s_main;
63                   s_funcs = List.rev sifted_prg.s_funcs} in
64
65   (* set up default environ *)
66   let trans_env =
67       let bf_names = [ "print"; "range"; "len"; "min"; "max" ] in

```



```

65     let bf_inds = enum 1 1 bf_names in
66     let bf_ind_map = ref (string_map_pairs StringMap.empty bf_inds) in
67     let bf_fdecl_map = ref (string_map_pairs StringMap.empty [(print_decl, "
68         print"); (range_decl, "range")]) in
69     {var_types = [ref StringMap.empty];
70       var_inds = [ref StringMap.empty];
71       func_obj = [bf_fdecl_map];
72       func_inds = [bf_ind_map];
73       return_type = Sast.Void} in
74
75     (* add declared functions to symbol tables *)
76     let rec func_def_adder env = function
77     | [] -> ignore()
78     | hd :: tl ->
79         (List.hd env.func_obj) := StringMap.add hd.s_fname hd !(List.hd env.
80             func_obj);
81         (List.hd env.func_inds) := StringMap.add hd.s_fname (find_max_index !(
82             List.hd env.func_inds)+1) !(List.hd env.func_inds); (* add index map
83             *)
84         ignore(func_def_adder env tl)
85     in
86     ignore(func_def_adder trans_env sifted_prg.s_funcs);
87
88     (* convert Sast to C Ast *)
89     let cprg = translate(trans_env, sifted_prg) in
90
91     (* output C code from C Ast *)
92     print_endline (translate_c(cprg.globals, cprg.cfuncs))

```

Listing 23: compile.ml

## B C Library Files

### B.1 node.h

```

1 struct node;
2 typedef struct node node_t;
3 typedef struct entry entry_t;
4
5 struct node {
6     char *data;
7     entry_t **in;
8     entry_t **out;
9 };
10
11 struct entry {
12     void *key;
13     void *value;
14     struct entry *next;
15 };
16
17 /* initialize a new node that contains *data */
18 node_t *init_node(char *data);

```

```

19
20 /* compares node data */
21 int node_compare(node_t *a, node_t *b, int (* comp) (void *a, void *b));
22
23 /* create undirected edge of weight 0 */
24 void connect_undir(node_t *a, node_t *b);
25
26 /* create weighted undirected edge */
27 void connect_undir_weighted(node_t *a, node_t *b, float weight);
28
29 /* create directed edge of weight 0 */
30 void connect_dir(node_t *src, node_t *dst);
31
32 /* create weighted directed edge */
33 void connect_dir_weighted(node_t *src, node_t *dst, float weight);
34
35 /* remove directed edge from src to dst */
36 void remove_dir_edge(node_t *src, node_t *dst);
37
38 /* remove undirected edge between a and b */
39 void remove_undir_edge(node_t *a, node_t *b);
40
41 /* deallocate node */
42 void free_node(node_t *n);

```

Listing 24: node.h

## B.2 node.c

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include "dict.h"
4
5 /* initialize a new node that contains *data */
6 node_t *init_node(char *data) {
7     node_t *n = (node_t *) malloc(sizeof(node_t));
8     n->data = data;
9     n->in = init_dict();
10    n->out = init_dict();
11    return n;
12 }
13
14 void free_node(node_t *n) {
15     printf("freeing node at %x\n", (int) n);
16     free(n->data);
17     free(n);
18 }
19
20 /* TODO decide on default behavior */
21 /* compares node data, should return 0 if a == b, 1 if a > b, and -1 if a < b
22    */
23 int node_compare(node_t *a, node_t *b, int (* comp)(void *a, void *b)) {
24     if(comp == NULL)

```

```

25     return *(a->data) == *(b->data);
26     */
27     return comp(a->data, b->data);
28 }
29
30 /* create undirected edge of weight 0 */
31 void connect_undir(node_t *a, node_t *b) {
32     connect_dir(a, b);
33     connect_dir(b, a);
34 }
35
36 /* create weighted undirected edge */
37 void connect_undir_weighted(node_t *a, node_t *b, float weight) {
38     connect_dir_weighted(a, b, weight);
39     connect_dir_weighted(b, a, weight);
40 }
41
42 /* create directed edge of weight 0 */
43 void connect_dir(node_t *src, node_t *dst) {
44     connect_dir_weighted(src, dst, 0);
45 }
46
47 /* create weighted directed edge */
48 void connect_dir_weighted(node_t *src, node_t *dst, float weight) {
49
50     void *a = malloc(sizeof(float));
51     *(float *)a = weight;
52     /* add dst to src->out */
53     put_node(src->out, dst, a);
54
55     /* add src to dst->in */
56     put_node(dst->in, src, a);
57
58     /*
59     edgelist_t *e = (edgelist_t *) malloc(sizeof(edgelist_t));
60     e->node = dst;
61     e->weight = weight;
62     e->previous = NULL;
63     e->next = src->out;
64     if(src->out != NULL)
65         src->out->previous = e;
66     src->out = e;
67
68     add src to dst->in
69     edgelist_t *f = (edgelist_t *) malloc(sizeof(edgelist_t));
70     f->node = src;
71     f->weight = weight;
72     f->previous = NULL;
73     f->next = dst->in;
74     if(dst->in != NULL)
75         dst->in->previous = f;
76     dst->in = f;
77     */
78 }

```

```

79
80 /* remove directed edge from src to dst */
81 void remove_dir_edge(node_t *src, node_t *dst) {
82     /*
83     edgelist_t *e = src->out;
84     while(e && e->node != dst)
85         e = e->next;
86     if(!e) {
87         printf("there is no edge from %s to %s\n", (char *) src->data, (char *)
88             dst->data);
89         return;
90     }
91     edgelist_t *f;
92     f = e->node->in;
93     while(f && f->node != src)
94         f = f->next;
95     if(!f)
96         printf("f is NULL\n");
97     else {
98         if(f->previous)
99             f->previous->next = f->next;
100         if(f->next)
101             f->next->previous = f->previous;
102         if(!f->next && !f->previous) {
103             e->node->in = 0;
104         }
105         free(f);
106     }
107     if(e->previous)
108         e->previous->next = e->next;
109     if(e->next)
110         e->next->previous = e->previous;
111     if(!(e->next || e->previous))
112         src->out = 0;
113     free(e);
114     e = NULL;
115     */
116 }
117
118 /* remove undirected edge between a and b */
119 void remove_undir_edge(node_t *a, node_t *b) {
120     remove_dir_edge(a, b);
121     remove_dir_edge(b, a);
122 }
123
124 char * value(node_t *n) {
125     return (n->data);
126 }

```

Listing 25: node.c

### B.3 graph.h

```

1 #include "node.h"

```

```

2
3 typedef struct list list_t;
4 struct list {
5     struct list *next;
6     struct list *previous;
7     void *data;
8 };
9
10 typedef struct graph graph_t;
11 struct graph {
12     list_t *nodes;
13     int count;
14 };
15
16 graph_t *init_graph();
17
18 void free_graph(graph_t * graph);
19
20 int contains(graph_t *graph, void *data, int (* comp)(void *a, void *b));
21
22 graph_t *add_node(graph_t *graph, const node_t *node);
23
24 int remove_node(graph_t *graph, node_t *node);
25
26 graph_t *plus(const graph_t *a, const graph_t *b);
27
28 graph_t *plus_equals(graph_t *a, const graph_t *b);
29
30 graph_t *minus(const graph_t *left, const graph_t *right);
31
32 graph_t *graph_copy(const graph_t *src);
33
34 int graph_equals(const graph_t *a, const graph_t *b);
35
36 graph_t *graph_plus_node(const graph_t *g, const node_t *n);
37
38 graph_t *node_plus_node(const node_t *n1, const node_t *n2);

```

Listing 26: graph.h

## B.4 graph.c

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include "graph.h"
4
5 /* initialize empty graph */
6 graph_t *init_graph() {
7     graph_t *g = (graph_t *) malloc(sizeof(graph_t));
8     g->nodes = NULL;
9     g->count = 0;
10    return g;
11 }
12

```

```

13 /* deallocate graph */
14 void free_graph(graph_t * g) {
15     if(g == NULL)
16         return;
17     list_t *temp = g->nodes;
18     while(temp->next) {
19         temp = temp->next;
20         free(temp->previous);
21     }
22     free(temp);
23     free(g);
24 }
25
26 /* check if graph contains data */
27 int contains(graph_t *g, void *data, int (* comp)(void *a, void *b)) {
28     list_t *temp = g->nodes;
29     while(temp)
30         if(comp(((node_t *) temp->data)->data, data))
31             return 1;
32     return 0;
33 }
34
35 /* add a node to g by iterating through the list, returning if the node is
    found, and if not, adding it to the end */
36 graph_t *add_node(graph_t *g, const node_t *node) {
37     if(g) {
38         list_t *n = (list_t *)malloc(sizeof(list_t));
39         n->data = (void *) node;
40         list_t *temp = g->nodes;
41         /* make temp point to last list_t in g->nodes */
42         if(temp) {
43             while(temp->next) {
44                 if(temp->data == node)
45                     return g;
46                 temp = temp->next;
47             }
48             temp->next = n;
49         } else {
50             g->nodes = n;
51         }
52         n->previous = temp;
53         n->next = NULL;
54         g->count++;
55     } else {
56         g = init_graph();
57         add_node(g, node);
58     }
59     return g;
60 }
61
62
63 /* returns 0 on success, 1 if node not found */
64 int remove_node(graph_t *g, node_t *n) {
65     if(n == NULL)

```

```

66     return 1;
67     list_t *temp = g->nodes;
68     while(temp && temp->data != n)
69         temp = temp->next;
70     if(temp == NULL)
71         return 1;
72     if(temp->previous)
73         temp->previous->next = temp->next;
74     else
75         g->nodes = temp->next;
76     if(temp->next)
77         temp->next->previous = temp->previous;
78     if(!temp->next && !temp->previous) {
79         g->nodes = 0;
80     }
81     free(temp);
82     g->count--;
83     return 0;
84 }
85
86 /* returns a graph containing all nodes in *a and all nodes in *b */
87 graph_t *plus(const graph_t *a, const graph_t *b) {
88     graph_t *g = (graph_t *) malloc(sizeof(graph_t));
89     g->nodes = NULL;
90     g->count = 0;
91     plus_equals(g, a);
92     plus_equals(g, b);
93     return g;
94 }
95
96 /* adds all nodes from *b to *a. returns a */
97 graph_t *plus_equals(graph_t *a, const graph_t *b) {
98     list_t *temp;
99     for(temp = b->nodes; temp; temp = temp->next)
100         add_node(a, temp->data);
101     return a;
102 }
103
104 /* removes all nodes of *right that exist in *left from *left */
105 graph_t *minus(const graph_t *left, const graph_t *right) {
106     list_t *temp;
107     graph_t *copy = graph_copy(left);
108     for(temp = right->nodes; temp; temp = temp->next) {
109         printf("removing temp = %x\n", (int) temp);
110         int i = remove_node(copy, temp->data);
111         if(i)
112             printf("not removed!\n");
113     }
114     return copy;
115 }
116
117 graph_t *graph_copy(const graph_t *src) {
118     graph_t *g = init_graph();
119     list_t *temp;

```

```

120     for(temp = src->nodes; temp; temp = temp->next)
121         add_node(g, temp->data);
122     return g;
123 }
124
125 int graph_equals(const graph_t *a, const graph_t *b) {
126     if(a == b)
127         return 1;
128     const list_t *temp_a = a->nodes;
129     const list_t *temp_b = b->nodes;
130     while(temp_a && temp_b) {
131         if(temp_a->data != temp_b->data)
132             return 0;
133         temp_a = temp_a->next;
134         temp_b = temp_b->next;
135     }
136     if(temp_a || temp_b)
137         return 0;
138     return 1;
139 }
140
141 graph_t *graph_plus_node(const graph_t *g, const node_t *n) {
142     graph_t *copy;
143     if(g)
144         copy = graph_copy(g);
145     else
146         copy = init_graph();
147     add_node(copy, n);
148     return copy;
149 }
150 }
151
152 graph_t *node_plus_node(const node_t *n1, const node_t *n2) {
153     graph_t *ret = init_graph();
154     add_node(ret, n1);
155     add_node(ret, n2);
156     return ret;
157 }

```

Listing 27: graph.c

## B.5 list.h

```

1  #include "graph.h"
2
3  /* copy constructors */
4  list_t *string_list_copy(const list_t *src);
5
6  list_t *num_list_copy(const list_t *src);
7
8  list_t *graph_list_copy(const list_t *src);
9
10 list_t *node_list_copy(const list_t *src);
11

```



```

12 /*
13 list_t *dict_copy(list_t *src);
14
15 list_t *other_copy(list_t *src);
16 */
17
18 /* insertion */
19 list_t *string_add_front(list_t *l, char *data);
20
21 list_t *num_add_front(list_t *l, float *data);
22
23 list_t *graph_add_front(list_t *l, graph_t *data);
24
25 list_t *node_add_front(list_t *l, node_t *data);
26
27 /*
28 list_t *dict_add_front(list_t *l, entry_t **data);
29
30 list_t *other_add_front(list_t *l, void *data);
31
32 */
33 list_t *string_add_back(list_t *l, char *data);
34
35 list_t *num_add_back(list_t *l, float *data);
36
37 list_t *graph_add_back(list_t *l, graph_t *data);
38
39 list_t *node_add_back(list_t *l, node_t *data);
40
41 /*
42 list_t *dict_add_back(list_t *l, entry_t **data);
43
44 list_t *other_add_back(list_t *l, void *data);
45
46 */
47 /* concatenation */
48 list_t *string_list_concat(const list_t *target, const list_t *src);
49
50 list_t *num_list_concat(const list_t *target, const list_t *src);
51
52 list_t *graph_list_concat(const list_t *target, const list_t *src);
53
54 list_t *node_list_concat(const list_t *target, const list_t *src);
55
56 /*
57 void dict_list_concat(list_t *target, const list_t *src);
58
59 */
60 /* comparison */
61 int string_list_equals(const list_t *a, const list_t *b);
62
63 int num_list_equals(const list_t *a, const list_t *b);
64
65 int node_list_equals(const list_t *a, const list_t *b);

```

```

66
67 int graph_list_equals(const list_t *a, const list_t *b);
68
69 /*
70 int other_list_equals(const list_t *a, const list_t *b, int (*comp)(void *a,
    void *b));
71
72 */
73
74 /* dequeue/pop */
75
76 list_t *pop(list_t *l);
77
78 /* peek */
79 void *peek(list_t *l);
80
81 /* freeing */
82 void free_list(list_t *r);
83
84 /*
85 void string_free_list(list_t *r);
86
87 void graph_free_list(list_t *r);
88
89 void node_free_list(list_t *r);
90 */
91
92 /*
93 void dict_free_list(list_t *r);
94
95 void other_free_list(list_t *r);
96
97 */
98 /* other */
99 list_t *range(int a, int b);
100
101 void *list_access(const list_t *l, int i);
102
103 void print_range(list_t *r);
104
105 void print_strings(list_t *r);
106
107 void free_list(list_t *r);
108
109 void free_range(list_t *r);
110
111 void num_index_insert(list_t *l, int i, float *a);
112
113 void string_index_insert(list_t *l, int i, char *a);
114
115 void node_index_insert(list_t *l, int i, node_t *a);
116
117 void graph_index_insert(list_t *l, int i, graph_t *a);
118

```

```

119 float num_list_min(list_t *l);
120
121 float num_list_max(list_t *l);
122
123 char *string_list_min(list_t *l);
124
125 char *string_list_max(list_t *l);
126
127 int list_len(list_t *l);

```

Listing 28: list.h

## B.6 list.c

```

1 #include "dict.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 static list_t *add_front(list_t *l, void *data, void *(*copy)(void *src)) {
7     list_t *new_node = (list_t *) malloc(sizeof(list_t));
8     if(copy)
9         new_node->data = copy(data);
10    else
11        new_node->data = data;
12    new_node->previous = NULL;
13    new_node->next = l;
14    if(l)
15        l->previous = new_node;
16    return new_node;
17 }
18
19 static void *void_strcpy(void *src) {
20     int len = strlen(src);
21     char *target = malloc(sizeof(char) * len + 1);
22     strncpy((char *) target, (char *) src, len + 1);
23     target[len] = 0;
24     return (void *) target;
25 }
26
27 list_t *string_add_front(list_t *l, char *data) {
28     return add_front(l, (void *) data, void_strcpy);
29 }
30
31 static void *float_copy(void *src) {
32     float *dst = malloc(sizeof(float));
33     *dst = *(float *) src;
34     return dst;
35 }
36
37 list_t *num_add_front(list_t *l, float *data) {
38     return add_front(l, (void *) data, float_copy);
39 }
40

```

```

41 static void *void_graph_copy(void *src) {
42     return (void *) graph_copy(src);
43 }
44
45 list_t *graph_add_front(list_t *l, graph_t *data) {
46     return add_front(l, (void *) data, void_graph_copy);
47 }
48
49 list_t *node_add_front(list_t *l, node_t *data) {
50     return add_front(l, (void *) data, NULL);
51 }
52
53 /*
54 static void *void_dict_copy(entry_t **src) {
55     return dict_copy(src);
56 }
57
58 list_t *dict_add_front(list_t *l, entry_t **data) {
59     return add_front(l, (void *) data, void_dict_copy);
60 }
61
62 list_t *other_add_front(list_t *l, void *data) {
63     return add_front(l, data, NULL);
64 }
65 */
66
67 static list_t *add_back(list_t *l, void *data, void *(*copy)(void *src)) {
68     list_t *new_node = (list_t *) malloc(sizeof(list_t));
69     if(copy)
70         new_node->data = copy(data);
71     else
72         new_node->data = data;
73     new_node->next = NULL;
74     if(l) {
75         list_t *temp = l;
76         while(temp->next)
77             temp = temp->next;
78         temp->next = new_node;
79         new_node->previous = temp;
80         return l;
81     }
82     new_node->previous = NULL;
83     return new_node;
84 }
85
86 list_t *string_add_back(list_t *l, char *data) {
87     return add_back(l, data, void_strcpy);
88 }
89
90 list_t *num_add_back(list_t *l, float *data) {
91     return add_back(l, data, float_copy);
92 }
93
94 list_t *graph_add_back(list_t *l, graph_t *data) {

```

```

95     return add_back(l, data, void_graph_copy);
96 }
97
98 list_t *node_add_back(list_t *l, node_t *data) {
99     return add_back(l, data, NULL);
100 }
101
102 list_t *pop(list_t *l) {
103     printf("l is null");
104     if(!l) {
105         return NULL;
106     }
107     list_t *head = l->next;
108     if(head)
109         head->previous = NULL;
110     return head;
111 }
112
113 void *peek(list_t *l) {
114     return l->data;
115 }
116
117 list_t *range(int a, int b) {
118     list_t *r = NULL;
119     int i;
120     list_t *t;
121     float *j;
122     int sign = a > b ? 1 : -1;
123     for(i = b; i != a + sign; i += sign) {
124         t = (list_t *) malloc(sizeof(list_t));
125         j = (float *) malloc(sizeof(float));
126         *j = (float) i;
127         t->data = j;
128         if(r)
129             r->previous = t;
130         t->next = r;
131         r = t;
132     }
133     return r;
134 }
135
136 void free_range(list_t *r) {
137     if(r->next)
138         for(r = r->next; r->next; r = r->next) {
139             free(r->previous->data);
140             free(r->previous);
141         }
142     free(r->data);
143     free(r);
144 }
145
146 void print_range(list_t *r) {
147     printf("[");
148     for(; r; r = r->next)

```

```

149     if(r->next)
150         printf("%g, ", *((float *) r->data));
151     else
152         printf("%g", *((float *) r->data));
153     printf("]\n");
154 }
155
156 void print_strings(list_t *r) {
157     printf("[");
158     for(; r; r = r->next)
159         if(r->next)
160             printf("%s, ", ((char *) r->data));
161         else
162             printf("%s]\n", ((char *) r->data));
163 }
164
165 void free_list(list_t *r) {
166     if(r->next)
167         for(r = r->next; r->next; r = r->next) {
168             free(r->previous->data);
169             free(r->previous);
170         }
171     else
172         free(r->data);
173     free(r);
174 }
175
176 list_t *num_list_copy(const list_t *src) {
177     if(!src)
178         return NULL;
179     list_t *ret = NULL;
180     const list_t *temp;
181     for(temp = src; temp; temp = temp->next) {
182         ret = num_add_back(ret, (float *) temp->data);
183     }
184     return ret;
185 }
186
187 list_t *string_list_copy(const list_t *src) {
188     if(!src)
189         return NULL;
190     list_t *ret = NULL;
191     const list_t *temp;
192     for(temp = src; temp; temp = temp->next) {
193         ret = string_add_back(ret, (char *) temp->data);
194     }
195     return ret;
196 }
197
198 list_t *node_list_copy(const list_t *src) {
199     if(!src)
200         return NULL;
201     list_t *ret = NULL;
202     const list_t *temp;

```

```

203     for(temp = src; temp; temp = temp->next) {
204         ret = node_add_back(ret, (node_t *) temp->data);
205     }
206     return ret;
207 }
208
209 list_t *graph_list_copy(const list_t *src) {
210     if(!src)
211         return NULL;
212     list_t *ret = NULL;
213     const list_t *temp;
214     for(temp = src; temp; temp = temp->next) {
215         ret = graph_add_back(ret, (graph_t *) temp->data);
216     }
217     return ret;
218 }
219
220 list_t *num_list_concat(const list_t *target, const list_t *src) {
221     list_t *new_list = num_list_copy(target);
222     if(new_list) {
223         list_t *temp;
224         for(temp = new_list; temp->next; temp = temp->next);
225         temp->next = num_list_copy(src);
226         return new_list;
227     }
228     return num_list_copy(src);
229 }
230
231 list_t *string_list_concat(const list_t *target, const list_t *src) {
232     list_t *new_list = num_list_copy(target);
233     if(new_list) {
234         list_t *temp;
235         for(temp = new_list; temp->next; temp = temp->next);
236         temp->next = num_list_copy(src);
237         return new_list;
238     }
239     return num_list_copy(src);
240 }
241
242 list_t *node_list_concat(const list_t *target, const list_t *src) {
243     list_t *new_list = num_list_copy(target);
244     if(new_list) {
245         list_t *temp;
246         for(temp = new_list; temp->next; temp = temp->next);
247         temp->next = num_list_copy(src);
248         return new_list;
249     }
250     return num_list_copy(src);
251 }
252
253 list_t *graph_list_concat(const list_t *target, const list_t *src) {
254     list_t *new_list = num_list_copy(target);
255     if(new_list) {
256         list_t *temp;

```

```

257     for(temp = new_list; temp->next; temp = temp->next);
258     temp->next = num_list_copy(src);
259     return new_list;
260 }
261 return num_list_copy(src);
262 }
263
264 void *list_access(const list_t *l, int i) {
265     const list_t *temp;
266     int j = 0;
267     for(temp = l; temp; temp = temp->next) {
268         if(j == i)
269             return temp->data;
270         j++;
271     }
272     return NULL;
273 }
274
275 int string_list_equals(const list_t *a, const list_t *b) {
276     const list_t *temp_a = a;
277     const list_t *temp_b = b;
278     while(temp_a && temp_b) {
279         if(strcmp((char *) temp_a->data, (char *) temp_b->data))
280             return 0;
281         temp_a = temp_a->next;
282         temp_b = temp_b->next;
283     }
284     if(temp_a || temp_b)
285         return 0;
286     return 1;
287 }
288
289 int num_list_equals(const list_t *a, const list_t *b) {
290     const list_t *temp_a = a;
291     const list_t *temp_b = b;
292     while(temp_a && temp_b) {
293         if(!float_equals(*(float *) temp_a->data, *(float *) temp_b->data))
294             return 0;
295         temp_a = temp_a->next;
296         temp_b = temp_b->next;
297     }
298     if(temp_a || temp_b)
299         return 0;
300     return 1;
301 }
302
303 int node_list_equals(const list_t *a, const list_t *b) {
304     const list_t *temp_a = a;
305     const list_t *temp_b = b;
306     while(temp_a && temp_b) {
307         if(temp_a->data != temp_b->data)
308             return 0;
309         temp_a = temp_a->next;
310         temp_b = temp_b->next;

```



```

311     }
312     if(temp_a || temp_b)
313         return 0;
314     return 1;
315 }
316
317 int graph_list_equals(const list_t *a, const list_t *b) {
318     const list_t *temp_a = a;
319     const list_t *temp_b = b;
320     while(temp_a && temp_b) {
321         if(!graph_equals((graph_t *) temp_a->data, (graph_t *) temp_b->data))
322             return 0;
323         temp_a = temp_a->next;
324         temp_b = temp_b->next;
325     }
326     if(temp_a || temp_b)
327         return 0;
328     return 1;
329 }
330
331 int other_list_equals(const list_t *a, const list_t *b, int (*comp)(void *a,
332     void *b)) {
333     const list_t *temp_a = a;
334     const list_t *temp_b = b;
335     while(temp_a && temp_b) {
336         if(comp(temp_a->data, temp_b->data))
337             return 0;
338         temp_a = temp_a->next;
339         temp_b = temp_b->next;
340     }
341     if(temp_a || temp_b)
342         return 0;
343     return 1;
344 }
345 static void index_insert(list_t *l, int i, void *data, void *(*copy)(void *src
346     )) {
347     int j = 0;
348     while(j < i) {
349         j++;
350         if(l->next)
351             l = l->next;
352         else
353             break;
354     }
355     if(l->next) {
356         if(copy)
357             l->data = copy(data);
358         else
359             l->data = data;
360     } else if(j == i) {
361         l->next = (list_t *) malloc(sizeof(list_t));
362         l->next->next = NULL;
363         l->next->previous = l;

```

```

363     if(copy)
364         l->next->data = copy(data);
365     else
366         l->next->data = data;
367 }
368 }
369
370 void num_index_insert(list_t *l, int i, float *a) {
371     index_insert(l, i, a, float_copy);
372 }
373
374 void string_index_insert(list_t *l, int i, char *a) {
375     index_insert(l, i, a, void_strcpy);
376 }
377
378 void node_index_insert(list_t *l, int i, node_t *a) {
379     index_insert(l, i, a, NULL);
380 }
381
382 void graph_index_insert(list_t *l, int i, graph_t *a) {
383     index_insert(l, i, a, void_graph_copy);
384 }
385
386 float num_list_min(list_t *l) {
387     float min = (float) *(float *) l->data;
388     for(; l; l = l->next)
389         if(*(float *) l->data < min)
390             min = (float) *(float *) l->data;
391     return min;
392 }
393
394 float num_list_max(list_t *l) {
395     float max = (float) *(float *) l->data;
396     for(; l; l = l->next)
397         if(*(float *) l->data > max)
398             max = (float) *(float *) l->data;
399     return max;
400 }
401
402 char *string_list_min(list_t *l) {
403     char *min = (char *) l->data;
404     for(; l; l = l->next)
405         if(strcmp((char *)l->data, min) < 0)
406             min = (char *) l->data;
407     return min;
408 }
409
410 char *string_list_max(list_t *l) {
411     char *max = (char *) l->data;
412     for(; l; l = l->next)
413         if(strcmp((char *)l->data, max) > 0)
414             max = (char *) l->data;
415     return max;
416 }

```

```

417
418 int list_len(list_t *l) {
419     int len = 0;
420     for(; l; l = l->next)
421         len++;
422     return len;
423 }

```

Listing 29: list.c

## B.7 main.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "dict.h"
5
6 void print_nodes(graph_t *g) {
7     list_t *n;
8     for(n = g->nodes; n; n=n->next)
9         printf("%s\n", (char *) ((node_t *) n->data)->data);
10 }
11
12 int main() {
13     char *a = malloc(12*sizeof(char));
14     char *b = malloc(12*sizeof(char));
15     char *c = malloc(12*sizeof(char));
16     strncpy(a, "n: lo world", 12);
17     strncpy(b, "m: hello bb", 12);
18     strncpy(c, "o: fsf sefs", 12);
19
20     node_t *n = init_node((void *) a);
21     node_t *m = init_node((void *) b);
22     node_t *o = init_node((void *) c);
23     graph_t *g = NULL; // = init_graph();
24     graph_t *h = init_graph();
25     printf("%o\n", (int) a);
26     printf("%o\n", (int) b);
27     printf("%o\n", (int) c);
28
29     g = graph_plus_node(g, n);
30     add_node(g, m);
31     add_node(h, o);
32
33     graph_t *sum = node_plus_node(n, m);
34
35     printf("print_nodes(g):\n");
36     print_nodes(g);
37     printf("print_nodes(h):\n");
38     print_nodes(h);
39     printf("print_nodes(sum):\n");
40     print_nodes(sum);
41
42     connect_dir_weighted(m, n, 0.4);

```

```

43 connect_dir_weighted(n, m, 0.8);
44 printf("m --[%g]-- n\n", *(float *) get_node(m->out, n));
45 printf("n --[%g]-- m\n", *(float *) get_node(n->out, m));
46
47 /*
48     printf("%s -> %s\n", (char *)m->data, (char *)m->out->node->data);
49     printf("%s -> %s\n", (char *)n->data, (char *)n->out->node->data);
50 */
51
52 entry_t **d = init_dict();
53 put_graph(d, g, (void *) "this is graph g's data");
54 printf("here:\n");
55 put_graph(d, h, "this is graph h");
56
57 printf("%s\n", get_graph(d, g));
58 printf("%s\n", get_graph(d, h));
59
60 remove_undir_edge(m, n);
61 /*
62     printf("%s -> %x\n", (char *)m->data, (int)m->out);
63     printf("%s -> %x\n", (char *)n->data, (int)n->out);
64 */
65
66 plus_equals(g, h);
67 printf("print_nodes(g):\n");
68 print_nodes(g);
69 printf("print_nodes(h):\n");
70 print_nodes(h);
71 graph_t *g_copy = graph_copy(g);
72 int x = graph_equals(g, g_copy);
73 printf("x = %d\n", x);
74 x = graph_equals(h, g_copy);
75 printf("x = %d\n", x);
76 printf("print_nodes(copy):\n");
77 print_nodes(g_copy);
78 graph_t *diff;
79
80 diff = minus(g, h);
81 remove_node(g, o);
82 printf("print_nodes(g):\n");
83 print_nodes(g);
84
85 printf("done printing\n");
86
87 // free everything
88 free_graph(g);
89 printf("g freed\n");
90 free_graph(h);
91 printf("h freed\n");
92 free_node(m);
93 printf("m freed\n");
94 free_node(n);
95 printf("n freed\n");
96 free_node(o);

```

```

97     printf("o freed\n");
98
99     return 0;
100 }

```

Listing 30: main.c

## B.8 test.c

```

1 #include <stdio.h>
2 #include "dict.h"
3
4 int main() {
5     list_t *l = range(0, 10);
6     print_range(l);
7     free_range(l);
8     l = range(10, 0);
9     print_range(l);
10    //free_range(l);
11    float a = 30;
12    list_t *m = range(20, 30);
13    list_t *concat = num_list_concat(l, m);
14    printf("concat: ");
15    print_range(concat);
16    printf("concat.len = %d\n", list_len(concat));
17    float minf = num_list_min(concat);
18    float maxf = num_list_max(concat);
19    printf("min: %f\nmax: %f\n", minf, maxf);
20    free_range(l);
21    free_range(m);
22    l = range(20, 20);
23    print_range(l);
24    free_range(l);
25
26    char *s1 = "hello ";
27    char *s2 = "world";
28    l = NULL;
29    l = string_add_front(l, s2);
30    l = string_add_front(l, s1);
31    char *min = string_list_min(l);
32    char *max = string_list_max(l);
33    printf("min: %s\nmax: %s\n", min, max);
34    print_strings(l);
35    free_list(l);
36    printf("list done\n");
37
38    l = NULL;
39    l = string_add_back(l, s1);
40    l = string_add_back(l, s2);
41    list_t *string_copy = string_list_copy(l);
42    printf("l = ");
43    print_strings(l);
44    printf("string_copy = ");
45    print_strings(string_copy);

```

```

46 printf("l[0] = %s, l[1] = %s\n", list_access(l, 0), list_access(l, 1));
47
48
49 free_list(l);
50
51 entry_t **d = NULL; //init_dict();
52 d = put_string(d, "hello", (void *) "world");
53 put_string(d, "hello", (void *) "world2");
54 put_string(d, "elloh", (void *) "orldw");
55 put_string(d, "something else", (void *) "new value");
56 min = string_dict_min(d);
57 max = string_dict_max(d);
58 printf("min: %s: %s\nmax: %s: %s\n", min, (char *) get_string(d, min), max,
        get_string(d, max));
59 char *got = (char *) get_string(d, "hello");
60 printf("got %s\n", got);
61 got = (char *) get_string(d, "elloh");
62 printf("got %s\n", got);
63 got = (char *) get_string(d, "something else");
64 printf("got %s\n", got);
65 got = (char *) get_string(d, "world");
66 printf("got %s\n", got);
67 printf("d.len = %d\n", dict_len(d));
68
69 printf("1.23 == 1.24: %d\n", float_equals(1.23,1.24));
70
71 entry_t **nums = init_dict();
72 put_num(nums, 1.23, (void *) "1.23 val");
73 put_num(nums, 1.23, (void *) "1.23 second val");
74 put_num(nums, 1.24, (void *) "1.24 val");
75 printf("nums.len = %d\n", dict_len(nums));
76 minf = *(float *) num_dict_min(nums);
77 maxf = *(float *) num_dict_max(nums);
78 printf("min: %f\nmax: %f\n", minf, maxf);
79 got = (char *) get_num(nums, 1.23);
80 printf("got %s\n", got);
81 got = (char *) get_num(nums, 1.24);
82 printf("got %s\n", got);
83 printf("removed 1.23:\n");
84 num_dict_remove(nums, 1.23);
85 got = (char *) get_num(nums, 1.23);
86 printf("got %s\n", got);
87
88 node_t *n = init_node("this is a node!");
89 node_t *n2 = init_node("this is a node's value!");
90
91 entry_t **other = init_dict();
92 put_other(other, n, n2);
93 node_t *g = get_other(other, n);
94 int size = dict_len(other);
95 printf("got node containing %s, size = %d\n", g->data, size);
96 node_dict_remove(other, n);
97 size = dict_len(other);
98 g = get_other(other, n);

```

```

99     printf("got node containing %x\n, size = %d\n", (unsigned long) g, size);
100
101     entry_t **empty = init_dict();
102     int len = dict_len(empty);
103
104
105     return 0;
106 }

```

Listing 31: test.c

## B.9 snippets.c

This file was just a few notes on what C code was for a semantically equivalent line of d.o.t.s. code.

```

1  /* for node in graph */
2  list_t *temp;
3  node_t *node;
4  for(temp = graph->nodes; temp; temp = temp->next) {
5      node = temp->data;
6      /* loop body */
7  }
8
9  /* for node in list */
10 list_t *node;
11 for(node = list; node; node = node->next) {
12     /* loop body */
13 }
14
15 /* string s = arg */
16 char *s = arg;
17
18 /* num n = arg */
19 float n = arg;
20
21 /* node x */
22 node_t *x = init_node("");
23
24 /* node y(str) */
25 node_t *x = init_node(string);
26
27 /* graph g1; */
28 graph_t *g1 = init_graph();
29
30 /* graph g2 = g1; */
31 graph_t *g2 = graph_copy(g1);
32
33 /* graph g3 = {
34     x
35     y
36 };
37 */
38 graph_t *g3 = init_graph();
39 add_node(g3, x);

```

```

40 add_node(g3, y);
41
42 /* function declarations:
43    def return_type function_name(num arg1, node arg2) {
44        return return_type;
45    }
46    */
47
48 return_type function_name(float arg1, node_t *arg2) {
49     return return_type;
50 }
51
52 /* node1 == node 2 */
53 node_compare(node1, node2);
54
55 /* node1 != node2 */
56 !node_compare(node1, node2);
57
58 /* while statement */
59 while(statement) {
60     /* loop body */
61 }
62
63 /* if condition */
64 if(condition) {
65
66     /* else if */
67 } else if {
68
69     /* else */
70 } else {
71 }
72
73 /* x -- y */
74 connect_undir(x, y);
75
76 /* x --> y */
77 connect_dir(x, y);
78
79 /* x--[n] y */
80 connect_undir(x, y, n);
81
82 /* x-->[n] y */
83 connect_dir(x, y, n);
84
85 /* x [m]--[n] y */
86 connect_dir(x, y, n);
87 connect_dir(y, x, m);
88
89 /* g1 = g2 + g3 */
90 g1 = plus(g2, g3);
91
92 /* g1 += g2 */
93 plus_equals(g1, g2);

```



```

94
95 /* list<num> l = [1, 2, 3] */
96 list_t *l = NULL;
97 int *i;
98
99 i = (int *) malloc(sizeof(int));
100 *i = 1;
101 l = add_back(l, i);
102 i = (int *) malloc(sizeof(int));
103 *i = 2;
104 l = add_back(l, i);
105 i = (int *) malloc(sizeof(int));
106 *i = 3;
107 l = add_back(l, i);
108
109 /** dict initialization */
110 /* dict<type, type> d; */
111 entry_t **d = init_dict();
112
113 /** dict insertion */
114 /* d["literal"] = something */
115 put_string(d, "literal", (void *) &something);
116
117 /* d[1.23] = something */
118 put_num(d, 1.23, (void *) &something);
119
120 /* d[_node] = something */
121 put_node(d, (void *) &_node, (void *) &something);
122
123 /** dict access */
124 /* something = d["key"]; */
125 something = *(type *) get_string(d, "key");
126
127 /* something = d[1.23]; */
128 something = *(type *) get_num(d, 1.23);
129
130 /* something = d[_node]; */
131 something = *(type *) get_other(d, &_node);
132
133 /* for key in d */
134 // d[i] --> (*d)[i] OR d[0][i]
135 int i;
136 entry_t *temp;
137 void *key;
138 for(i = 0; i < TABLE_SIZE; i++) {
139     for(temp = d[i]; temp; temp = temp->next) {
140         key = temp->key;
141     }
142 }
143 }
144
145 /** printing dicts */
146 /** print d */
147 int i;

```

```

148 entry_t *temp;
149 void *key;
150 /* print "{"; */
151 int first = 1;
152 for(i = 0; i < TABLE_SIZE; i++) {
153     for(temp = d[i]; temp; temp = temp->next) {
154         key = temp->key;
155         if(first) {
156             first = 0;
157             /* print key, ": ", value */
158         } else {
159             /* print ", " , key, ": ", value */
160         }
161     }
162 }
163 /* print "}\n" */
164
165 /** printing lists */
166 /** print list_1; */
167 list_t *temp;
168 int first = 1;
169 /* print "[" */
170 for(temp = list_1; temp; temp = temp->next) {
171     if(first) {
172         first = 0;
173         /* print temp->data */
174     } else {
175         /* print ", ", temp->data */
176     }
177 }
178 /* print "]\n" */
179
180 /** adding strings */
181 /* s3 = s1 + s2; */
182 int len = strlen(s1) + strlen(s2) + 1;
183 char *s3 = (char *) calloc(len, sizeof(char));
184 strncpy(s3, s1, strlen(s1));
185 strncpy(s3, s2, strlen(s2));
186
187 /** removals */
188 /** dict - key1 */
189 int i;
190 entry_t *temp;
191 void *key;
192 for(i = 0; i < TABLE_SIZE; i++) {
193     for(temp = d[i]; temp; temp = temp->next) {
194         key = temp->key;
195         if(/*key == key1*/) {
196             dict_remove(d[i], temp);
197             i = TABLE_SIZE;
198             temp = NULL;
199         }
200     }
201 }

```

```

202
203 /* list.enqueue(data) */
204 <type>_add_back(list, data);
205
206 /* list.push(data) */
207 <type>_add_front(list, data);
208
209 /* list l2 = l1 */
210 list_t *l2 = <type>_list_copy(l1);
211
212 /* list l3 = l1 + l2 */
213 list_t *l3 = <type>_list_concat(l1, l2);
214
215 /* data = list.peek() */
216 <type> *data = (<type> *) peek(list);
217
218 /** ONLY removes first element from list and discards data */
219 /* list.pop() */
220 list = pop(list);
221
222 /* something = list[i] */
223 something = list_access(list, i);
224
225 /* g1 = g2 - g3 */
226 graph_t *g1 = minus(g2, g3);
227
228 /* list list_1 = list_2 + list_3 */
229 list_t *list_1 = <type>_list_concat(list_2, list_3);
230
231 /** graph = node + node */
232 /* graph g2 = n1 + n2 */
233 graph_t *g2 = node_plus_node(n1, n2);
234
235 /** graph = graph + node */
236 /* graph g2 = g1 + n2 */
237 graph_t *g2 = graph_plus_node(g1, n2);
238
239 /** dict.remove(key) */
240 <type>_dict_remove(dict, key);
241
242 /* dict.min() */
243 <type>_dict_min(dict);
244
245 /* list.max() */
246 <type>_list_max(list);
247
248 /** list[i] = something */
249 <type>_index_insert(list, i, something);

```

Listing 32: snippets.c

## C Makefiles

### C.1 src/Makefile

```
1 OBJS = ast.cmo Sast.cmo parser.cmo scanner.cmo translate.cmo analyzer.cmo
   typeConverter.cmo compile.cmo
2
3 TESTS = \
4 arith1 \
5 arith2 \
6 fib \
7 for1 \
8 func1 \
9 func2 \
10 func3 \
11 gcd \
12 gcd2 \
13 global1 \
14 hello \
15 if1 \
16 if2 \
17 if3 \
18 if4 \
19 ops1 \
20 var1 \
21 while1
22
23 # Choose one
24 YACC = ocaml yacc -v
25 # YACC = menhir --explain
26
27 TARFILES = Makefile testall.sh scanner.mll parser.mly \
28 ast.ml Sast.ml bytecode.ml interpret.ml compile.ml execute.ml microc.ml \
29 $(TESTS:%=tests/test-%.mc) \
30 $(TESTS:%=tests/test-%.out)
31
32
33 build: analyzer clib
34
35 .PHONY: setup
36 setup:
37 cd clib ; make clean
38 cd clib ; make library
39
40 clib:
41 make -f clib/Makefile
42
43 analyzer : str $(OBJS)
44 ocamlc -o dotc str.cma $(OBJS)
45
46 ast_print : str $(OBJS)
47 ocamlc -o dotc str.cma $(OBJS) astPrinter.ml
48
```

```

49 str :
50   ocamlpt str.cmxa
51
52 microc : $(OBJS)
53   ocamlc -o microc $(OBJS)
54
55 .PHONY : test
56 test : microc testall.sh
57   ./testall.sh
58
59 scanner.ml : scanner.mll
60   ocamllex scanner.mll
61
62 parser.ml parser.mli : parser.mly
63   $(YACC) parser.mly
64
65 %.cmo : %.ml
66   ocamlc -c $<
67
68 %.cmi : %.mli
69   ocamlc -c $<
70
71 microc.tar.gz : $(TARFILES)
72   cd .. && tar czf microc/microc.tar.gz $(TARFILES:%=microc/%)
73
74 .PHONY : clean
75 clean :
76   rm -f dotc microc parser.output parser.automaton parser.ml parser.mli
77       scanner.ml testall.log \
78       *.cmo *.cmi *.out *.diff exec compile.c
79 # Generated by ocamldep *.ml *.mli
80
81 analyzer.cmo: sast.cmo ast.cmo
82 analyzer.cmx: sast.cmx ast.cmx
83 generator.cmo: sast.cmo
84 generator.cmx: sast.cmx
85 parser.cmo: ast.cmo parser.cmi
86 parser.cmx: ast.cm parser.cmi
87 dot.cmo: scanner.cmo sast.cmo parser.cmi ast.cmo analyzer.cmo
88 dot.cmx: scanner.cmx sast.cmo parser.cmx ast.cmx analyzer.cmx
89 sast.cmo: ast.cmo
90 sast.cmx: ast.cmx
91 scanner.cmo: parser.cmi
92 scanner.cmx: parser.cmx
93 parser.cmi: ast.cmo
94 str.cma: str.cmxa

```

Listing 33: src/Makefile

## C.2 src/clib/Makefile

```

1 CC = gcc

```

```

2

```

```

3 CFLAGS = -Wall
4
5 CFILES= graph.c node.c list.c dict.c
6
7 main: main.o graph.o node.o dict.o
8
9 main.o: main.c graph.h node.h
10
11 graph.o: graph.h node.h graph.c
12
13 node.o: node.h node.c dict.o
14
15 test: test.o list.o dict.o graph.o node.o
16
17 list.o: list.c list.h
18
19 test.o: list.h test.c
20
21 dict.o: dict.c dict.h
22
23 .PHONY: objects
24 objects:
25     $(CC) -c $(CFILES)
26
27 .PHONY: library
28 library: objects
29     ar -cvq libdots.a *.o
30
31 .PHONY: clean
32 clean:
33     rm -f *.o main test *.a

```

Listing 34: src/clib/Makefile

## D Test Suite

### D.1 runtest.py

This was the main test suite script. It tested the whole compilation process.

```

1 # Test automation script
2
3 import os, sys, glob
4 import argparse
5 from subprocess32 import check_output, Popen, PIPE, call
6
7 #####
8 # ARGUMENT PARSING: #
9 #####
10
11 parser = argparse.ArgumentParser(description='Run tests on compilation of dots
12     files.')
12 parser.add_argument("-c", "--clean", action="store_true",

```

```

13     help="removes all created files after tests have finished")
14 args = parser.parse_args()
15
16 #####
17 # TESTS #
18 #####
19
20 path = r'dtest'
21 npath = r'ntests'
22
23 print "Using positive testing dir: " + path
24 print "Using negative testing dir: " + npath
25
26 summary_results = {}
27 summary_results_n = {}
28
29 for directory in os.walk(path):
30     # walk through the test directory
31     print ('\nRunning tests in "' + directory[0] + '" folder:')
32     print ('*****')
33     for dir_entry in os.listdir(directory[0]):
34         filepath = os.path.join(directory[0], dir_entry)
35         if os.path.isfile(filepath) and filepath[-5:] == '.dots':
36             print ('\nRunning tests: ' + dir_entry)
37             print ('=====')
38
39         comp_success = False
40         try:
41             return_code = call(['./gdc', filepath, os.path.join(directory[0],
42                 dir_entry[:-5] + '.exec')],
43                 timeout=15)
44             if return_code == 0:
45                 print 'COMPILATION SUCCESSFUL'
46                 comp_success = True
47             else:
48                 print 'COMPILATION FAILED'
49                 summary_results[dir_entry[:-5]] = ('fail', directory[0])
50         except:
51             print 'compile executable. Stop.'
52             continue;
53
54         if (comp_success):
55             out_child = Popen('./' + os.path.join(directory[0], dir_entry
56                 [:-5]) + '.exec',
57                 shell=True, stdout=PIPE, stderr=PIPE)
58             output = out_child.communicate()[0]
59
60             output_filepath = os.path.join(directory[0], dir_entry[:-5] + '.
61                 outgdc')
62             with open(output_filepath, 'w') as intermediate_output:
63                 intermediate_output.write(output)
64
65             out_filepath = os.path.join(directory[0], dir_entry[:-5] + '.out')

```

```

63     output_filepath = os.path.join(directory[0], dir_entry[:-5] + '.
        outgdc')
64
65     if (os.path.exists(out_filepath)):
66         diff_command = ['diff', '-bB', out_filepath, output_filepath]
67         diff_child = Popen(diff_command, stdout=PIPE)
68         diff_output = diff_child.communicate()[0]
69
70         if diff_output.strip() == '':
71             print 'PASSED TEST'
72             summary_results[dir_entry[:-5]] = ('pass', directory[0])
73         else:
74             print 'FAILED TEST....writing diff files'
75             summary_results[dir_entry[:-5]] = ('fail', directory[0])
76             with open(os.path.join(directory[0], dir_entry[:-5] + '.dif'
77                 ), 'w') as output_diff:
78                 output_diff.write(diff_output.strip())
79         else:
80             print "FAIL: no .out file exists to check against"
81
82 for directory in os.walk(npth):
83     print ('\nRunning tests in "' + directory[0] + '" folder:')
84     print ('*****')
85     for dir_entry in os.listdir(directory[0]):
86         filepath = os.path.join(directory[0], dir_entry)
87         if os.path.isfile(filepath) and filepath[-5:] == '.dots':
88             print('\nRunning tests: ' + dir_entry)
89             print('=====')
90
91     comp_success = False
92     try:
93         return_code = call(['./gdc', filepath, os.path.join(directory[0],
94             dir_entry[:-5] + '.exec')],
95             timeout=30)
96         if return_code == 0:
97             print 'COMPILATION SUCCESSFUL'
98             comp_success = True
99         else:
100            print 'COMPILATION FAILED'
101            summary_results_n[dir_entry[:-5]] = ('fail', directory[0])
102    except:
103        continue;
104
105    if (comp_success):
106        out_child = Popen('./' + os.path.join(directory[0], dir_entry
107           [:-5]) + '.exec',
108            shell=True, stdout=PIPE)
109        output = out_child.communicate()[0]
110
111        output_filepath = os.path.join(directory[0], dir_entry[:-5] + '.
        outgdc')
        with open(output_filepath, 'w') as intermediate_output:

```



```

112         intermediate_output.write(output)
113
114         out_filepath = os.path.join(directory[0], dir_entry[:-5] + '.out')
115         output_filepath = os.path.join(directory[0], dir_entry[:-5] + '.
            outgdc')
116
117         if (os.path.exists(out_filepath)):
118             diff_command = ['diff', '-bB', out_filepath, output_filepath]
119             diff_child = Popen(diff_command, stdout=PIPE)
120             diff_output = diff_child.communicate()[0]
121
122             if diff_output.strip() == '':
123                 print 'PASSED TEST'
124                 summary_results_n[dir_entry[:-5]] = ('pass', directory[0])
125             else:
126                 print 'FAILED TEST....writing diff files'
127                 summary_results_n[dir_entry[:-5]] = 'fail'
128                 with open(os.path.join(directory[0], dir_entry[:-5] + '.dif'
                    ), 'w') as output_diff:
129                     output_diff.write(diff_output.strip())
130         else:
131             print "FAIL: no .out file exists to check against"
132
133     print('\n Tests completed.')
134     print('\n Summary below (checked boxes = performed as expected): \n')
135
136     #####
137     # PRINT SUMMARY #
138     #####
139     print('Tests that should pass:')
140     for test_name in sorted(summary_results):
141         if summary_results[test_name][0] == 'pass':
142             print('[X] ' + test_name + '(' + summary_results[test_name][1] + ')')
143         else:
144             print('[ ] ' + test_name + '(' + summary_results[test_name][1] + ')')
145
146
147     print('\nTests that should fail:')
148     for test_name in sorted(summary_results_n):
149         if summary_results_n[test_name][0] == 'pass':
150             print('[ ] ' + test_name + '(' + summary_results_n[test_name][1] + ')')
151         else:
152             print('[X] ' + test_name + '(' + summary_results_n[test_name][1] + ')')
153
154     #####
155     # CLEAN-UP #
156     #####
157
158     # remove all the intermediate file output if the clean flag is set
159     if args.clean:
160         file_exts = ['*.outgdc', '*.dif', '*.c', '*.exec']
161         for ext in file_exts:
162             for directory in os.walk(path):
163                 for f in glob.glob(os.path.join(directory[0], ext)):

```

```

164         print(directory[0])
165         os.remove(f)
166     for directory in os.walk(npath):
167         for f in glob.glob(os.path.join(directory[0], ext)):
168             os.remove(f)

```

Listing 35: runtest.py

## D.2 testall.py

This was an alternate test script for testing using menhir.

```

1 # Menhir test automating script
2 # Might use it to automate more tests
3 # Make sure all of the input files are .txt files!
4
5 import os, sys, subprocess
6 run_normal = True
7 suppress_stderr = True
8
9 if len(sys.argv) == 1:
10     print('Running with default configurations:\n')
11 elif '-f' in sys.argv and '-e' in sys.argv:
12     run_normal = False
13     suppress_stderr = False
14 elif '-f' in sys.argv:
15     run_normal = False
16 elif '-e' in sys.argv:
17     suppress_stderr = False
18 else:
19     print('usage: -f, or no command line args')
20     print('-f: prints full results of every test')
21     print('-e: show stderr of menhir')
22     print('no command line args: runs all tests and only prints tests that
23         failed, suppresses stderr of menhir.\n')
24     sys.exit()
25
26 path = r'menhir-tests'
27 for dir_entry in os.listdir(path):
28     filepath = os.path.join(path, dir_entry)
29     if os.path.isfile(filepath) and filepath[-3:] == 'txt':
30         print('Running tests in ' + dir_entry)
31         print('=====\n')
32         with open(filepath, 'r') as test_file:
33             for line in test_file:
34                 supposed_to_pass = True
35                 if line[:3] == '***' or line[0] == '\n':
36                     continue
37
38                 to_pipe = line.strip().split()
39                 if to_pipe[0] == 'f**':
40                     supposed_to_pass = False
41                     to_pipe = to_pipe[1:]

```

```

41
42     to_pipe.insert(0, 'echo')
43
44     #can pipe with subprocess only by opening another process, not
45     #standard syntax
46     menhir_input = subprocess.Popen(to_pipe, stdout=subprocess.PIPE)
47     menhir_cmd = ['menhir', '--interpret', '--interpret-show-cst', '
48     parser.mly']
49     if suppress_stderr == True:
50         with open(os.devnull, 'w') as devnull:
51             output = subprocess.check_output(menhir_cmd, stdin=
52             menhir_input.stdout, stderr=devnull)
53     else:
54         output = subprocess.check_output(menhir_cmd, stdin=menhir_input
55         .stdout)
56
57     if run_normal == False:
58         print(output)
59     else:
60         if supposed_to_pass == True:
61             if 'REJECT' in output:
62                 print('"' + line.strip() + '" failed when it should pass
63                 .\n')
64             else:
65                 if 'ACCEPT' in output:
66                     print('"' + line.strip() + '" passed when it should fail
67                     .\n')
68
69     else:
70         print (dir_entry + ' is messed up.\n')
71         print ('-----\n\n')
72
73 print('\n Tests completed.')

```

Listing 36: testall.py

## E Example Code Compiled C Programs

### E.1 hello-world.dots.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <dict.h>
5 node_t * v1;
6 char * v2;
7 int main (int argc, char ** argv)
8 {
9     char ** v3;
10    v3 = malloc(sizeof(char *));
11    *(v3) = malloc(strlen("miami") + 1);
12    strcpy(*(v3), "miami");

```

```

13 v1 = init_node("");
14 (v1)->data = *(v3);
15 char ** v4;
16 v4 = malloc(sizeof(char *));
17 *(v4) = malloc(strlen("Hello World!") + 1);
18 strcpy(*(v4), "Hello World!");
19 char ** v5;
20 v5 = &(v2);
21 *(v5) = *(v4);
22 char ** v6;
23 v6 = &(v2);
24 printf("%s", *(v6));
25 char ** v7;
26 v7 = malloc(sizeof(char *));
27 *(v7) = malloc(strlen("\n") + 1);
28 strcpy(*(v7), "\n");
29 printf("%s", *(v7));
30 node_t ** v8;
31 v8 = &(v1);
32 printf("%s", "N-");
33 printf("%d", (int) (*(v8)));
34 printf("%s", "(\"");
35 printf("%s", (char *) ((*(v8))->data));
36 printf("\")");
37 char ** v9;
38 v9 = malloc(sizeof(char *));
39 *(v9) = malloc(strlen("\n") + 1);
40 strcpy(*(v9), "\n");
41 printf("%s", *(v9));
42 }

```

Listing 37: hello-world.dots.c

## E.2 sample01.dots.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <dict.h>
5 node_t * v1;
6 node_t * v2;
7 node_t * v3;
8 node_t * v4;
9 list_t * v5 = NULL;
10 char * v6;
11 graph_t * v7 = NULL;
12 graph_t * v8 = NULL;
13 list_t * v9 = NULL;
14 char * f6 (list_t * v89, node_t * v90)
15 {
16 list_t ** v91;
17 v91 = &(v89);
18 node_t * v93;
19 list_t * v92 = NULL;

```

```

20 for (v92 = *(v91); v92; v92 = (v92)->next) {
21 v93 = *((node_t **)((v92)->data));
22 node_t ** v94;
23 v94 = &(v93);
24 node_t ** v95;
25 v95 = &(v90);
26 if (*(v94) == *(v95)) {
27 char ** v96;
28 v96 = malloc(sizeof(char *));
29 *(v96) = malloc(strlen("yes") + 1);
30 strcpy(*(v96), "yes");
31 return *(v96);
32 } else {
33
34 }
35 }
36 char ** v97;
37 v97 = malloc(sizeof(char *));
38 *(v97) = malloc(strlen("no") + 1);
39 strcpy(*(v97), "no");
40 return *(v97);
41 }
42
43 char * f7 (graph_t * v89, node_t * v90)
44 {
45 graph_t ** v91;
46 v91 = &(v89);
47 node_t * v93;
48 list_t * v92 = NULL;
49 for (v92 = (*v91)->nodes; v92; v92 = (v92)->next) {
50 v93 = (node_t *)((v92)->data);
51 node_t ** v94;
52 v94 = &(v93);
53 node_t ** v95;
54 v95 = &(v90);
55 if (*(v94) == *(v95)) {
56 char ** v96;
57 v96 = malloc(sizeof(char *));
58 *(v96) = malloc(strlen("yes") + 1);
59 strcpy(*(v96), "yes");
60 return *(v96);
61 } else {
62
63 }
64 }
65 char ** v97;
66 v97 = malloc(sizeof(char *));
67 *(v97) = malloc(strlen("no") + 1);
68 strcpy(*(v97), "no");
69 return *(v97);
70 }
71
72 list_t * f8 (graph_t * v89, graph_t * v90)
73 {

```

```

74 list_t * v91 = NULL;
75 graph_t ** v92;
76 v92 = &(v89);
77 node_t * v94;
78 list_t * v93 = NULL;
79 for (v93 = (*v92)->nodes; v93; v93 = (v93)->next) {
80 v94 = (node_t *) ((v93)->data);
81 graph_t ** v95;
82 v95 = &(v90);
83 node_t * v97;
84 list_t * v96 = NULL;
85 for (v96 = (*v95)->nodes; v96; v96 = (v96)->next) {
86 v97 = (node_t *) ((v96)->data);
87 node_t ** v98;
88 v98 = &(v94);
89 node_t ** v99;
90 v99 = &(v97);
91 if (*(v98) == *(v99)) {
92 list_t ** v100;
93 v100 = &(v91);
94 node_t ** v101;
95 v101 = &(v94);
96 list_t ** v102;
97 v102 = malloc(sizeof(void));
98 *(v100) = node_add_back(*(v100), *(v101));;
99 } else {
100
101 }
102 }
103 }
104 list_t ** v103;
105 v103 = &(v91);
106 return *(v103);
107 }
108
109 int main (int argc, char ** argv)
110 {
111 char ** v10;
112 v10 = malloc(sizeof(char *));
113 *(v10) = malloc(strlen("chicago") + 1);
114 strcpy(*(v10), "chicago");
115 v1 = init_node("");
116 (v1)->data = *(v10);
117 char ** v11;
118 v11 = malloc(sizeof(char *));
119 *(v11) = malloc(strlen("bar") + 1);
120 strcpy(*(v11), "bar");
121 v2 = init_node("");
122 (v2)->data = *(v11);
123 char ** v12;
124 v12 = malloc(sizeof(char *));
125 *(v12) = malloc(strlen("foo") + 1);
126 strcpy(*(v12), "foo");
127 v3 = init_node("");

```

```

128 (v3)->data = *(v12);
129 char ** v13;
130 v13 = malloc(sizeof(char *));
131 *(v13) = malloc(strlen("blah") + 1);
132 strcpy(*(v13), "blah");
133 v4 = init_node("");
134 (v4)->data = *(v13);
135 list_t ** v14;
136 v14 = malloc(sizeof(list_t *));
137 *(v14) = NULL;
138 list_t ** v18;
139 node_t ** v15;
140 v15 = &(v1);
141 *(v14) = node_add_back(*(v14), v15);
142 node_t ** v16;
143 v16 = &(v2);
144 *(v14) = node_add_back(*(v14), v16);
145 node_t ** v17;
146 v17 = &(v3);
147 *(v14) = node_add_back(*(v14), v17);
148 v18 = v14;
149 list_t ** v19;
150 v19 = &(v5);
151 *(v19) = *(v18);
152 char ** v20;
153 v20 = malloc(sizeof(char *));
154 *(v20) = malloc(strlen("list contains: \n") + 1);
155 strcpy(*(v20), "list contains: \n");
156 printf("%s", *(v20));
157 printf("[");
158 list_t ** v22;
159 v22 = &(v5);
160 node_t * v24;
161 list_t * v23 = NULL;
162 for (v23 = *(v22); v23; v23 = (v23)->next) {
163 v24 = *((node_t **)((v23)->data));
164 node_t ** v25;
165 v25 = &(v24);
166 printf("%s", "N-");
167 printf("%d", (int)(*v25));
168 printf("%s", "(\"");
169 printf("%s", (char *)((*v25)->data));
170 printf("\")");
171 char ** v26;
172 v26 = malloc(sizeof(char *));
173 *(v26) = malloc(strlen(", ") + 1);
174 strcpy(*(v26), ", ");
175 printf("%s", *(v26));
176 }
177 printf("]");
178 char ** v27;
179 v27 = malloc(sizeof(char *));
180 *(v27) = malloc(strlen("\n") + 1);
181 strcpy(*(v27), "\n");

```

```

182 printf("%s", *(v27));
183 char ** v28;
184 v28 = malloc(sizeof(char *));
185 *(v28) = malloc(strlen("\n") + 1);
186 strcpy(*(v28), "\n");
187 printf("%s", *(v28));;
188 list_t ** v29;
189 v29 = &(v5);
190 node_t ** v30;
191 v30 = &(v1);
192 char ** v31;
193 v31 = malloc(sizeof(char *));
194 *(v31) = f6(*(v29), *(v30));
195 char ** v32;
196 v32 = &(v6);
197 *(v32) = *(v31);
198 node_t ** v33;
199 v33 = &(v1);
200 printf("%s", "N-");
201 printf("%d", (int) (*(v33)));;
202 printf("%s", "(\"");
203 printf("%s", (char *) ((*(v33))->data));
204 printf("\")");
205 char ** v34;
206 v34 = malloc(sizeof(char *));
207 *(v34) = malloc(strlen(" in node_list?\n") + 1);
208 strcpy(*(v34), " in node_list?\n");
209 printf("%s", *(v34));
210 char ** v35;
211 v35 = malloc(sizeof(char *));
212 *(v35) = malloc(strlen("\t") + 1);
213 strcpy(*(v35), "\t");
214 printf("%s", *(v35));
215 char ** v36;
216 v36 = &(v6);
217 printf("%s", *(v36));
218 char ** v37;
219 v37 = malloc(sizeof(char *));
220 *(v37) = malloc(strlen("\n") + 1);
221 strcpy(*(v37), "\n");
222 printf("%s", *(v37));;
223 list_t ** v38;
224 v38 = &(v5);
225 node_t ** v39;
226 v39 = &(v4);
227 char ** v40;
228 v40 = malloc(sizeof(char *));
229 *(v40) = f6(*(v38), *(v39));
230 char ** v41;
231 v41 = &(v6);
232 *(v41) = *(v40);
233 node_t ** v42;
234 v42 = &(v4);
235 printf("%s", "N-");

```



```

236 printf("%d", (int)(*v42));
237 printf("%s", "(");
238 printf("%s", (char *)((*v42)->data));
239 printf("\n");
240 char ** v43;
241 v43 = malloc(sizeof(char *));
242 *(v43) = malloc(strlen(" in node_list?\n") + 1);
243 strcpy(*(v43), " in node_list?\n");
244 printf("%s", *(v43));
245 char ** v44;
246 v44 = malloc(sizeof(char *));
247 *(v44) = malloc(strlen("\t") + 1);
248 strcpy(*(v44), "\t");
249 printf("%s", *(v44));
250 char ** v45;
251 v45 = &(v6);
252 printf("%s", *(v45));
253 char ** v46;
254 v46 = malloc(sizeof(char *));
255 *(v46) = malloc(strlen("\n") + 1);
256 strcpy(*(v46), "\n");
257 printf("%s", *(v46));
258 char ** v47;
259 v47 = malloc(sizeof(char *));
260 *(v47) = malloc(strlen("\n\n") + 1);
261 strcpy(*(v47), "\n\n");
262 printf("%s", *(v47));
263 node_t ** v48;
264 v48 = &(v1);
265 node_t ** v49;
266 v49 = &(v4);
267 graph_t ** v50;
268 v50 = malloc(sizeof(graph_t *));
269 *(v50) = (node_plus_node(*(v48), *(v49)));
270 graph_t ** v51;
271 v51 = &(v7);
272 *(v51) = *(v50);
273 graph_t ** v52;
274 v52 = &(v7);
275 node_t ** v53;
276 v53 = &(v2);
277 graph_t ** v54;
278 v54 = malloc(sizeof(graph_t *));
279 *(v54) = (graph_plus_node(*(v52), *(v53)));
280 graph_t ** v55;
281 v55 = &(v7);
282 *(v55) = *(v54);
283 char ** v56;
284 v56 = malloc(sizeof(char *));
285 *(v56) = malloc(strlen("G1 contains:\n") + 1);
286 strcpy(*(v56), "G1 contains:\n");
287 printf("%s", *(v56));
288 graph_t ** v57;
289 v57 = &(v7);

```

```

290 node_t * v59;
291 list_t * v58 = NULL;
292 for (v58 = (*v57)->nodes; v58; v58 = (v58)->next) {
293 v59 = (node_t *) ((v58)->data);
294 node_t ** v60;
295 v60 = &(v59);
296 printf("%s", "N-");
297 printf("%d", (int) (*v60));
298 printf("%s", "(\\");
299 printf("%s", (char *) ((*v60)->data));
300 printf("\\");
301 char ** v61;
302 v61 = malloc(sizeof(char *));
303 *(v61) = malloc(strlen("\\n") + 1);
304 strcpy(*(v61), "\\n");
305 printf("%s", *(v61));
306 }
307 char ** v62;
308 v62 = malloc(sizeof(char *));
309 *(v62) = malloc(strlen("\\n") + 1);
310 strcpy(*(v62), "\\n");
311 printf("%s", *(v62));
312 v8 = init_graph();
313 float* v63;
314 v63 = malloc(sizeof(float));
315 *(v63) = 22.3;
316 connect_dir_weighted (v3, v1, *(v63));
317 connect_dir(v2, v3);
318 connect_undir(v1, v2);
319 add_node(v8, v1);
320 add_node(v8, v2);
321 add_node(v8, v2);
322 add_node(v8, v3);
323 add_node(v8, v3);
324 add_node(v8, v1);
325 char ** v64;
326 v64 = malloc(sizeof(char *));
327 *(v64) = malloc(strlen("G2 contains: \\n") + 1);
328 strcpy(*(v64), "G2 contains: \\n");
329 printf("%s", *(v64));
330 graph_t ** v65;
331 v65 = &(v8);
332 node_t * v67;
333 list_t * v66 = NULL;
334 for (v66 = (*v65)->nodes; v66; v66 = (v66)->next) {
335 v67 = (node_t *) ((v66)->data);
336 node_t ** v68;
337 v68 = &(v67);
338 printf("%s", "N-");
339 printf("%d", (int) (*v68));
340 printf("%s", "(\\");
341 printf("%s", (char *) ((*v68)->data));
342 printf("\\");
343 char ** v69;

```

```

344 v69 = malloc(sizeof(char *));
345 *(v69) = malloc(strlen("\n") + 1);
346 strcpy(*(v69), "\n");
347 printf("%s", *(v69));;
348 }
349 char ** v70;
350 v70 = malloc(sizeof(char *));
351 *(v70) = malloc(strlen("\n") + 1);
352 strcpy(*(v70), "\n");
353 printf("%s", *(v70));;
354 graph_t ** v71;
355 v71 = &(v7);
356 node_t ** v72;
357 v72 = &(v3);
358 char ** v73;
359 v73 = malloc(sizeof(char *));
360 *(v73) = f7(*(v71), *(v72));
361 char ** v74;
362 v74 = &(v6);
363 *(v74) = *(v73);
364 char ** v75;
365 v75 = malloc(sizeof(char *));
366 *(v75) = malloc(strlen("z in g1? ") + 1);
367 strcpy(*(v75), "z in g1? ");
368 printf("%s", *(v75));
369 char ** v76;
370 v76 = &(v6);
371 printf("%s", *(v76));
372 char ** v77;
373 v77 = malloc(sizeof(char *));
374 *(v77) = malloc(strlen("\n") + 1);
375 strcpy(*(v77), "\n");
376 printf("%s", *(v77));;
377 graph_t ** v78;
378 v78 = &(v8);
379 node_t ** v79;
380 v79 = &(v3);
381 char ** v80;
382 v80 = malloc(sizeof(char *));
383 *(v80) = f7(*(v78), *(v79));
384 char ** v81;
385 v81 = &(v6);
386 *(v81) = *(v80);
387 char ** v82;
388 v82 = malloc(sizeof(char *));
389 *(v82) = malloc(strlen("z in g2? ") + 1);
390 strcpy(*(v82), "z in g2? ");
391 printf("%s", *(v82));
392 char ** v83;
393 v83 = &(v6);
394 printf("%s", *(v83));
395 char ** v84;
396 v84 = malloc(sizeof(char *));
397 *(v84) = malloc(strlen("\n") + 1);

```

```

398 strcpy(*(v84), "\n");
399 printf("%s", *(v84));;
400 graph_t ** v85;
401 v85 = &(v7);
402 graph_t ** v86;
403 v86 = &(v8);
404 list_t ** v87;
405 v87 = malloc(sizeof(list_t *));
406 *(v87) = f8(*(v85), *(v86));
407 list_t ** v88;
408 v88 = &(v9);
409 *(v88) = *(v87);
410 char ** v89;
411 v89 = malloc(sizeof(char *));
412 *(v89) = malloc(strlen("\nSHARED NODES:\n") + 1);
413 strcpy(*(v89), "\nSHARED NODES:\n");
414 printf("%s", *(v89));;
415 }

```

Listing 38: sample01.dots.c

### E.3 bst.dots.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <dict.h>
5 graph_t * v1 = NULL;
6 node_t * v2;
7 node_t * v3;
8 node_t * v4;
9 node_t * v5;
10 node_t * v6;
11 node_t * v7;
12 list_t * v8 = NULL;
13 list_t * v9 = NULL;
14 entry_t** v10 = NULL;
15 node_t * v11;
16 float v12;
17 int f6 (list_t * v111, node_t * v112)
18 {
19 list_t ** v113;
20 v113 = &(v111);
21 node_t * v115;
22 list_t * v114 = NULL;
23 for (v114 = *(v113); v114; v114 = (v114)->next) {
24 v115 = *((node_t **)((v114)->data));
25 node_t ** v116;
26 v116 = &(v115);
27 node_t ** v117;
28 v117 = &(v112);
29 if (*(v116) == *(v117)) {
30 int* v118;
31 v118 = malloc(sizeof(int));

```

```

32 *(v118) = 1;
33 return *(v118);
34 } else {
35
36 }
37 }
38 int* v119;
39 v119 = malloc(sizeof(int));
40 *(v119) = 0;
41 return *(v119);
42 }
43
44 int main (int argc, char ** argv)
45 {
46 char ** v13;
47 v13 = malloc(sizeof(char *));
48 *(v13) = malloc(strlen("Searching\n") + 1);
49 strcpy(*(v13), "Searching\n");
50 printf("%s", *(v13));;
51 char ** v14;
52 v14 = malloc(sizeof(char *));
53 *(v14) = malloc(strlen("x") + 1);
54 strcpy(*(v14), "x");
55 v2 = init_node("");
56 (v2)->data = *(v14);
57 char ** v15;
58 v15 = malloc(sizeof(char *));
59 *(v15) = malloc(strlen("y") + 1);
60 strcpy(*(v15), "y");
61 v3 = init_node("");
62 (v3)->data = *(v15);
63 char ** v16;
64 v16 = malloc(sizeof(char *));
65 *(v16) = malloc(strlen("z") + 1);
66 strcpy(*(v16), "z");
67 v4 = init_node("");
68 (v4)->data = *(v16);
69 char ** v17;
70 v17 = malloc(sizeof(char *));
71 *(v17) = malloc(strlen("a") + 1);
72 strcpy(*(v17), "a");
73 v5 = init_node("");
74 (v5)->data = *(v17);
75 char ** v18;
76 v18 = malloc(sizeof(char *));
77 *(v18) = malloc(strlen("b") + 1);
78 strcpy(*(v18), "b");
79 v6 = init_node("");
80 (v6)->data = *(v18);
81 char ** v19;
82 v19 = malloc(sizeof(char *));
83 *(v19) = malloc(strlen("c") + 1);
84 strcpy(*(v19), "c");
85 v7 = init_node("");

```

```

86 (v7)->data = *(v19);
87 node_t ** v20;
88 v20 = &(v2);
89 node_t ** v21;
90 v21 = &(v3);
91 graph_t ** v22;
92 v22 = malloc(sizeof(graph_t *));
93 *(v22) = (node_plus_node(*(v20), *(v21)));
94 graph_t ** v23;
95 v23 = &(v1);
96 *(v23) = *(v22);
97 graph_t ** v24;
98 v24 = &(v1);
99 node_t ** v25;
100 v25 = &(v4);
101 graph_t ** v26;
102 v26 = malloc(sizeof(graph_t *));
103 *(v26) = (graph_plus_node(*(v24), *(v25)));
104 graph_t ** v27;
105 v27 = &(v1);
106 *(v27) = *(v26);
107 float* v28;
108 v28 = malloc(sizeof(float));
109 *(v28) = 2;
110 connect_dir_weighted (v2, v3, *(v28));
111 float* v29;
112 v29 = malloc(sizeof(float));
113 *(v29) = 1.5;
114 connect_dir_weighted (v2, v4, *(v29));
115 float* v30;
116 v30 = malloc(sizeof(float));
117 *(v30) = 4;
118 connect_dir_weighted (v4, v3, *(v30));
119 float* v31;
120 v31 = malloc(sizeof(float));
121 *(v31) = 2;
122 connect_dir_weighted (v3, v7, *(v31));
123 float* v32;
124 v32 = malloc(sizeof(float));
125 *(v32) = 2.5;
126 connect_dir_weighted (v4, v6, *(v32));
127 float* v33;
128 v33 = malloc(sizeof(float));
129 *(v33) = .5;
130 connect_dir_weighted (v7, v6, *(v33));
131 float* v34;
132 v34 = malloc(sizeof(float));
133 *(v34) = 333;
134 connect_dir_weighted (v2, v5, *(v34));
135 float* v35;
136 v35 = malloc(sizeof(float));
137 *(v35) = 15;
138 connect_dir_weighted (v4, v5, *(v35));
139 char ** v36;

```

```

140 v36 = malloc(sizeof(char *));
141 *(v36) = malloc(strlen("Graph Initialized\n") + 1);
142 strcpy(*(v36), "Graph Initialized\n");
143 printf("%s", *(v36));;
144
145 v11 = init_node("");
146 (v11)->data = "";
147 node_t ** v37;
148 v37 = &(v2);
149 node_t ** v38;
150 v38 = &(v11);
151 *(v38) = *(v37);
152 float* v39;
153 v39 = malloc(sizeof(float));
154 *(v39) = 0;
155 float* v40;
156 v40 = &(v12);
157 *(v40) = *(v39);
158 node_t ** v41;
159 v41 = &(v11);
160 entry_t*** v42;
161 v42 = &((*(v41))->out);
162 int v69;
163 entry_t* v43;
164 void* v44;
165 if (*(v42)) {
166 for (v69 = 0; v69 < TABLE_SIZE; v69 = v69 + 1) {
167 for (v43 = (*(v42))[v69]; v43; v43 = (v43)->next) {
168 v44 = (v43)->key;
169 char ** v70;
170 v70 = malloc(sizeof(char *));
171 *(v70) = malloc(strlen("current node: ") + 1);
172 strcpy(*(v70), "current node: ");
173 printf("%s", *(v70));
174 node_t ** v71;
175 v71 = &(v44);
176 printf("%s", "N-");
177 printf("%d", (int) (*(v71)));
178 printf("%s", "(\\");
179 printf("%s", (char *) ((*(v71))->data));
180 printf("\\");
181 char ** v72;
182 v72 = malloc(sizeof(char *));
183 *(v72) = malloc(strlen("\\n") + 1);
184 strcpy(*(v72), "\\n");
185 printf("%s", *(v72));
186 char ** v73;
187 v73 = malloc(sizeof(char *));
188 *(v73) = malloc(strlen("\\n") + 1);
189 strcpy(*(v73), "\\n");
190 printf("%s", *(v73));;
191 float* v74;
192 v74 = &(v12);
193 float* v75;

```

```

194 v75 = malloc(sizeof(float));
195 *(v75) = 1;
196 float* v76;
197 v76 = malloc(sizeof(float));
198 *(v76) = (*(v74) + *(v75));
199 float* v77;
200 v77 = &(v12);
201 *(v77) = *(v76);
202
203 list_t ** v78;
204 v78 = &(v9);
205 node_t ** v79;
206 v79 = &(v2);
207 list_t ** v80;
208 v80 = malloc(sizeof(void));
209 *(v78) = node_add_back(*(v78), *(v79));;
210 list_t ** v81;
211 v81 = &(v8);
212 node_t ** v82;
213 v82 = &(v44);
214 list_t ** v83;
215 v83 = malloc(sizeof(void));
216 *(v81) = node_add_back(*(v81), *(v82));;
217 entry_t*** v84;
218 v84 = &(v10);
219 node_t ** v85;
220 v85 = &(v44);
221 float* v86;
222 v86 = &(v12);
223 node_t * v88;
224 v88 = *(v85);
225 float v87;
226 v87 = *(v86);
227 *(v84) = put_node(*(v84), (node_t *) (v88), (void*) (&(v87)));;
228 list_t ** v89;
229 v89 = &(v8);
230 node_t ** v90;
231 v90 = peek(*(v89));
232 node_t ** v91;
233 v91 = &(v11);
234 *(v91) = *(v90);
235 list_t ** v92;
236 v92 = &(v8);
237 void* v93;
238 *(v92) = pop(*(v92));;
239 }
240 }
241 } else {
242
243 }
244 printf("{");
245 entry_t*** v95;
246 v95 = &(v10);
247 int v104;

```



```

248 entry_t* v96;
249 void* v97;
250 if (*(v95)) {
251 for (v104 = 0; v104 < TABLE_SIZE; v104 = v104 + 1) {
252 for (v96 = (*(v95))[v104]; v96; v96 = (v96)->next) {
253 v97 = (v96)->key;
254 node_t ** v105;
255 v105 = &(v97);
256 printf("%s", "N-");
257 printf("%d", (int) (*(v105)));
258 printf("%s", "(\\");
259 printf("%s", (char *) ((*(v105))->data));
260 printf("\\");
261 char ** v106;
262 v106 = malloc(sizeof(char *));
263 *(v106) = malloc(strlen(": ") + 1);
264 strcpy(*(v106), ": ");
265 printf("%s", *(v106));
266 entry_t*** v107;
267 v107 = &(v10);
268 node_t ** v108;
269 v108 = &(v97);
270 float* v109;
271 v109 = (float*) (get_node(*(v107), *(v108)));
272 printf("%.3f", *(v109));
273 char ** v110;
274 v110 = malloc(sizeof(char *));
275 *(v110) = malloc(strlen(", ") + 1);
276 strcpy(*(v110), ", ");
277 printf("%s", *(v110));
278 }
279 }
280 } else {
281
282 }
283 printf("}");
284 char ** v111;
285 v111 = malloc(sizeof(char *));
286 *(v111) = malloc(strlen("\\n\\n") + 1);
287 strcpy(*(v111), "\\n\\n");
288 printf("%s", *(v111));
289 }

```

Listing 39: bst.dots.c

## E.4 tutorial.dots.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <dict.h>
5 float v1;
6 char * v2;
7 node_t * v3;

```

```

8 node_t * v4;
9 graph_t * v5 = NULL;
10 list_t * v6 = NULL;
11 entry_t** v7 = NULL;
12 int main (int argc, char ** argv)
13 {
14 float* v8;
15 v8 = malloc(sizeof(float));
16 *(v8) = 5.3;
17 float* v9;
18 v9 = &(v1);
19 *(v9) = *(v8);
20 char ** v10;
21 v10 = malloc(sizeof(char *));
22 *(v10) = malloc(strlen("This is a d.o.t.s. program.") + 1);
23 strcpy(*(v10), "This is a d.o.t.s. program.");
24 char ** v11;
25 v11 = &(v2);
26 *(v11) = *(v10);
27
28 v3 = init_node("");
29 (v3)->data = "";
30 char ** v12;
31 v12 = malloc(sizeof(char *));
32 *(v12) = malloc(strlen("florida") + 1);
33 strcpy(*(v12), "florida");
34 v4 = init_node("");
35 (v4)->data = *(v12);
36 node_t ** v13;
37 v13 = &(v3);
38 node_t ** v14;
39 v14 = &(v4);
40 graph_t ** v15;
41 v15 = malloc(sizeof(graph_t *));
42 *(v15) = (node_plus_node(*(v13), *(v14)));
43 graph_t ** v16;
44 v16 = &(v5);
45 *(v16) = *(v15);
46 float* v17;
47 v17 = malloc(sizeof(float));
48 *(v17) = 2.3;
49 connect_dir_weighted (v3, v4, *(v17));
50 connect_dir(v4, v3);
51 list_t ** v18;
52 v18 = malloc(sizeof(list_t *));
53 *(v18) = NULL;
54 list_t ** v21;
55 node_t ** v19;
56 v19 = &(v3);
57 *(v18) = node_add_back(*(v18), v19);
58 node_t ** v20;
59 v20 = &(v4);
60 *(v18) = node_add_back(*(v18), v20);
61 v21 = v18;

```

```

62 list_t ** v22;
63 v22 = &(v6);
64 *(v22) = *(v21);
65 entry_t*** v23;
66 v23 = &(v7);
67 char ** v24;
68 v24 = malloc(sizeof(char *));
69 *(v24) = malloc(strlen("e") + 1);
70 strcpy(*(v24), "e");
71 float* v25;
72 v25 = malloc(sizeof(float));
73 *(v25) = 12.7;
74 float v26;
75 v26 = *(v25);
76 *(v23) = put_string(*(v23), *(v24), (void*)&(v26));
77 entry_t*** v27;
78 v27 = &(v7);
79 char ** v28;
80 v28 = malloc(sizeof(char *));
81 *(v28) = malloc(strlen("a") + 1);
82 strcpy(*(v28), "a");
83 float* v29;
84 v29 = malloc(sizeof(float));
85 *(v29) = 8.17;
86 float v30;
87 v30 = *(v29);
88 *(v27) = put_string(*(v27), *(v28), (void*)&(v30));
89 entry_t*** v31;
90 v31 = &(v7);
91 char ** v32;
92 v32 = malloc(sizeof(char *));
93 *(v32) = malloc(strlen("d") + 1);
94 strcpy(*(v32), "d");
95 float* v33;
96 v33 = malloc(sizeof(float));
97 *(v33) = 4.25;
98 float v34;
99 v34 = *(v33);
100 *(v31) = put_string(*(v31), *(v32), (void*)&(v34));
101 float* v35;
102 v35 = &(v1);
103 printf("%.3f", *(v35));
104 char ** v36;
105 v36 = malloc(sizeof(char *));
106 *(v36) = malloc(strlen("\n") + 1);
107 strcpy(*(v36), "\n");
108 printf("%s", *(v36));
109 char ** v37;
110 v37 = &(v2);
111 printf("%s", *(v37));
112 char ** v38;
113 v38 = malloc(sizeof(char *));
114 *(v38) = malloc(strlen("\n") + 1);
115 strcpy(*(v38), "\n");

```

```

116 printf("%s", *(v38));
117 node_t ** v39;
118 v39 = &(v3);
119 printf("%s", "N-");
120 printf("%d", (int) (*(v39)));
121 printf("%s", "(\\");
122 printf("%s", (char *) ((*(v39))->data));
123 printf("\\");
124 char ** v40;
125 v40 = malloc(sizeof(char *));
126 *(v40) = malloc(strlen("\\n") + 1);
127 strcpy(*(v40), "\\n");
128 printf("%s", *(v40));
129 node_t ** v41;
130 v41 = &(v4);
131 printf("%s", "N-");
132 printf("%d", (int) (*(v41)));
133 printf("%s", "(\\");
134 printf("%s", (char *) ((*(v41))->data));
135 printf("\\");
136 char ** v42;
137 v42 = malloc(sizeof(char *));
138 *(v42) = malloc(strlen("\\n") + 1);
139 strcpy(*(v42), "\\n");
140 printf("%s", *(v42));;
141 graph_t ** v43;
142 v43 = &(v5);
143 node_t * v45;
144 list_t * v44 = NULL;
145 for (v44 = (*(v43))->nodes; v44; v44 = (v44)->next) {
146 v45 = (node_t *) ((v44)->data);
147 node_t ** v46;
148 v46 = &(v45);
149 printf("%s", "N-");
150 printf("%d", (int) (*(v46)));
151 printf("%s", "(\\");
152 printf("%s", (char *) ((*(v46))->data));
153 printf("\\");
154 char ** v47;
155 v47 = malloc(sizeof(char *));
156 *(v47) = malloc(strlen("\\n") + 1);
157 strcpy(*(v47), "\\n");
158 printf("%s", *(v47));;
159 }
160 printf("[");
161 list_t ** v49;
162 v49 = &(v6);
163 node_t * v51;
164 list_t * v50 = NULL;
165 for (v50 = *(v49); v50; v50 = (v50)->next) {
166 v51 = *((node_t **) ((v50)->data));
167 node_t ** v52;
168 v52 = &(v51);
169 printf("%s", "N-");

```

```

170 printf("%d", (int) (*(v52)));
171 printf("%s", "\\");
172 printf("%s", (char *) ((*(v52))->data));
173 printf("\\");
174 char ** v53;
175 v53 = malloc(sizeof(char *));
176 *(v53) = malloc(strlen(", ") + 1);
177 strcpy(*(v53), ", ");
178 printf("%s", *(v53));
179 }
180 printf("]");
181 char ** v54;
182 v54 = malloc(sizeof(char *));
183 *(v54) = malloc(strlen("\\n") + 1);
184 strcpy(*(v54), "\\n");
185 printf("%s", *(v54));
186 printf("{");
187 entry_t*** v56;
188 v56 = &(v7);
189 int v65;
190 entry_t* v57;
191 void* v58;
192 if (*(v56)) {
193 for (v65 = 0; v65 < TABLE_SIZE; v65 = v65 + 1) {
194 for (v57 = (*(v56))[v65]; v57; v57 = (v57)->next) {
195 v58 = (v57)->key;
196 char ** v66;
197 v66 = &(v58);
198 printf("%s", *(v66));
199 char ** v67;
200 v67 = malloc(sizeof(char *));
201 *(v67) = malloc(strlen(": ") + 1);
202 strcpy(*(v67), ": ");
203 printf("%s", *(v67));
204 entry_t*** v68;
205 v68 = &(v7);
206 char ** v69;
207 v69 = &(v58);
208 float* v70;
209 v70 = (float*) (get_string(*(v68), *(v69)));
210 printf("%.3f", *(v70));
211 char ** v71;
212 v71 = malloc(sizeof(char *));
213 *(v71) = malloc(strlen(", ") + 1);
214 strcpy(*(v71), ", ");
215 printf("%s", *(v71));
216 }
217 }
218 } else {
219
220 }
221 printf("}");
222 char ** v72;
223 v72 = malloc(sizeof(char *));

```

```

224 *(v72) = malloc(strlen("\n") + 1);
225 strcpy(*(v72), "\n");
226 printf("%s", *(v72));
227 }

```

Listing 40: tutorial.dots.c

## F Git Commit History

```

1 commit 9a32082c6edab4f17fcdc72c34350de433b91cc4
2 Author: rgordon <rgordon@umass.edu>
3 Date: Tue Dec 22 11:22:46 2015 -0500
4
5     cleaned up code
6
7 commit 5eb8a0527a529fbaea3cfcacbb0eefd7177ffa52
8 Author: rgordon <rgordon@umass.edu>
9 Date: Tue Dec 22 10:58:39 2015 -0500
10
11     got rid of unused rules by having fdecls use the alternate stmt rule that
12     doesn't include fdecl (no nested fdecls)
13
14 commit 4bflc9acced90a35bd1607b6a2638f45c4b6502c
15 Author: rgordon <rgordon@umass.edu>
16 Date: Tue Dec 22 10:50:00 2015 -0500
17
18     cleaned up runtest.py
19
20 commit 154e9dad2d9f7da88caad984a68e73ce678662e6
21 Author: rgordon <rgordon@umass.edu>
22 Date: Tue Dec 22 01:22:55 2015 -0500
23
24     removed commented out code. added clean to the setup reule in Makefile
25
26 commit 78869c4034340a6aeba668c523863bd58262897d
27 Merge: 12b7090 8bd68d1
28 Author: Adam Incera <aji2112@columbia.edu>
29 Date: Tue Dec 22 01:00:46 2015 -0500
30
31     merge
32
33 commit 12b7090680d61ae64d2c2941fc7d2e9e47ccf416
34 Author: Adam Incera <aji2112@columbia.edu>
35 Date: Tue Dec 22 00:59:58 2015 -0500
36
37     cleaning out src directory
38
39 commit f5c9d3d34302c5b9246c0a79419b2f6065484e04
40 Author: Adam Incera <aji2112@columbia.edu>
41 Date: Tue Dec 22 00:55:07 2015 -0500
42
43     cleaned up translator and analyzer
44
45 commit 8bd68d157e51c8113e1142142fd4bfba58f0bf4e

```

```
45 Author: rgordon <rcgordon@umass.edu>
46 Date: Tue Dec 22 00:23:34 2015 -0500
47
48     added some image files for our report. removed extraneous file from src
49
50 commit ff9b8a16d22bb9746428d21ed4671f588f8bbdf4
51 Author: rgordon <rcgordon@umass.edu>
52 Date: Mon Dec 21 19:58:05 2015 -0500
53
54     put copies of our example code in src/sample-code to make it easier for TAs
55     to find
56
57 commit aed9f26d63082bf69f78afac27cc36248110fbeb
58 Merge: 4b89c4b e4008a6
59 Author: rgordon <rcgordon@umass.edu>
60 Date: Mon Dec 21 19:53:46 2015 -0500
61
62     Merge branch 'compile'
63
64     merged current compile branch into master
65
66 commit e4008a6284d765a2b50e777e83e0e7721f9682f0
67 Author: rgordon <rcgordon@umass.edu>
68 Date: Mon Dec 21 19:53:20 2015 -0500
69
70     minor changes to make output prettier
71
72 commit 4b89c4be9e2fde92ab5dfdd4bd3ec5960090d6df
73 Author: Hosanna Fuller <Miramonte23@gmail.com>
74 Date: Mon Dec 21 09:12:02 2015 -0500
75
76     text
77
78 commit 31315eed5b12031f7676c19a00d8d4a53bc566c3
79 Author: Hosanna Fuller <Miramonte23@gmail.com>
80 Date: Mon Dec 21 09:11:17 2015 -0500
81
82     format
83
84 commit 5cb160fe68a92a85e81f240effe6a13893b960b1
85 Author: Hosanna Fuller <Miramonte23@gmail.com>
86 Date: Mon Dec 21 08:56:00 2015 -0500
87
88     syntax highlighting
89
90 commit 8c373b266992ceddf06e472d22f5f6f82f65d00a
91 Author: Hosanna Fuller <Miramonte23@gmail.com>
92 Date: Mon Dec 21 08:53:34 2015 -0500
93
94     syntax high lighting
95
96 commit 19f843182137359f2c74f881e32d22a90afaa0f8
97 Author: Hosanna Fuller <Miramonte23@gmail.com>
98 Date: Mon Dec 21 08:49:14 2015 -0500
```

```
98
99   hosanna
100
101 commit 2d91001030f8c7894832725679f337621fafbc57
102 Author: Hosanna Fuller <Miramonte23@gmail.com>
103 Date: Mon Dec 21 08:48:04 2015 -0500
104
105   format changes
106
107 commit 170dfcf384ec1d664e3a5b0c92adf224d5e3f290
108 Author: Hosanna Fuller <Miramonte23@gmail.com>
109 Date: Mon Dec 21 08:47:06 2015 -0500
110
111   compilation in structions
112
113 commit f04a58b268747796d6cee19e412c736ac2035da1
114 Author: Hosanna Fuller <Miramonte23@gmail.com>
115 Date: Mon Dec 21 08:41:39 2015 -0500
116
117   duplicate readme
118
119 commit e3d9cc6f6ee456521bb80de8f199745359d42ae8
120 Author: Hosanna Fuller <Miramonte23@gmail.com>
121 Date: Mon Dec 21 08:41:13 2015 -0500
122
123   update to readme.md
124
125 commit 0324e8834ffc893f5a4ce97c22b3aade004d68fe
126 Merge: 4ab2577 10d6dd8
127 Author: rgordon <rcgordon@umass.edu>
128 Date: Mon Dec 21 06:06:50 2015 -0500
129
130   mergeMerge branch 'compile' of https://github.com/adamincera/dots into
131   compile
132
133 commit 4ab2577fd7105c852adbfe57aa9b6bc136d6f95e
134 Author: rgordon <rcgordon@umass.edu>
135 Date: Mon Dec 21 06:06:48 2015 -0500
136
137   sample files
138
139 commit lcccafbb5739cf8999458ac8cad837461cf664cf
140 Author: rgordon <rcgordon@umass.edu>
141 Date: Mon Dec 21 06:04:46 2015 -0500
142
143   sample algors
144
145 commit 10d6dd880f532a8845a08a2b77a82625ff6e3201
146 Author: hosannajfull <miramonte23@gmail.com>
147 Date: Mon Dec 21 05:43:49 2015 -0500
148
149   working bst
150
151 commit 6b9ce3d237c0a804e63dc665a8eac2becb9391cc
```



```
151 Author: hosannajfull <miramonte23@gmail.com>
152 Date: Mon Dec 21 05:13:44 2015 -0500
153
154     pushing
155
156 commit 95012f41c8dd6fbcd6009a0238b5483721c9522e
157 Author: hosannajfull <miramonte23@gmail.com>
158 Date: Mon Dec 21 03:51:19 2015 -0500
159
160     fixes for the segfault on values
161
162 commit 6a5bd4d61d1bb78c23ecf25030e5bd571f4fad98
163 Merge: 72ea472 51954bb
164 Author: hosannajfull <miramonte23@gmail.com>
165 Date: Mon Dec 21 02:36:10 2015 -0500
166
167     Merge branch 'compile' of github.com:adamincera/dots into compile
168
169 commit 72ea47227eacf44d9d89d0e52ba255a35ce7ba4c
170 Author: hosannajfull <miramonte23@gmail.com>
171 Date: Mon Dec 21 02:36:00 2015 -0500
172
173     temp
174
175 commit 51954bb6e94fff2185f5df86ce8dd8401bb48c4a
176 Author: Adam Incera <aji2112@columbia.edu>
177 Date: Mon Dec 21 02:35:25 2015 -0500
178
179     took out free
180
181 commit 28ed5d5ba4cd10c20dd8054e86b80e04c41f9f56
182 Merge: 9e7d310 31c2276
183 Author: Adam Incera <aji2112@columbia.edu>
184 Date: Mon Dec 21 02:06:17 2015 -0500
185
186     merge
187
188 commit 9e7d3101cc7f6df5a13d4be36b27e1587aad460c
189 Author: Adam Incera <aji2112@columbia.edu>
190 Date: Mon Dec 21 02:06:08 2015 -0500
191
192     graph + node
193
194 commit 31c2276f761cc4fd9afeeabc8c9a8457b7a797d7
195 Author: rgordon <rcgordon@umass.edu>
196 Date: Mon Dec 21 01:53:09 2015 -0500
197
198     adds
199
200 commit fdcbc66caa0a13dcd68b19afb77707b2be9171e0
201 Merge: eea7a53 791c1c4
202 Author: hosannajfull <miramonte23@gmail.com>
203 Date: Mon Dec 21 01:11:21 2015 -0500
204
```

```
205 Merge branch 'compile' of github.com:adamincera/dots into compile
206
207 merge
208
209 commit eea7a53717a4ad32398b707b43cd9d7afc756f4b
210 Author: hosannajfull <miramonte23@gmail.com>
211 Date: Mon Dec 21 01:11:04 2015 -0500
212
213 fixed bfs error. altered if stmt handling in analyzer.ml
214
215 commit 791c1c406a5b771d9f1bf8c4e97467baf359d572
216 Merge: 5b462e1 67a35d2
217 Author: Adam Incera <aji2112@columbia.edu>
218 Date: Mon Dec 21 01:03:38 2015 -0500
219
220 merge
221
222 commit 5b462e119334f4494063df1d9a865a0685046cb6
223 Author: Adam Incera <aji2112@columbia.edu>
224 Date: Mon Dec 21 01:03:30 2015 -0500
225
226 pulling
227
228 commit 67a35d2d96d5f08715a706bcd471fe8965697a2d
229 Merge: 88995a8 df268e8
230 Author: rgordon <rcgordon@umass.edu>
231 Date: Mon Dec 21 00:39:04 2015 -0500
232
233 Merge branch 'compile' of https://github.com/adamincera/dots into compile
234
235 merge
236
237 commit 88995a86b22cc34353fd99c3377496a7d3fb493d
238 Author: rgordon <rcgordon@umass.edu>
239 Date: Mon Dec 21 00:38:49 2015 -0500
240
241 alterations
242
243 commit df268e835a18e20963be66daa9b995875c0fa03e
244 Author: hosannajfull <miramonte23@gmail.com>
245 Date: Mon Dec 21 00:31:36 2015 -0500
246
247 rcg and hosanna added breadth first search example
248
249 commit 5c51b503c5d694af5d448e3e419e73d7f1b4f5dc
250 Author: hosannajfull <miramonte23@gmail.com>
251 Date: Mon Dec 21 00:29:59 2015 -0500
252
253 test cases
254
255 commit 4b53058ddc80d55cdfb9240ab0c8608b613ed813
256 Merge: 705c5ae ef256a2
257 Author: hosannajfull <miramonte23@gmail.com>
258 Date: Mon Dec 21 00:27:27 2015 -0500
```

```
259
260 Merge branch 'compile' of github.com:adamincera/dots into compile
261
262 merge
263
264 commit 705c5aef630cbda9b5e731a8b6e03d7d591e9360
265 Author: hosannajfull <miramonte23@gmail.com>
266 Date: Mon Dec 21 00:27:10 2015 -0500
267
268 rachel & hosanna fixed enqueue call
269
270 commit 241eff4d6ab8d73bf27c7bed5ed0d5f132055ad5
271 Author: rgordon <rcgordon@umass.edu>
272 Date: Sun Dec 20 23:48:09 2015 -0500
273
274 Test case
275
276 commit ef256a2dcadcf54609620fd2e6a931c5768bee09
277 Merge: 2a4799f 84cdfcc
278 Author: Adam Incera <aji2112@columbia.edu>
279 Date: Sun Dec 20 23:32:33 2015 -0500
280
281 merge
282
283 commit 2a4799f32cfb6373200ba89f66863d813b6f7b7a
284 Author: Adam Incera <aji2112@columbia.edu>
285 Date: Sun Dec 20 23:32:08 2015 -0500
286
287 pulling
288
289 commit e73a862e8e193c77ea0bb656792d93412396a973
290 Author: Adam Incera <aji2112@columbia.edu>
291 Date: Sun Dec 20 23:31:24 2015 -0500
292
293 added initialization into graph_plus_node()
294
295 commit 84cdfcceda948f345ce831a5ef3ed892afac8ca4
296 Merge: a706f7a fcc869e
297 Author: rgordon <rcgordon@umass.edu>
298 Date: Sun Dec 20 23:27:09 2015 -0500
299
300 Merge branch 'compile' of https://github.com/adamincera/dots into compile
301
302 merge
303
304 commit a706f7af90e14bcb7888b27185db9e849bb618da
305 Author: rgordon <rcgordon@umass.edu>
306 Date: Sun Dec 20 23:26:49 2015 -0500
307
308 fixed node + node
309
310 commit fcc869ed89263b5a581c5465a510455ddf48b492
311 Author: Adam Incera <aji2112@columbia.edu>
312 Date: Sun Dec 20 23:23:59 2015 -0500
```

```
313
314     simplified dijkstra
315
316 commit 02fdef29963dc95b880a803562300346a14290cb
317 Merge: 83f7c1a 9be880b
318 Author: Adam Incera <aji2112@columbia.edu>
319 Date: Sun Dec 20 23:12:47 2015 -0500
320
321     mergin
322
323 commit 83f7c1aef9741b6f7c4b6a2295c0e810417508bc
324 Author: Adam Incera <aji2112@columbia.edu>
325 Date: Sun Dec 20 23:12:18 2015 -0500
326
327     test for initializing graph in add_node
328
329 commit e3101e2ca2cbdeed3fd9089d6935db0352d36b69
330 Author: Adam Incera <aji2112@columbia.edu>
331 Date: Sun Dec 20 23:10:12 2015 -0500
332
333     pulling
334
335 commit 4e2321b2d7f9cf7714fa871317ab0e93b578d0d3
336 Author: Adam Incera <aji2112@columbia.edu>
337 Date: Sun Dec 20 23:09:46 2015 -0500
338
339     integrated init_graph() into add_node
340
341 commit 9be880b6e2fef401c493e7f175963989920e125b
342 Author: hosannajfull <miramonte23@gmail.com>
343 Date: Sun Dec 20 23:03:19 2015 -0500
344
345     fleshed out test case for ine and oute
346
347 commit 099c99b9c483fe5cfef5c479267e9f15ec78a859
348 Author: hosannajfull <miramonte23@gmail.com>
349 Date: Sun Dec 20 22:55:49 2015 -0500
350
351     fixed oute
352
353 commit f8e5bc6865ef92f974a2bf69b831d2e9f930f51b
354 Merge: 8b0f6ec 90d7f7a
355 Author: hosannajfull <miramonte23@gmail.com>
356 Date: Sun Dec 20 22:08:08 2015 -0500
357
358     Merge branch 'compile' of github.com:adamincera/dots into compile
359
360 commit 8b0f6ecdf831c11b473684a00de355e229fa8598
361 Author: hosannajfull <miramonte23@gmail.com>
362 Date: Sun Dec 20 22:08:02 2015 -0500
363
364     progress
365
366 commit 90d7f7ae1613ce5a2a1e7a66d0bfdb1818a616e1
```

```
367 Author: Adam Incera <aji2112@columbia.edu>
368 Date: Sun Dec 20 22:03:47 2015 -0500
369
370     fixed dict removal
371
372 commit 25078894c0fe748dd256351977831b062958d6ef
373 Merge: db88290 6ef44bb
374 Author: Adam Incera <aji2112@columbia.edu>
375 Date: Sun Dec 20 21:41:09 2015 -0500
376
377     merge
378
379 commit db88290ae4748a9c653e4d7ceb9151eff2861d72
380 Author: Adam Incera <aji2112@columbia.edu>
381 Date: Sun Dec 20 21:41:00 2015 -0500
382
383     pulling again
384
385 commit 6ef44bbbc8c554e15d9d59b7c344a6705fd6f402
386 Author: rgordon <rcgordon@umass.edu>
387 Date: Sun Dec 20 21:38:27 2015 -0500
388
389     fixed the printing of random % fmt strings
390
391 commit 58f36f41fece18aaf675cde201abbc8660e5c34c
392 Merge: 65adba5 9dc3ce2
393 Author: rgordon <rcgordon@umass.edu>
394 Date: Sun Dec 20 21:23:16 2015 -0500
395
396     Merge branch 'compile' of https://github.com/adamincera/dots into compile
397
398     merge
399
400 commit 65adba5ff62c2456cd8d9939eaafbf3a8db68075
401 Author: rgordon <rcgordon@umass.edu>
402 Date: Sun Dec 20 21:23:15 2015 -0500
403
404     partial dijstras
405
406 commit 9dc3ce2c5d95d27db9ae37e916c9bcc3cc62de83
407 Author: hosannajfull <miramonte23@gmail.com>
408 Date: Sun Dec 20 21:21:02 2015 -0500
409
410     all the dots tests pushed
411
412 commit 44c60b245827fb250f833c4e210abc6beff488f3
413 Merge: cd03369 a4cc926
414 Author: Adam Incera <aji2112@columbia.edu>
415 Date: Sun Dec 20 21:13:48 2015 -0500
416
417     merge
418
419 commit cd0336936b5253f8c3c22c0d6a56a51be73e393c
420 Author: Adam Incera <aji2112@columbia.edu>
```

```
421 Date: Sun Dec 20 21:13:32 2015 -0500
422
423   pulling rachoho
424
425 commit a4cc9269a942b684c1c50e6df40ceb7720f0d605
426 Author: hosannajfull <miramonte23@gmail.com>
427 Date: Sun Dec 20 21:12:59 2015 -0500
428
429   remove
430
431 commit 1a703e81a775bc1c07cf3f29ef40c0687e52a640
432 Author: hosannajfull <miramonte23@gmail.com>
433 Date: Sun Dec 20 21:12:28 2015 -0500
434
435   ll
436
437 commit a67f79e5da30ca9698a1745313eedfeff221f6e6
438 Merge: 4233073 69c734c
439 Author: Adam Incera <aji2112@columbia.edu>
440 Date: Sun Dec 20 20:35:49 2015 -0500
441
442   pulling double pointer fix
443
444 commit 69c734c29fcedf721b256454f5caa15fae14c63c
445 Author: hosannajfull <miramonte23@gmail.com>
446 Date: Sun Dec 20 20:35:18 2015 -0500
447
448   fixes to graph_t declarations. min/max handling
449
450 commit 4233073102750450d5bb9cf816c942ba36bfd528
451 Merge: b6b6b9a bf565c1
452 Author: Adam Incera <aji2112@columbia.edu>
453 Date: Sun Dec 20 20:27:56 2015 -0500
454
455   pulled
456
457 commit b6b6b9adalb889d8b7778a79ac72639e67303ed9
458 Author: Adam Incera <aji2112@columbia.edu>
459 Date: Sun Dec 20 20:27:43 2015 -0500
460
461   pulling
462
463 commit f5747f4e7da7c1e4a9929979d0239d631cce0093
464 Author: Adam Incera <aji2112@columbia.edu>
465 Date: Sun Dec 20 19:59:53 2015 -0500
466
467   pulling
468
469 commit bf565c1d7942792142eabe736d874b167217e379
470 Merge: 13361e6 d6cbe16
471 Author: hosannajfull <miramonte23@gmail.com>
472 Date: Sun Dec 20 19:01:46 2015 -0500
473
474   Merge branch 'compile' of github.com:adamincera/dots into compile
```

```
475
476 commit 13361e61fbbf66a2c322514efa9c32153e61d16c
477 Author: hosannajfull <miramonte23@gmail.com>
478 Date: Sun Dec 20 19:01:30 2015 -0500
479
480     print for node fixed
481
482 commit d6cbe167bb7c6977e395681b7fb7bc8181c21400
483 Merge: 284e2c2 73d4116
484 Author: rgordon <rcgordon@umass.edu>
485 Date: Sun Dec 20 18:17:25 2015 -0500
486
487     Merge branch 'compile' of https://github.com/adamincera/dots into compile
488
489     merge
490
491 commit 284e2c23e9d7eaf3289bee478245513f4be361d9
492 Author: rgordon <rcgordon@umass.edu>
493 Date: Sun Dec 20 18:17:15 2015 -0500
494
495     fixed while loop condition handling
496
497 commit 73d411672a58631cd732538115a4eaa4cbc5d26d
498 Merge: a2460fd 07a6alc
499 Author: hosannajfull <miramonte23@gmail.com>
500 Date: Sun Dec 20 18:04:47 2015 -0500
501
502     fixed merge conflict
503
504 commit a2460fd848e4a4f981fec65e6e6698dae7083a08
505 Author: hosannajfull <miramonte23@gmail.com>
506 Date: Sun Dec 20 18:03:01 2015 -0500
507
508     fancy graph assignment, fixing other stuff
509
510 commit 07a6alc5c82d1f150774e9322428fef0d69d2c04
511 Author: rgordon <rcgordon@umass.edu>
512 Date: Sun Dec 20 16:43:21 2015 -0500
513
514     fixes to while loop
515
516 commit 865461a8da1968319b11ab30937d9d49e0d150dc
517 Merge: c860bcf ed05ff9
518 Author: rgordon <rcgordon@umass.edu>
519 Date: Sun Dec 20 16:25:47 2015 -0500
520
521     Merge branch 'compile' of https://github.com/adamincera/dots into compile
522
523     merge
524
525 commit c860bcf73d07e1ff68dc321d3698859bd290ebba
526 Author: rgordon <rcgordon@umass.edu>
527 Date: Sun Dec 20 16:25:41 2015 -0500
528
```

```
529     initial work on making while loops conform to the reference variable
        standard
530
531 commit ed05ff936228e4915f64e2e9feb6cd491916acf9
532 Merge: e7f6ecd 8d099f2
533 Author: Adam Incera <aji2112@columbia.edu>
534 Date: Sun Dec 20 16:13:28 2015 -0500
535
536     mergin
537
538 commit e7f6ecd98d222bc6bc0f45c5ac146f1de06e48e4
539 Author: Adam Incera <aji2112@columbia.edu>
540 Date: Sun Dec 20 16:12:57 2015 -0500
541
542     dict printing, fixed some C bugs
543
544 commit a9f74572627467da5896c9b20f528de5634ad67d
545 Merge: e2c76ee 8d099f2
546 Author: rgordon <rcgordon@umass.edu>
547 Date: Sun Dec 20 13:55:38 2015 -0500
548
549     Merge branch 'compile' of https://github.com/adamincera/dots into compile
550
551     merg
552
553 commit 8d099f23c7f63e32abe657af12dd359a057c2caa
554 Merge: f81bf59 e493890
555 Author: hosannajfull <miramonte23@gmail.com>
556 Date: Sun Dec 20 13:55:19 2015 -0500
557
558     Merge branch 'compile' of github.com:adamincera/dots into compile
559
560 commit f81bf5908eb8c4c6854f7a26dae4cb0330328982
561 Author: hosannajfull <miramonte23@gmail.com>
562 Date: Sun Dec 20 13:55:09 2015 -0500
563
564     fully tested len built in function
565
566 commit e2c76ee717353dc4acfe3d65229b3fbbda2ecf62
567 Merge: f0301b4 e493890
568 Author: rgordon <rcgordon@umass.edu>
569 Date: Sun Dec 20 13:19:28 2015 -0500
570
571     Merge branch 'compile' of https://github.com/adamincera/dots into compile
572
573     merge
574
575 commit f0301b4664e506a244cf21979b9c515127fd82bd
576 Author: rgordon <rcgordon@umass.edu>
577 Date: Sun Dec 20 13:19:22 2015 -0500
578
579     moved call rule into the term rule
580
581 commit e493890422a6c12fdf88400b44b51a9eb4004c59
```



```
582 Merge: e81fc67 1d469e7
583 Author: Yumeng Liao <yl2908@columbia.edu>
584 Date: Sun Dec 20 13:16:27 2015 -0500
585
586 Merge branch 'compile' of https://github.com/adamincera/dots into compile
587
588 commit e81fc67922be63cdd1687dd640a5b26715ad587c
589 Author: Yumeng Liao <yl2908@columbia.edu>
590 Date: Sun Dec 20 13:16:10 2015 -0500
591
592 fixed function args not taking in types problem
593
594 commit 05f5d928d69e486b407bc8f8db06740d16398a0f
595 Merge: a769efc 1d469e7
596 Author: rgordon <rcgordon@umass.edu>
597 Date: Sun Dec 20 12:51:32 2015 -0500
598
599 Merge branch 'compile' of https://github.com/adamincera/dots into compile
600
601 merge
602
603 commit a769efcbf69b5722b6b9c1817aecb053576d2664
604 Author: rgordon <rcgordon@umass.edu>
605 Date: Sun Dec 20 12:51:30 2015 -0500
606
607 minor change
608
609 commit 1d469e7be1d99a34f134d30ad702572e75049f5e
610 Merge: 235e792 e94b8cd
611 Author: hosannajfull <miramonte23@gmail.com>
612 Date: Sun Dec 20 12:46:03 2015 -0500
613
614 hoho and rachel and adamame to the rescue
615
616 commit 235e7929cb7b3ab9004679c6dcc276a0cca4f1da
617 Author: hosannajfull <miramonte23@gmail.com>
618 Date: Sun Dec 20 12:43:50 2015 -0500
619
620 progress
621
622 commit e94b8cd2e90d17521c7ecd6918ad04be746409ac
623 Merge: 3ec97b9 2d60a71
624 Author: rgordon <rcgordon@umass.edu>
625 Date: Sun Dec 20 11:26:57 2015 -0500
626
627 Merge branch 'compile' of https://github.com/adamincera/dots into compile
628
629 merge
630
631 commit 3ec97b973f2ed99f7b68d76ef35ef78c24e4bb6b
632 Author: rgordon <rcgordon@umass.edu>
633 Date: Sun Dec 20 11:26:39 2015 -0500
634
```

```
635     implemented c translation for generic function call. i.e. implemented
        handling of dealing with result vars
636
637 commit 2d60a71f17b0c4a206fc686c10f27e93cd289bd1
638 Merge: 1f09692 56edcb2
639 Author: Yumeng Liao <y.liao.2908@gmail.com>
640 Date: Sun Dec 20 11:08:48 2015 -0500
641
642     Merge branch 'compile' of https://github.com/adamincera/dots into compile
643
644 commit 1f09692c505abba250a022e78cee6b4379409598
645 Author: Yumeng Liao <y.liao.2908@gmail.com>
646 Date: Sun Dec 20 11:08:21 2015 -0500
647
648     gotta print stderr too... in case of segfault yayaya
649
650 commit 56edcb2bdd8c6e3e30da00769330b3e873ee375a
651 Merge: 160aa51 1a2112f
652 Author: hosannajfull <miramonte23@gmail.com>
653 Date: Sun Dec 20 11:01:02 2015 -0500
654
655     merge conflicts for handled
656
657 commit 160aa51ee47faad64be92a4aeb0433412d2613f
658 Author: hosannajfull <miramonte23@gmail.com>
659 Date: Sun Dec 20 10:58:42 2015 -0500
660
661     all member call functions working
662
663 commit 1a2112fdbbe3cb4ea579080942417bab83109094
664 Author: rgordon <rcgordon@umass.edu>
665 Date: Sun Dec 20 10:11:36 2015 -0500
666
667     made fixes to AccessAssign. it looks like maybe the underlying C function
        is broken
668
669 commit 0f82e8belbd8d2595127fc27124d3b07630ec988
670 Author: rgordon <rcgordon@umass.edu>
671 Date: Sun Dec 20 09:52:05 2015 -0500
672
673     fixed list access translation
674
675 commit 04f82b6d6b68598ced75040c953979360d161db6
676 Author: rgordon <rcgordon@umass.edu>
677 Date: Sun Dec 20 09:39:26 2015 -0500
678
679     fixed list printing
680
681 commit da3b51f9fbdfa0665c488d640ef3174cbf398f90
682 Author: rgordon <rcgordon@umass.edu>
683 Date: Sun Dec 20 08:26:58 2015 -0500
684
685     for dict progress
686
```

```
687 commit fcbf505f9efc0570cd323cd3a7f1e538700b1136
688 Merge: 41dae0d 8ebd9f0
689 Author: rgordon <rcgordon@umass.edu>
690 Date: Sun Dec 20 06:45:59 2015 -0500
691
692 Merge branch 'compile' of https://github.com/adamincera/dots into compile
693
694 merge
695
696 commit 8ebd9f023c68b6d6bcb22b7858556b4591443a43
697 Author: Adam Incera <aji2112@columbia.edu>
698 Date: Sun Dec 20 06:45:47 2015 -0500
699
700 put init_dict() into put_whatever
701
702 commit 41dae0de4b8546337d42082b5e1e1d45d08097e3
703 Merge: b6613da 8a6401c
704 Author: rgordon <rcgordon@umass.edu>
705 Date: Sun Dec 20 06:45:09 2015 -0500
706
707 Merge branch 'compile' of https://github.com/adamincera/dots into compile
708
709 merge
710
711 commit b6613da54f020c9f7d82564607c07bfcc40906fa
712 Author: rgordon <rcgordon@umass.edu>
713 Date: Sun Dec 20 06:45:06 2015 -0500
714
715 altered Dict declaration to initialize it to NULL. more work on for dict
716
717 commit 1fc17dda95c9758cf0fc549791b98111fda62573
718 Merge: 6883e45 8a6401c
719 Author: hosannajfull <miramonte23@gmail.com>
720 Date: Sun Dec 20 06:25:28 2015 -0500
721
722 Merge branch 'compile' of github.com:adamincera/dots into compile
723
724 commit 6883e458cdf0a6665a31b1934a02fc8b52dfaad9
725 Author: hosannajfull <miramonte23@gmail.com>
726 Date: Sun Dec 20 06:25:21 2015 -0500
727
728 out member f in progress
729
730 commit 8a6401c5d02064248edfb8clad41cf240f721158
731 Merge: e7fd4b1 fbbacd3
732 Author: Adam Incera <aji2112@columbia.edu>
733 Date: Sun Dec 20 06:22:57 2015 -0500
734
735 pulled pork
736
737 commit e7fd4b101010146d7c684068be8192b76bbbbeba
738 Author: Adam Incera <aji2112@columbia.edu>
739 Date: Sun Dec 20 06:22:28 2015 -0500
740
```

```
741     implemented edgeops! ~meaningful 6am commit message~
742
743 commit fbbacd3f43bf3c9eb2b8c64e81699e089956accf
744 Merge: 1214dda e588611
745 Author: rgordon <rcgordon@umass.edu>
746 Date: Sun Dec 20 05:54:59 2015 -0500
747
748     Merge branch 'compile' of https://github.com/adamincera/dots into compile
749
750     merge
751
752 commit 1214ddacf8206a801ebf958d33f01722d7b63dcb
753 Author: rgordon <rcgordon@umass.edu>
754 Date: Sun Dec 20 05:54:53 2015 -0500
755
756     fixing For loop translation over dict
757
758 commit e588611356c7e187fe99a25dbe4c0312a8848a75
759 Merge: 90c7b04 df7e52f
760 Author: Adam Incera <aji2112@columbia.edu>
761 Date: Sun Dec 20 05:53:59 2015 -0500
762
763     to: rachel from: adamame
764
765 commit 90c7b04d1f3df63ddb5d8dd036dc0502f3338702
766 Author: Adam Incera <aji2112@columbia.edu>
767 Date: Sun Dec 20 05:47:30 2015 -0500
768
769     sending things to rachel
770
771 commit df7e52f1a695d7f3c5648c334e015f1152a647c1
772 Merge: e085ef5 b8d46bb
773 Author: hosannajfull <miramonte23@gmail.com>
774 Date: Sun Dec 20 05:44:18 2015 -0500
775
776     Merge branch 'compile' of github.com:adamincera/dots into compile
777
778 commit e085ef5958acabc963c4b9806dcf42b3bf7bfb64
779 Author: hosannajfull <miramonte23@gmail.com>
780 Date: Sun Dec 20 05:44:04 2015 -0500
781
782     pop and enqueue implemented and tested
783
784 commit ab376d38b61c3blada36f3076a00256433d66ac5
785 Merge: 98c6f89 b8d46bb
786 Author: Adam Incera <aji2112@columbia.edu>
787 Date: Sun Dec 20 02:46:48 2015 -0500
788
789     pulled
790
791 commit 98c6f89dba91ea866d8dde397412084089c7b33e
792 Author: Adam Incera <aji2112@columbia.edu>
793 Date: Sun Dec 20 02:46:23 2015 -0500
794
```

```
795     tryna pull
796
797 commit b8d46bb4ae5e93c70cba74dbc2defc5fa01bcf05
798 Author: rgordon <rcgordon@umass.edu>
799 Date: Sun Dec 20 01:48:35 2015 -0500
800
801     fixed list literal translation
802
803 commit cc2e019aa2216ca68b8e607456717b8153c86bb4
804 Merge: a9b78cc 3f8550b
805 Author: hosannajfull <miramonte23@gmail.com>
806 Date: Sun Dec 20 01:38:32 2015 -0500
807
808     Merge branch 'compile' of github.com:adamincera/dots into compile
809
810 commit a9b78cc4750cd930ecafd96cdf25d77df7942a64
811 Author: hosannajfull <miramonte23@gmail.com>
812 Date: Sun Dec 20 01:38:23 2015 -0500
813
814     progress
815
816 commit 3f8550b283135b9b3ed874cdfbce0e10db90bc1
817 Merge: 9632c38 e561b42
818 Author: rgordon <rcgordon@umass.edu>
819 Date: Sun Dec 20 01:37:57 2015 -0500
820
821     Merge branch 'compile' of https://github.com/adamincera/dots into compile
822
823     merge
824
825 commit 9632c38ce43fe3a97507161fb8bdac3add94b28e
826 Author: rgordon <rcgordon@umass.edu>
827 Date: Sun Dec 20 01:37:54 2015 -0500
828
829     improvements to implementation of list literal
830
831 commit e561b4200c73246d33236525a2c2fd85bffff5ed
832 Merge: a793e08 bd4b52d
833 Author: hosannajfull <miramonte23@gmail.com>
834 Date: Sun Dec 20 01:18:13 2015 -0500
835
836     Merge branch 'compile' of github.com:adamincera/dots into compile
837
838 commit a793e08c4e462d89d32378c5249cb2d7800a297b
839 Author: hosannajfull <miramonte23@gmail.com>
840 Date: Sun Dec 20 01:18:00 2015 -0500
841
842     binop working for ints, longs, and floats
843
844 commit bd4b52d0a8f7192c529346e32bff0f49f6898b02
845 Merge: a33cdbf ea84af7
846 Author: rgordon <rcgordon@umass.edu>
847 Date: Sun Dec 20 00:24:13 2015 -0500
848
```

```
849 Merge branch 'compile' of https://github.com/adamincera/dots into compile
850
851 merge
852
853 commit a33cdbfcf712ec6ff104aa9f670e472f9ab56664
854 Author: rgordon <rcgordon@umass.edu>
855 Date: Sun Dec 20 00:24:02 2015 -0500
856
857 implemented list literal translation. still needs to be tested
858
859 commit ea84af7421f25ebb4cca048c839ad66e3fab196c
860 Author: hosannajfull <miramonte23@gmail.com>
861 Date: Sun Dec 20 00:14:07 2015 -0500
862
863 added the strliteral and numliteral to add auto var return
864
865 commit 1ef7378189f7c538b212eb5cb64a9e58091de457
866 Merge: f5883a8 eecfae5
867 Author: Adam Incera <aji2112@columbia.edu>
868 Date: Sat Dec 19 23:35:18 2015 -0500
869
870 pulled
871
872 commit f5883a848ffc187d0dc5da618946d4050e72a4b9
873 Author: Adam Incera <aji2112@columbia.edu>
874 Date: Sat Dec 19 23:35:00 2015 -0500
875
876 tryna pull
877
878 commit eecfae5f97be97343e39c7663ea5d22431929cbf
879 Author: hosannajfull <miramonte23@gmail.com>
880 Date: Sat Dec 19 22:31:46 2015 -0500
881
882 added printf that works
883
884 commit 04560b6f4c49ae87a8e95c991f60fd65a70291a7
885 Author: hosannajfull <miramonte23@gmail.com>
886 Date: Sat Dec 19 22:05:08 2015 -0500
887
888 altered Access translation to deal with pointers
889
890 commit 95232deef2443bd398c9c5e03c580ec6062bc33f
891 Merge: dleff43 5e3b9d7
892 Author: hosannajfull <miramonte23@gmail.com>
893 Date: Sat Dec 19 21:49:30 2015 -0500
894
895 Merge branch 'compile' of github.com:adamincera/dots into compile
896
897 g
898
899 commit dleff431c4f45e7cb13fb9d9651e8b14c49c9b93
900 Author: hosannajfull <miramonte23@gmail.com>
901 Date: Sat Dec 19 21:49:14 2015 -0500
902
```

```
903   Changed handling of Id, NumLiteral to work with pointers for result vars
904
905 commit 5e3b9d73500a515dc14c44fe655ecab91278112e
906 Merge: d8872c3 f3cc977
907 Author: Adam Incera <aji2112@columbia.edu>
908 Date: Sat Dec 19 21:27:10 2015 -0500
909
910   merging
911
912 commit d8872c3719212c7663e966e8a813d1056307c4d5
913 Author: Adam Incera <aji2112@columbia.edu>
914 Date: Sat Dec 19 21:25:04 2015 -0500
915
916   added len to analyzer
917
918 commit f3cc977205d01e00ec7073fe6e74591420280d39
919 Author: hosannajfull <miramonte23@gmail.com>
920 Date: Sat Dec 19 21:11:39 2015 -0500
921
922   added comment explanation for how to do automatic result variable handling.
   implemented Access with result_var handling
923
924 commit 5e45c5a5ed585a29e32f0facfalld65f19ac69f3e
925 Merge: f50676f a7e3012
926 Author: hosannajfull <miramonte23@gmail.com>
927 Date: Sat Dec 19 20:40:30 2015 -0500
928
929   merge handled
930
931 commit f50676f1026d14e2d016bfc4526c2d060e0f22e4
932 Merge: 7e7b70d f40e18a
933 Author: hosannajfull <miramonte23@gmail.com>
934 Date: Sat Dec 19 20:38:49 2015 -0500
935
936   altered function names for sexpr_type
937
938 commit a7e30128936e2fb244fed5372219f968787ab86d
939 Merge: eabe61b 341af15
940 Author: Adam Incera <aji2112@columbia.edu>
941 Date: Sat Dec 19 20:28:34 2015 -0500
942
943   pulling
944
945 commit eabe61b26c43c729ad80fe52ab2268f6be3d96ee
946 Author: Adam Incera <aji2112@columbia.edu>
947 Date: Sat Dec 19 20:28:15 2015 -0500
948
949   compiling version of let function
950
951 commit 341af15e60a0eef1636db0ccca852f6ba3b04df5
952 Author: Yumeng Liao <yl2908@columbia.edu>
953 Date: Sat Dec 19 20:19:57 2015 -0500
954
955   tried to fix it
```

```
956
957 commit 4e1856e072accb6056760e8a9111250e4b1f01be
958 Merge: 5adcae3 277f28f
959 Author: Yumeng Liao <y.liao.2908@gmail.com>
960 Date: Sat Dec 19 19:24:29 2015 -0500
961
962 Merge branch 'compile' of https://github.com/adamincera/dots into compile
963
964 commit 5adcae380eb8553f4c7ffbf25cfb00938701bd67
965 Author: Yumeng Liao <y.liao.2908@gmail.com>
966 Date: Sat Dec 19 19:24:15 2015 -0500
967
968 analyzer accessassign
969
970 commit 277f28f12fad566351b4669d16ff0abcf55f3eaf
971 Merge: 466f003 f40e18a
972 Author: rgordon <rcgordon@umass.edu>
973 Date: Sat Dec 19 18:57:21 2015 -0500
974
975 Merge branch 'compile' of https://github.com/adamincera/dots into compile
976
977 merge
978
979 commit 466f00306d2170d47708e38a4fab26293f51d002
980 Author: rgordon <rcgordon@umass.edu>
981 Date: Sat Dec 19 18:57:12 2015 -0500
982
983 fixed For loop declaration of auto var
984
985 commit 7e7b70d343737b97b84e29b9940d7ab282d4cbe3
986 Author: hosannajfull <miramonte23@gmail.com>
987 Date: Sat Dec 19 18:15:40 2015 -0500
988
989 prgress
990
991 commit f40e18a56ef3c56cc01c0b09a81901ac3a350164
992 Merge: f371a5e 7013610
993 Author: Adam Incera <aji2112@columbia.edu>
994 Date: Sat Dec 19 18:11:38 2015 -0500
995
996 pulling
997
998 commit f371a5e7ab65420c306bf05729e78992d364e316
999 Author: Adam Incera <aji2112@columbia.edu>
1000 Date: Sat Dec 19 18:11:24 2015 -0500
1001
1002 added tests for len functions
1003
1004 commit 70136107c0e8f271430cd1e9ab76e2144acfa56b
1005 Merge: b89f332 3bla8c1
1006 Author: rgordon <rcgordon@umass.edu>
1007 Date: Sat Dec 19 18:10:05 2015 -0500
1008
1009 Merge branch 'compile' of https://github.com/adamincera/dots into compile
```



```

1010
1011     merge
1012
1013 commit b89f3326d4164c3d9b929f1eaf471401f5f96d2a
1014 Author: rgordon <rcgordon@umass.edu>
1015 Date: Sat Dec 19 18:09:55 2015 -0500
1016
1017     implemented translation of custom function definitions. includes: updating
        the environment variables to include names of functions and local scope
        to add parameters to variable mappings
1018
1019 commit 3b1a8c1f27fe3a8516ed10121cdfb1c86ceb3c92
1020 Author: Adam Incera <aji2112@columbia.edu>
1021 Date: Sat Dec 19 18:08:19 2015 -0500
1022
1023     changed nodelist_t to regular old list_t
1024
1025 commit 4b4dffc02af854d22d747bf1ff4c1e2a7c92edbf
1026 Merge: 6ddeab2 4208526
1027 Author: Adam Incera <aji2112@columbia.edu>
1028 Date: Sat Dec 19 17:54:17 2015 -0500
1029
1030     pulling
1031
1032 commit 6ddeab22c40c3629715dae31cd1dc96588c78bc2
1033 Author: Adam Incera <aji2112@columbia.edu>
1034 Date: Sat Dec 19 17:53:52 2015 -0500
1035
1036     redefined node->in and out as dict<node, num>
1037
1038 commit 4208526b73136e528e0c00ffd9d53ad0027ba172
1039 Merge: 3b27467 e9088b5
1040 Author: Yumeng Liao <yl2908@columbia.edu>
1041 Date: Sat Dec 19 17:40:23 2015 -0500
1042
1043     Merge branch 'compile' of https://github.com/adamincera/dots into compile
1044
1045 commit 3b27467513345eff99f4bcaac423d6842d8a40e9
1046 Author: Yumeng Liao <yl2908@columbia.edu>
1047 Date: Sat Dec 19 17:40:06 2015 -0500
1048
1049     typo in analyzer, wrote access assign simple test
1050
1051 commit e9088b5cb869d9b848257b60d4262023a9e714ba
1052 Author: Adam Incera <aji2112@columbia.edu>
1053 Date: Sat Dec 19 17:36:11 2015 -0500
1054
1055     snippet for list access assignment
1056
1057 commit 65462b8ba7c881b7dc1170d7b8dbd5977e3bc78e
1058 Merge: 0537c03 e4a73d7
1059 Author: Adam Incera <aji2112@columbia.edu>
1060 Date: Sat Dec 19 17:04:34 2015 -0500
1061

```

```
1062     pulling
1063
1064 commit 0537c0396d902278efbaa76b570826ab7edc65ca
1065 Author: Adam Incera <aji2112@columbia.edu>
1066 Date: Sat Dec 19 17:04:22 2015 -0500
1067
1068     access and accessAssign
1069
1070 commit e4a73d780e1f539aac90701754ea20910159e38a
1071 Author: hosannajfull <miramonte23@gmail.com>
1072 Date: Sat Dec 19 15:57:45 2015 -0500
1073
1074     dequeue
1075
1076 commit 66f7fd1f2883aad57e5895fe0ddc55fdb324173e
1077 Merge: d5a9b1f c8f68d1
1078 Author: hosannajfull <miramonte23@gmail.com>
1079 Date: Sat Dec 19 15:35:15 2015 -0500
1080
1081     Merge branch 'compile' of github.com:adamincera/dots into compile
1082
1083 commit d5a9b1f4f50a60bd47a606ef04a625334e7e7921
1084 Author: hosannajfull <miramonte23@gmail.com>
1085 Date: Sat Dec 19 15:35:08 2015 -0500
1086
1087     preliminary dequeue
1088
1089 commit c8f68d191159cdaa8567bed6f97e5507430a3762
1090 Merge: 27dbd80 aad6167
1091 Author: Adam Incera <aji2112@columbia.edu>
1092 Date: Sat Dec 19 15:14:26 2015 -0500
1093
1094     pulleen
1095
1096 commit 27dbd80db0edca2295a55ca7351b33e79cf00732
1097 Author: Adam Incera <aji2112@columbia.edu>
1098 Date: Sat Dec 19 15:11:42 2015 -0500
1099
1100     fixed pop
1101
1102 commit aad6167ee56d4b0092a4b0ac09e95d32b5ef8674
1103 Author: hosannajfull <miramonte23@gmail.com>
1104 Date: Sat Dec 19 14:49:17 2015 -0500
1105
1106     revert list of list impossible
1107
1108 commit 42693e07d5f981d0feca7c838f04518ddfa986fe
1109 Author: hosannajfull <miramonte23@gmail.com>
1110 Date: Sat Dec 19 14:47:18 2015 -0500
1111
1112     added enqueue dequeue for all types
1113
1114 commit 9320b9b55abeladd54aef69daf81770f0423971a
1115 Author: hosannajfull <miramonte23@gmail.com>
```

```
1116 Date: Sat Dec 19 14:39:20 2015 -0500
1117
1118     add back add front
1119
1120 commit 0a29c22fc86d26bbc336906ed50b3bc86de9e31c
1121 Merge: d87d96f 5ec5a53
1122 Author: hosannajfull <miramonte23@gmail.com>
1123 Date: Sat Dec 19 12:51:05 2015 -0500
1124
1125     merge conflicts handled
1126
1127 commit d87d96f0a2a49a689c21731034733fd25eaf7cd4
1128 Author: hosannajfull <miramonte23@gmail.com>
1129 Date: Sat Dec 19 12:48:29 2015 -0500
1130
1131     added MemberCall for enqueue in Analyzer
1132
1133 commit 5ec5a5349989cd4034a012d2a883784d9d16541e
1134 Merge: dcc692a a8627f5
1135 Author: Adam Incera <aji2112@columbia.edu>
1136 Date: Sat Dec 19 12:47:06 2015 -0500
1137
1138     pulling
1139
1140 commit dcc692a10bfc2d77545c78f7e2d9e5aa6fbab4bb
1141 Author: Adam Incera <aji2112@columbia.edu>
1142 Date: Sat Dec 19 12:46:48 2015 -0500
1143
1144     added min and max for dicts and lists and tests and snippets
1145
1146 commit a8627f565d7d331f7155c72fff6358dd8186d7ec
1147 Author: rgordon <rcgordon@umass.edu>
1148 Date: Sat Dec 19 12:29:30 2015 -0500
1149
1150     removed references to MemberVar
1151
1152 commit c9015f18b637c11471d4aca5a97577218ffbe4ed
1153 Author: rgordon <rcgordon@umass.edu>
1154 Date: Sat Dec 19 12:01:32 2015 -0500
1155
1156     moved old memberVar handling into memberCall. adjusted memberCall
1157
1158 commit d8cb810123e4a9aaf436780e24d44754e10f128f
1159 Author: rgordon <rcgordon@umass.edu>
1160 Date: Sat Dec 19 11:00:08 2015 -0500
1161
1162     fixed parsing rules so that dijkstra's example now compiles. and no shift/
        reduce conflicts
1163
1164 commit 74dc455d12d305d21f661b79dcb61ecabe6201e4
1165 Author: Yumeng Liao <y.liao.2908@gmail.com>
1166 Date: Sat Dec 19 05:06:30 2015 -0500
1167
1168     ok doing better... uncommented assign
```

```
1169
1170 commit b5d9e045dd4835d7a0f9c823ed923a395bf6a60e
1171 Author: Yumeng Liao <y.liao.2908@gmail.com>
1172 Date: Sat Dec 19 04:58:37 2015 -0500
1173
1174     separated logical exprs
1175
1176 commit a75b7a7665cdacb4dc786812c8cc18c3a0e4c854
1177 Author: Yumeng Liao <y.liao.2908@gmail.com>
1178 Date: Sat Dec 19 04:50:40 2015 -0500
1179
1180     fixes
1181
1182 commit f53f3f1961b1b7cfd4e24fcd0b4253262fe07e30
1183 Merge: 72f7c7a f0098c6
1184 Author: Adam Incera <aji2112@columbia.edu>
1185 Date: Sat Dec 19 03:06:57 2015 -0500
1186
1187     pulling
1188
1189 commit 72f7c7a402b104a26d3a2ebb6e03e06dac4e8eca
1190 Author: Adam Incera <aji2112@columbia.edu>
1191 Date: Sat Dec 19 03:06:45 2015 -0500
1192
1193     added dict remove and snippets for adding nodes and graphs
1194
1195 commit f0098c68235cf4406cf76bbe040119d4bf893d45
1196 Merge: 8a12b32 d3d684e
1197 Author: rgordon <rcgordon@umass.edu>
1198 Date: Sat Dec 19 03:02:25 2015 -0500
1199
1200     Merge branch 'compile' of https://github.com/adamincera/dots into compile
1201
1202     merge
1203
1204 commit 8a12b323c0fa4d193f442a54461ffaef38ae184d
1205 Author: rgordon <rcgordon@umass.edu>
1206 Date: Sat Dec 19 03:02:16 2015 -0500
1207
1208     old versions of expr parse rule added in comments. additional test files.
1209     some menhir test cases
1210
1211 commit d3d684e89bd97728138d5424a5edfbb2be83e741
1212 Merge: b114ceb 5b60c2e
1213 Author: hosannajfull <miramonte23@gmail.com>
1214 Date: Sat Dec 19 02:01:22 2015 -0500
1215
1216     Merge branch 'compile' of github.com:adamincera/dots into compile
1217
1218 commit b114ceb92722b48c99a585a420f610c298823b01
1219 Author: hosannajfull <miramonte23@gmail.com>
1220 Date: Sat Dec 19 02:01:07 2015 -0500
1221
1222     added the binop add string cases and LEQ for strings
```

```
1222
1223 commit 0eab6f18a1bf3af3668e6aefb2cd3d591955f374
1224 Author: Adam Incera <aji2112@columbia.edu>
1225 Date: Sat Dec 19 01:52:12 2015 -0500
1226
1227     node plus node, graph plus node
1228
1229 commit 5b60c2eb2860ed48a4ca5435ca7fc84bc758a9d6
1230 Author: Adam Incera <aji2112@columbia.edu>
1231 Date: Sat Dec 19 00:58:48 2015 -0500
1232
1233     modified graph subtraction and fixed list concatenation
1234
1235 commit bebccacad0eaac2f48e913143feae44bf394bab8
1236 Author: rgordon <rcgordon@umass.edu>
1237 Date: Fri Dec 18 23:45:28 2015 -0500
1238
1239     mango and I fixed the conversion of For loops to string * expr
1240
1241 commit 38229fd51277f91a81043cff9f445e13438e75d4
1242 Author: Yumeng Liao <yl2908@columbia.edu>
1243 Date: Fri Dec 18 22:38:08 2015 -0500
1244
1245     have to add the node case
1246
1247 commit ce5a2232279f14689b1e5463da0c75ac3446cdf4
1248 Author: Yumeng Liao <y.liao.2908@gmail.com>
1249 Date: Fri Dec 18 22:06:08 2015 -0500
1250
1251     first attempt at fixing for loops to do 'for x in expr'
1252
1253 commit 7f2c428b09230fcd8a5011c4ccd4cd63447183
1254 Author: rgordon <rcgordon@umass.edu>
1255 Date: Fri Dec 18 20:42:01 2015 -0500
1256
1257     fixed dict printing
1258
1259 commit 7d2cc33083eb5e5ec1322e916a44eecd6840d6fc
1260 Author: rgordon <rcgordon@umass.edu>
1261 Date: Fri Dec 18 20:33:08 2015 -0500
1262
1263     fixed library includes
1264
1265 commit eda2688ce9d5f6a9f53e861142f209a42e02a07a
1266 Author: rgordon <rcgordon@umass.edu>
1267 Date: Fri Dec 18 20:31:53 2015 -0500
1268
1269     actually fixed merge conflict
1270
1271 commit 553be92fa6ef346c51e8b4b08ba1dc39687ceb9b
1272 Merge: dc20867 9368ba3
1273 Author: rgordon <rcgordon@umass.edu>
1274 Date: Fri Dec 18 20:29:18 2015 -0500
1275
```

```
1276     addressed merge conflict
1277
1278 commit dc20867f5b3db6135abe6f7226d1416694f5388f
1279 Author: rgordon <rcgordon@umass.edu>
1280 Date: Fri Dec 18 20:28:11 2015 -0500
1281
1282     implemented dict printing
1283
1284 commit 9368ba3fd266f03b0e466f043dbb3219a32cdaf5
1285 Author: Adam Incera <aji2112@columbia.edu>
1286 Date: Fri Dec 18 20:27:59 2015 -0500
1287
1288     it compiles now
1289
1290 commit d9c6dc28398cdbe536de16c4fd0ccc38d592d923
1291 Author: hosannajfull <miramonte23@gmail.com>
1292 Date: Fri Dec 18 20:24:41 2015 -0500
1293
1294     pusheen
1295
1296 commit 2a82bcc6767e84204c5ca47e8842f4ad7f9c3c0d
1297 Merge: f8da6b5 5d1c173
1298 Author: hosannajfull <miramonte23@gmail.com>
1299 Date: Fri Dec 18 19:01:18 2015 -0500
1300
1301     Merge branch 'compile' of github.com:adamincera/dots into compile
1302
1303 commit f8da6b5a0eb61f4b966f20ecf9a8885f5854dea0
1304 Author: hosannajfull <miramonte23@gmail.com>
1305 Date: Fri Dec 18 19:01:02 2015 -0500
1306
1307     fixed it
1308
1309 commit 5d1c173f9f8bb80eb6ed534bd0871c3386404aaa
1310 Author: Adam Incera <aji2112@columbia.edu>
1311 Date: Fri Dec 18 18:57:34 2015 -0500
1312
1313     fixed for node in graph snippet
1314
1315 commit 2b3e8af41f3c75152b44f83a4e544c87e3e68b6d
1316 Author: Adam Incera <aji2112@columbia.edu>
1317 Date: Fri Dec 18 18:42:15 2015 -0500
1318
1319     list access and corresponding test
1320
1321 commit 20e843f0ac4f11a93ce94c731e796e43de5996dd
1322 Merge: 3af0cd9 677b93f
1323 Author: Adam Incera <aji2112@columbia.edu>
1324 Date: Fri Dec 18 18:31:05 2015 -0500
1325
1326     pulling once again
1327
1328 commit 3af0cd9c9e0752d9f3806071bb4c2df931ca96ab
1329 Author: Adam Incera <aji2112@columbia.edu>
```

1330 Date: Fri Dec 18 18:30:37 2015 -0500  
1331  
1332 generalized list insertion/copying and added corresponding tests  
1333  
1334 commit 677b93fb5889e409be466fdd57f0a7a6ead98289  
1335 Author: Yumeng Liao <y.liao.2908@gmail.com>  
1336 Date: Fri Dec 18 18:28:48 2015 -0500  
1337  
1338 fixed some missing pattern matches, bug with nostmt  
1339  
1340 commit 8c007cf5af7e89c5eadd6883b9e86df9eb6f5127  
1341 Merge: 5520037 19e32cf  
1342 Author: Yumeng Liao <y.liao.2908@gmail.com>  
1343 Date: Fri Dec 18 18:20:50 2015 -0500  
1344  
1345 Merge branch 'compile' of <https://github.com/adamincera/dots> into compile  
1346  
1347 commit 5520037c35e4c065084cd6629782ca3c2ff1f8c6  
1348 Author: Yumeng Liao <y.liao.2908@gmail.com>  
1349 Date: Fri Dec 18 18:20:39 2015 -0500  
1350  
1351 AccessAssign first attempt  
1352  
1353 commit 19e32cf5fdb7d7a9d0161987a4107127503a2832  
1354 Merge: 7c86f44 efa7018  
1355 Author: hosannajfull <miramonte23@gmail.com>  
1356 Date: Fri Dec 18 17:52:58 2015 -0500  
1357  
1358 merge conflicts handled  
1359  
1360 commit 7c86f44b0d363730e8619c7345a8e6ba003b05fe  
1361 Author: hosannajfull <miramonte23@gmail.com>  
1362 Date: Fri Dec 18 17:51:47 2015 -0500  
1363  
1364 string of stmt  
1365  
1366 commit efa70180554b314eef0c75e0035b707aba6675a0  
1367 Author: Adam Incera <aji2112@columbia.edu>  
1368 Date: Fri Dec 18 16:51:50 2015 -0500  
1369  
1370 snippets for peek and pop  
1371  
1372 commit 7d1f7d08195b8038ee07da074b2febce69edcaae  
1373 Merge: 06348ce 63eb56b  
1374 Author: Adam Incera <aji2112@columbia.edu>  
1375 Date: Fri Dec 18 16:46:43 2015 -0500  
1376  
1377 merging  
1378  
1379 commit 06348cee49562728265008e4b74a711af585a42d  
1380 Author: Adam Incera <aji2112@columbia.edu>  
1381 Date: Fri Dec 18 16:46:21 2015 -0500  
1382  
1383 moar snippets

```
1384
1385 commit 63eb56b60685e849c785e4803502a41e6466d883
1386 Author: Yumeng Liao <y.liao.2908@gmail.com>
1387 Date: Fri Dec 18 16:30:35 2015 -0500
1388
1389     fancy graph decl tests
1390
1391 commit c8677a40abc7a7fe512283b7cd0947ea161247cd
1392 Author: Yumeng Liao <y.liao.2908@gmail.com>
1393 Date: Fri Dec 18 15:11:31 2015 -0500
1394
1395     some list issues let's see if this works
1396
1397 commit c05f35dbd12858f34fd6c5651fcfdc29737459833
1398 Author: Yumeng Liao <y.liao.2908@gmail.com>
1399 Date: Fri Dec 18 15:09:37 2015 -0500
1400
1401     forgot to look for the variable in symbol table
1402
1403 commit 62e66d575d7fac990b620d61ca09a8aff12771fe
1404 Author: Yumeng Liao <y.liao.2908@gmail.com>
1405 Date: Fri Dec 18 15:07:47 2015 -0500
1406
1407     trying to translate fancy graph declaration with braces
1408
1409 commit cb6a6faefe58e54da94567f181e435b6932c02d9
1410 Author: hosannajfull <miramonte23@gmail.com>
1411 Date: Fri Dec 18 14:23:55 2015 -0500
1412
1413     pattern match graphdef
1414
1415 commit b29dc5b9690a215b1c419e7f01f3deab276dde48
1416 Merge: 34204ac d0465a3
1417 Author: hosannajfull <miramonte23@gmail.com>
1418 Date: Fri Dec 18 14:19:33 2015 -0500
1419
1420     Merge branch 'compile' of github.com:adamincera/dots into compile
1421
1422 commit 34204ac2f3234d0b7d6df45605801419d531adb5
1423 Author: hosannajfull <miramonte23@gmail.com>
1424 Date: Fri Dec 18 14:19:18 2015 -0500
1425
1426     changed member var 'in' to 'ine' and 'out' to 'oute' because in was being
        read as 'in' token keyword
1427
1428 commit d0465a3478259112395bcaf0347d16638894ab8c
1429 Author: rgordon <rcgordon@umass.edu>
1430 Date: Fri Dec 18 14:19:00 2015 -0500
1431
1432     fixed list printing to list the node->data field
1433
1434 commit 25d7ab88fae08bc30a66a26e4d2182eb6813d717
1435 Merge: 67ae323 236cc64
1436 Author: rgordon <rcgordon@umass.edu>
```



```
1437 Date: Fri Dec 18 14:03:44 2015 -0500
1438
1439 Merge branch 'compile' of https://github.com/adamincera/dots into compile
1440
1441 merge `
1442
1443 commit 67ae32326f3fa51935aa5b7b133121ab8be8491e
1444 Author: rgordon <rcgordon@umass.edu>
1445 Date: Fri Dec 18 14:03:34 2015 -0500
1446
1447 implemented list printing
1448
1449 commit 236cc644fd1481f20044d65ce60e82136e5dda06
1450 Author: hosannajfull <miramonte23@gmail.com>
1451 Date: Fri Dec 18 13:57:36 2015 -0500
1452
1453 Mango and Hosanna added graphdef to parser, ast, sast, and typeconverter
1454
1455 commit 5e058c9668d8749e6243fd3fd050451ae2cc87a2
1456 Author: hosannajfull <miramonte23@gmail.com>
1457 Date: Fri Dec 18 12:26:38 2015 -0500
1458
1459 new parser rule to get rid of the shift reduce conflicts
1460
1461 commit 625f85060a04a0349a23491337320c8e9df29c27
1462 Author: rgordon <rcgordon@umass.edu>
1463 Date: Fri Dec 18 11:39:21 2015 -0500
1464
1465 added (expr) rule in parser
1466
1467 commit 22e2643b022c60544037b3bf971c8f092dbf6b7a
1468 Author: hosannajfull <miramonte23@gmail.com>
1469 Date: Thu Dec 17 21:11:38 2015 -0500
1470
1471 hoho and mangy fixed the Member Expr calls
1472
1473 commit d30201c3e9da6751dcc1262f5eb3fde9249132d6
1474 Merge: fad31c4 24fbbd0
1475 Author: Yumeng Liao <y.liao.2908@gmail.com>
1476 Date: Thu Dec 17 20:17:57 2015 -0500
1477
1478 Merge branch 'compile' of https://github.com/adamincera/dots into compile
1479
1480 commit fad31c402c8cf7c80ca6ed403e52c6d021bbf6f2
1481 Author: Yumeng Liao <y.liao.2908@gmail.com>
1482 Date: Thu Dec 17 20:17:37 2015 -0500
1483
1484 started changing access, membervar, membercall to expr * expr
1485
1486 commit 24fbbd0202ed9842dcfef014c78af1e2f9d4ece4
1487 Merge: 0f7027a ccf099a
1488 Author: Adam Incera <aji2112@columbia.edu>
1489 Date: Wed Dec 16 23:47:59 2015 -0500
1490
```

```
1491     pulling
1492
1493 commit 0f7027ade07a9cf860c76ab1ba8bb566bd6a2ce8
1494 Author: Adam Incera <aji2112@columbia.edu>
1495 Date: Wed Dec 16 23:47:25 2015 -0500
1496
1497     changed TABLE_SIZE from 1024 to 256
1498
1499 commit fc1977814060823577d3ab59adb9b68038104b33
1500 Author: Adam Incera <aji2112@columbia.edu>
1501 Date: Wed Dec 16 23:47:00 2015 -0500
1502
1503     fixed graph hashing
1504
1505 commit ccf099ab93d611c311cdefb5df559c256f92d740
1506 Author: Yumeng Liao <y.liao.2908@gmail.com>
1507 Date: Wed Dec 16 15:35:58 2015 -0500
1508
1509     tried to fix it so node_init function is called not in the global scope
1510
1511 commit ebec4ee3168d3c2ca0d15f31faf3742c25a82d4d
1512 Author: Yumeng Liao <y.liao.2908@gmail.com>
1513 Date: Wed Dec 16 05:07:18 2015 -0500
1514
1515     Well, sometimes you accidentally delete everything and then have to fix
1516     it.
1517
1518 commit 20654aef7e511e390636ce961a6d0df5b0df9fe8
1519 Author: Yumeng Liao <yl2908@columbia.edu>
1520 Date: Wed Dec 16 04:48:10 2015 -0500
1521
1522     debugged testing script, all tests should run but there might be some
1523     issues....
1524
1525 commit ee2701fb0addc027b61e1203395d62c919b0a832
1526 Author: Yumeng Liao <y.liao.2908@gmail.com>
1527 Date: Wed Dec 16 04:35:44 2015 -0500
1528
1529     moved tests around into folders
1530
1531 commit e2884a90f7038565f26ec541e46ebd41c6df9f38
1532 Author: Yumeng Liao <y.liao.2908@gmail.com>
1533 Date: Wed Dec 16 04:20:47 2015 -0500
1534
1535     modified testing script to allow for different folders of tests
1536
1537 commit c3b3e03c674555e0f497fe1d09cd2f6ddc1a7747
1538 Author: Yumeng Liao <yl2908@columbia.edu>
1539 Date: Wed Dec 16 03:50:38 2015 -0500
1540
1541     debugged analyzer.ml for binop... added a bunch of cases to suppress
1542     warnings even though in theory they should never be reached
```

```
1543 Merge: d5534df 080a55e
1544 Author: Yumeng Liao <y.liao.2908@gmail.com>
1545 Date: Wed Dec 16 03:39:03 2015 -0500
1546
1547 Merge branch 'compile' of https://github.com/adamincera/dots into compile
1548
1549 commit d5534dfd1c2a630136012aa36cfc4d21c1652e79
1550 Author: Yumeng Liao <y.liao.2908@gmail.com>
1551 Date: Wed Dec 16 03:38:29 2015 -0500
1552
1553 rest of binops completed as much as I can right now
1554
1555 commit 15010a4ad9dbd60c3794f57f0bcd7dbde31be121
1556 Author: Yumeng Liao <y.liao.2908@gmail.com>
1557 Date: Wed Dec 16 03:14:02 2015 -0500
1558
1559 just realized a ton of binop rules could be condensed....
1560
1561 commit e4a66524e64c4bccbc057f790036ac35584c1c6d
1562 Author: Yumeng Liao <y.liao.2908@gmail.com>
1563 Date: Wed Dec 16 03:09:34 2015 -0500
1564
1565 condensed an Ast to Sast rule
1566
1567 commit e71901f3e0a7d4f561f99a5229e550849be6274d
1568 Author: Yumeng Liao <y.liao.2908@gmail.com>
1569 Date: Wed Dec 16 03:08:16 2015 -0500
1570
1571 in analyzer, add + subtract of binop first draft
1572
1573 commit 080a55e28293764b3cbe9baee90a953664164bf1
1574 Author: Yumeng Liao <yl2908@columbia.edu>
1575 Date: Wed Dec 16 02:29:55 2015 -0500
1576
1577 added gitignore for .o and .a files in clib from setting up
1578
1579 commit 68ba704aeabceda7eaeebaef1d8ae65ac1139daf
1580 Author: Yumeng Liao <yl2908@columbia.edu>
1581 Date: Wed Dec 16 02:16:50 2015 -0500
1582
1583 node decl fixes compile, let's see if it passes tests...
1584
1585 commit 34dd071a40d688f88183cc72ece80b606863a350
1586 Author: Yumeng Liao <y.liao.2908@gmail.com>
1587 Date: Wed Dec 16 02:11:45 2015 -0500
1588
1589 first attempt at fixing nodes to not call init in global scope
1590
1591 commit a3326346b15821010d64e6e091f9bcd4df4c1614
1592 Merge: 827726a e86213f
1593 Author: Yumeng Liao <y.liao.2908@gmail.com>
1594 Date: Wed Dec 16 01:45:11 2015 -0500
1595
1596 fixing merge conflict
```

```
1597
1598 commit 827726a942872556f34340274bfe1c7c8755885c
1599 Author: rgordon <rcgordon@umass.edu>
1600 Date: Tue Dec 15 23:15:49 2015 -0500
1601
1602     mango and I semi-finished separating globals from regular statements and
1603     finagling the Sast tree into something more suited to C ast
1604
1605 commit 44a07ec1d6b0847f4fb5b6563fd36f2e29a13ec6
1606 Merge: 128de36 cebc5a4
1607 Author: hosannajfull <miramonte23@gmail.com>
1608 Date: Tue Dec 15 19:34:16 2015 -0500
1609
1610     Merge branch 'compile' of github.com:adamincera/dots into compile
1611
1612 commit 128de3614a7e1d3c8050905394cbe8b87cb96c26
1613 Merge: 189717c fe792c1
1614 Author: hosannajfull <miramonte23@gmail.com>
1615 Date: Tue Dec 15 19:33:59 2015 -0500
1616
1617     merge conflicts handled
1618
1619 commit cebc5a456db42ec0fa0a239cb7ebb3c1d0c5b04f
1620 Author: rgordon <rcgordon@umass.edu>
1621 Date: Tue Dec 15 19:25:49 2015 -0500
1622
1623     removed extra inclusion of graph.h
1624
1625 commit 189717c584cc1aeabc3c89e9e6a6e27b4d090351
1626 Author: hosannajfull <miramonte23@gmail.com>
1627 Date: Tue Dec 15 17:15:42 2015 -0500
1628
1629     program intermediate object update
1630
1631 commit fe792c17922d2232367c21f235c4beb328ac2039
1632 Merge: e548ab6 3ccc4cb
1633 Author: rgordon <rcgordon@umass.edu>
1634 Date: Tue Dec 15 16:24:40 2015 -0500
1635
1636     Merge branch 'compile' of https://github.com/adamincera/dots into compile
1637
1638     merge
1639
1640 commit e548ab68675c528d1e8764a1af5ff21cc67d6a08
1641 Author: rgordon <rcgordon@umass.edu>
1642 Date: Tue Dec 15 16:24:30 2015 -0500
1643
1644     changed Cast to only have stmts (no longer has exprs). moved the handling
1645     of print back into the translate_expr function in analyzer.ml -- this
1646     works because translate_expr now outputs things of type cstmt
1647
1648 commit 3ccc4cb7bae92a07041c703091feccc9545c08ac
1649 Merge: 2179f4f a70354b
1650 Author: Yumeng Liao <y.liao.2908@gmail.com>
```

```

1648 Date: Tue Dec 15 15:59:21 2015 -0500
1649
1650 Merge branch 'compile' of https://github.com/adamincera/dots into compile
1651
1652 commit 2179f4fae180032b9f2bd31477526da0c5680df4
1653 Author: Yumeng Liao <y.liao.2908@gmail.com>
1654 Date: Tue Dec 15 15:58:58 2015 -0500
1655
1656 tests for member functions and variables
1657
1658 commit a70354b6048bf771e10070790ed868deedde6bd4
1659 Merge: 1e0d332 badb885
1660 Author: Adam Incera <aji2112@columbia.edu>
1661 Date: Tue Dec 15 15:42:25 2015 -0500
1662
1663 pulling so i can push
1664
1665 commit 1e0d3329b41ab1dd68a8c364c0866c3316dc81c1
1666 Author: Adam Incera <aji2112@columbia.edu>
1667 Date: Tue Dec 15 15:40:46 2015 -0500
1668
1669 UNTESTED clib stuff. graph copy constructor/comparison have been tested and
1670 work. everything else compiles but that's it
1671
1672 commit badb8852ff83ee0bf3c3fb8d4cb6e936db430b40
1673 Author: rgordon <rcgordon@umass.edu>
1674 Date: Tue Dec 15 00:53:31 2015 -0500
1675
1676 added more verbose error message for parsing error. made fixes to dijkstras
1677 test case. added stub rule for fancy graph declaration notation
1678
1679 commit c9bacf8bc014e77566fd5aad1afe8db40e44d8
1680 Author: rgordon <rcgordon@umass.edu>
1681 Date: Tue Dec 15 00:30:26 2015 -0500
1682
1683 Altered assign to take 2 cexprs which led to changes in how Assign is being
1684 called in analyzer.ml. also added 'N-' to the beginning of node
1685 printouts. fixed test files
1686
1687 commit 1cdd25222998f75ae20e400d51f5847f907ff594
1688 Author: rgordon <rcgordon@umass.edu>
1689 Date: Mon Dec 14 23:47:21 2015 -0500
1690
1691 removed the *.out line in the root's .gigignore file. DO NOT PUT *.out IN .
1692 gitignore --- WE WANT TO TRACK .OUT FILES IN dtest. added all the
1693 missing .out files
1694
1695 commit 93df1a2968a928d343cdf8eb86645b733b4835ae
1696 Author: rgordon <rcgordon@umass.edu>
1697 Date: Mon Dec 14 23:44:45 2015 -0500
1698
1699 -- Altered automatic variables to behave like normal variables, including
1700 adding them to the symbol table so they can be referenced later
1701
1702 -- Restricted float printing to 3 decimal places

```

```
1696 -- Added function to convert Sast datatype to string.
1697 -- Added declaration checking to Ast.Id -> Sast.Id
1698 -- Implemented type checking and variable checking for Ast.For -> Sast.For
1699
1700 commit 363d2be5be1deb8308075497ea80049aaf8c2a93
1701 Author: hosannajfull <miramonte23@gmail.com>
1702 Date: Sun Dec 13 17:39:17 2015 -0500
1703
1704     enqueue dequeue remove min max implemented
1705
1706 commit 6c9ec9750e9d7d93fc5a6b4f2483feae6e3fe934
1707 Merge: 4efcd4a 148bd83
1708 Author: rgordon <rcgordon@umass.edu>
1709 Date: Sun Dec 13 15:58:40 2015 -0500
1710
1711     Merge branch 'compile' of https://github.com/adamincera/dots into compile
1712
1713     merge
1714
1715 commit 4efcd4aeea9a0be999c896a743c5647c7e58dd8b
1716 Author: rgordon <rcgordon@umass.edu>
1717 Date: Sun Dec 13 15:58:35 2015 -0500
1718
1719     added handling for checking types of print call
1720
1721 commit db899a5a079ce327c5a69b7ee1a5834dc4a634d0
1722 Author: rgordon <rcgordon@umass.edu>
1723 Date: Sun Dec 13 15:55:31 2015 -0500
1724
1725     updated readme with install instructs
1726
1727 commit 148bd83ba1965c9d41199b5c7ecc3ba1bd9e8cf9
1728 Merge: 45deb11 9d92232
1729 Author: hosannajfull <miramonte23@gmail.com>
1730 Date: Sun Dec 13 15:48:18 2015 -0500
1731
1732     Merge branch 'compile' of github.com:adamincera/dots into compile
1733
1734 commit 45deb119ca6df2f359b898e8522655ce906e487b
1735 Author: hosannajfull <miramonte23@gmail.com>
1736 Date: Sun Dec 13 15:48:01 2015 -0500
1737
1738     all warnings are done
1739
1740 commit 9d9223243a4b6527e2f7cd9264ca2a8d570f95da
1741 Author: Adam Incera <aji2112@columbia.edu>
1742 Date: Sun Dec 13 15:47:21 2015 -0500
1743
1744     snippets for printing list, and for adding strings
1745
1746 commit 6e4007b68e9fbae4c1a0490b95d0ed8a77a0bba
1747 Author: Adam Incera <aji2112@columbia.edu>
1748 Date: Sun Dec 13 15:02:08 2015 -0500
1749
```

```
1750     snippet for printing dicts
1751
1752 commit 414869db83a3ecd47883b3ed0b241f706422c5c7
1753 Merge: 058b610 0c9a2d5
1754 Author: Yumeng Liao <y.liao.2908@gmail.com>
1755 Date: Sun Dec 13 14:18:14 2015 -0500
1756
1757     Merge branch 'compile' of https://github.com/adamincera/dots into compile
1758
1759 commit 058b6100fc9189bee8f857f3772bf734372d3f94
1760 Author: Yumeng Liao <y.liao.2908@gmail.com>
1761 Date: Sun Dec 13 14:17:51 2015 -0500
1762
1763     forgot to add dijstras test
1764
1765 commit 0c9a2d5bc22f567075felceb53a52ade633aee8d
1766 Merge: 412f6f8 80d5136
1767 Author: rgordon <rcgordon@umass.edu>
1768 Date: Sun Dec 13 14:11:19 2015 -0500
1769
1770     merge
1771
1772 commit 412f6f857d6bf3f98ac6c4805059ec51819a4d93
1773 Author: rgordon <rcgordon@umass.edu>
1774 Date: Sun Dec 13 14:10:26 2015 -0500
1775
1776     new test case and splitting up types different in get_fmt_val funct
1777
1778 commit 80d51367159b8e619a4febf3f9078266e5ebf55c
1779 Author: Yumeng Liao <y.liao.2908@gmail.com>
1780 Date: Sat Dec 12 23:40:56 2015 -0500
1781
1782     tests to call range function in for loop
1783
1784 commit 0d7048f5e04d1fe7cae8623cf3b2dcbc045c9d3e
1785 Merge: 718adb1 80b44ce
1786 Author: Adam Incera <aji2112@columbia.edu>
1787 Date: Sat Dec 12 19:34:26 2015 -0500
1788
1789     pulling again again
1790
1791 commit 718adb1b52436c2b14dc809bd9256ce3249991d3
1792 Author: Adam Incera <aji2112@columbia.edu>
1793 Date: Sat Dec 12 19:34:03 2015 -0500
1794
1795     implemented for in dict c translation
1796
1797 commit 80b44cebf9fbb4b476d82a4ac0419eaa98c62a28
1798 Author: Yumeng Liao <yl2908@columbia.edu>
1799 Date: Sat Dec 12 18:56:14 2015 -0500
1800
1801     tests for calling range function with 1 or 2 args
1802
1803 commit 65e260796a356dde93d242e8947c358567733ddb
```

```
1804 Merge: 6ee720b 8640189
1805 Author: hosannajfull <miramonte23@gmail.com>
1806 Date: Sat Dec 12 19:02:40 2015 -0500
1807
1808     call works changed fdecl
1809
1810 commit 6ee720b1987cb993554ebb4d4550e5469d12ad62
1811 Author: hosannajfull <miramonte23@gmail.com>
1812 Date: Sat Dec 12 19:01:25 2015 -0500
1813
1814     call working with functions ya bish
1815
1816 commit 8640189a12d55691cd003153cc0727b906a45670
1817 Author: Adam Incera <aji2112@columbia.edu>
1818 Date: Sat Dec 12 18:35:42 2015 -0500
1819
1820     translated for key in dict
1821
1822 commit 404f898340b17f73e3f789ddd0daeca5771db5d1
1823 Author: rgordon <rcgordon@umass.edu>
1824 Date: Sat Dec 12 16:58:26 2015 -0500
1825
1826     fixed printing of nodes. adding setup rule to Makefile that will run 'make
1827         library' inside clib. altered print statement in runtest.py
1828
1829 commit cedb611e9038ef21304b76736bfa0eb229796837
1830 Merge: edf0a2b 5460531
1831 Author: Adam Incera <aji2112@columbia.edu>
1832 Date: Sat Dec 12 14:49:23 2015 -0500
1833
1834     pulling again again
1835
1836 commit edf0a2b1cbf67312aae7806b21b649aaf4e66511
1837 Author: Adam Incera <aji2112@columbia.edu>
1838 Date: Sat Dec 12 14:48:58 2015 -0500
1839
1840     tested other dicts
1841
1842 commit 5460531569dddf342c1d3f488725a3a52462873b
1843 Merge: 00aeabe d309f87
1844 Author: hosannajfull <miramonte23@gmail.com>
1845 Date: Sat Dec 12 14:46:25 2015 -0500
1846
1847     Merge branch 'compile' of github.com:adamincera/dots into compile
1848
1849 commit 00aeabe20d67e08e2faa6b2c947256692f5fd4c4
1850 Author: hosannajfull <miramonte23@gmail.com>
1851 Date: Sat Dec 12 14:46:08 2015 -0500
1852
1853     fdecl working thanks Ratchel:
1854
1855 commit d309f873f74597366614069cf1ede69018feaaf2
1856 Merge: edbe506 1659bbe
1857 Author: Adam Incera <aji2112@columbia.edu>
```



1857 Date: Sat Dec 12 14:39:51 2015 -0500  
1858  
1859 pulling again  
1860  
1861 commit edbe506355854ec845405f5fd819a3e06403f70d  
1862 Author: Adam Incera <aji2112@columbia.edu>  
1863 Date: Sat Dec 12 14:39:23 2015 -0500  
1864  
1865 fixed num putting  
1866  
1867 commit 1659bbelf355600c1649536861ae0f11c9b9e022  
1868 Author: Yumeng Liao <yl2908@columbia.edu>  
1869 Date: Sat Dec 12 14:36:25 2015 -0500  
1870  
1871 Fixed wrong folder name, now works as intended  
1872  
1873 commit a9995dcac8e4472fc296037d668118885834a059  
1874 Merge: 0013e30 aa3e2f6  
1875 Author: Yumeng Liao <yl2908@columbia.edu>  
1876 Date: Sat Dec 12 14:31:28 2015 -0500  
1877  
1878 Merge branch 'compile' of <https://github.com/adamincera/dots> into compile  
1879  
1880 commit 0013e302c3ac80301d65bad6575fefedba8f61bd  
1881 Author: Yumeng Liao <yl2908@columbia.edu>  
1882 Date: Sat Dec 12 14:30:50 2015 -0500  
1883  
1884 applied test script changes to the right branch this time....  
1885  
1886 commit aa3e2f6d778ef950a6fd8fba8fbafcc9falb9be5  
1887 Author: Adam Incera <aji2112@columbia.edu>  
1888 Date: Sat Dec 12 14:24:53 2015 -0500  
1889  
1890 fixed snippets typo  
1891  
1892 commit e86213f47aa084a3af79284131f2df6159400ae4  
1893 Author: Yumeng Liao <y.liao.2908@gmail.com>  
1894 Date: Sat Dec 12 14:12:34 2015 -0500  
1895  
1896 added handling of negative tests to testing script, deleted old one  
1897  
1898 commit 4337dd1276c9ed89ea3f26233989e4a61e119a79  
1899 Merge: d2580fe eaa8d76  
1900 Author: Adam Incera <aji2112@columbia.edu>  
1901 Date: Sat Dec 12 13:52:06 2015 -0500  
1902  
1903 pulling  
1904  
1905 commit d2580fe4e745049ffa2f72bcc0957734802ebf7f  
1906 Author: Adam Incera <aji2112@columbia.edu>  
1907 Date: Sat Dec 12 13:51:14 2015 -0500  
1908  
1909 added snippets for dicts  
1910

```

1911 commit eaa8d761e5fbba8a8215b9bde8385be1ce7f9a5f
1912 Author: hosannajfull <miramonte23@gmail.com>
1913 Date: Sat Dec 12 13:40:35 2015 -0500
1914
1915     node def
1916
1917 commit ab98dd2657da84ae08f08a81de58b8409cc79180
1918 Author: hosannajfull <miramonte23@gmail.com>
1919 Date: Fri Dec 11 23:39:38 2015 -0500
1920
1921     fixed parser rules
1922
1923 commit 0132432d45b41b2f3638e70ee63ae34b5e615558
1924 Merge: 6495cd7 f9212da
1925 Author: hosannajfull <miramonte23@gmail.com>
1926 Date: Fri Dec 11 23:24:55 2015 -0500
1927
1928     Merge branch 'compile' of github.com:adamincera/dots into compile
1929
1930 commit 6495cd712f46e616f18586c4ccd3e3cf7e314408
1931 Author: hosannajfull <miramonte23@gmail.com>
1932 Date: Fri Dec 11 23:24:52 2015 -0500
1933
1934     merger
1935
1936 commit f9212da61a12ab722e11e15434916b74calc1df1
1937 Author: rgordon <rcgordon@umass.edu>
1938 Date: Fri Dec 11 23:24:03 2015 -0500
1939
1940     mango added initial work for new type NodeDef
1941
1942 commit 16fcf935f07906a69977b41b20728c7218e73bd8
1943 Merge: 3d6bcd0 224155e
1944 Author: hosannajfull <miramonte23@gmail.com>
1945 Date: Fri Dec 11 23:14:35 2015 -0500
1946
1947     Merge branch 'compile' of github.com:adamincera/dots into compile
1948
1949 commit 3d6bcd07de58e260113b08852ca0b295351617da
1950 Author: hosannajfull <miramonte23@gmail.com>
1951 Date: Fri Dec 11 23:02:49 2015 -0500
1952
1953     finished typeconvertergit statusgit statusgit statusgit statusgit status!
1954     maybe
1955
1956 commit 224155e98e695103f09bbbc935871c63ca69783d
1957 Merge: 060a7ea 2c546ac
1958 Author: rgordon <rcgordon@umass.edu>
1959 Date: Fri Dec 11 21:09:13 2015 -0500
1960
1961     Merge branch 'compile' of https://github.com/adamincera/dots into compile
1962
1963     merge

```

```
1964 commit 060a7ead3c5fad95bfc0a5c13896b67b905abc49
1965 Author: rgordon <rcgordon@umass.edu>
1966 Date: Fri Dec 11 21:09:04 2015 -0500
1967
1968     altered the way print translation works
1969
1970 commit 2c546ac6f00ed8753fb6f4b7d6ee65c3d9458abe
1971 Author: hosannajfull <miramonte23@gmail.com>
1972 Date: Fri Dec 11 14:00:49 2015 -0500
1973
1974     s_expr done
1975
1976 commit 702d94d4e6c9ce2c2a2ecfea18d6a5d33534e902
1977 Merge: bc0e6df 804f95a
1978 Author: hosannajfull <miramonte23@gmail.com>
1979 Date: Fri Dec 11 11:03:05 2015 -0500
1980
1981     Merge branch 'compile' of github.com:adamincera/dots into compile
1982
1983 commit 804f95a8fd7049d432d83c9622bd29ad6c6c3fb5
1984 Author: Adam Incera <aji2112@columbia.edu>
1985 Date: Tue Dec 8 03:58:15 2015 -0500
1986
1987     dict is working for string keys. more testing needed for floats and void *
1988
1989 commit bc0e6df34c3790fbb356f070d4158a8b24109a47
1990 Merge: be0075e 56c1b47
1991 Author: hosannajfull <miramonte23@gmail.com>
1992 Date: Mon Dec 7 23:56:28 2015 -0500
1993
1994     Merge branch 'compile' of github.com:adamincera/dots into compile
1995
1996 commit be0075e4cae5d69915ab1f29fc41ec2ef63a2300
1997 Author: hosannajfull <miramonte23@gmail.com>
1998 Date: Mon Dec 7 23:56:10 2015 -0500
1999
2000     Ast.MemberVar complete
2001
2002 commit 56c1b47f0ff3caf276976ef7cbb381ac998698a5
2003 Author: Adam Incera <aji2112@columbia.edu>
2004 Date: Mon Dec 7 23:04:29 2015 -0500
2005
2006     removed commented out code from gdc
2007
2008 commit c1542b69ae6eadc5b43b17cbece31ea7868c1d12
2009 Author: Adam Incera <aji2112@columbia.edu>
2010 Date: Mon Dec 7 22:54:32 2015 -0500
2011
2012     added library and objects target to makefile
2013
2014 commit b45353a3ec488af3e3e9c45c60283091147e65a0
2015 Author: rgordon <rcgordon@umass.edu>
2016 Date: Mon Dec 7 22:46:07 2015 -0500
2017
```

```
2018   adam fixed the c linking for gcc
2019
2020 commit c6342f0b2a23def1678e63351ab293ba2debf3cb
2021 Merge: cff0152 5b9ba2b
2022 Author: Adam Incera <aji2112@columbia.edu>
2023 Date: Mon Dec 7 21:55:21 2015 -0500
2024
2025   pulling
2026
2027 commit cff0152ceb09be447200da744efe2ff661ad8d19
2028 Author: Adam Incera <aji2112@columbia.edu>
2029 Date: Mon Dec 7 21:54:48 2015 -0500
2030
2031   changed node data type to char *
2032
2033 commit 5b9ba2beaa776a7542123a479003cb6b3e095d91
2034 Author: rgordon <rcgordon@umass.edu>
2035 Date: Mon Dec 7 21:37:15 2015 -0500
2036
2037   fixed type used in Id to C translation. fixed include headers for clib
   files. removed extra line from gdc. added some new test cases
2038
2039 commit 56816a18501f8e88ee8102d533fe6030b231e256
2040 Author: hosannajfull <miramonte23@gmail.com>
2041 Date: Mon Dec 7 21:09:29 2015 -0500
2042
2043   parser changes to DictLiteral
2044
2045 commit 17b558d35693777638a4c3dd3272424c10c6e33f
2046 Merge: 5abd029 0930670
2047 Author: hosannajfull <miramonte23@gmail.com>
2048 Date: Mon Dec 7 21:06:42 2015 -0500
2049
2050   Merge branch 'compile' of github.com:adamincera/dots into compile
2051
2052   merge
2053
2054 commit 5abd0294bedb10de374cbc1a2a5b9f849e7ecb61
2055 Author: hosannajfull <miramonte23@gmail.com>
2056 Date: Mon Dec 7 21:06:25 2015 -0500
2057
2058   made functions part of statements and fixed compilation issues
2059
2060 commit 093067066d1d9b27985a3f8d45ea5a3988fcca30
2061 Merge: 7854be5 9c5fafc
2062 Author: rgordon <rcgordon@umass.edu>
2063 Date: Mon Dec 7 20:37:43 2015 -0500
2064
2065   merge
2066
2067 commit 9c5fafc94348891b932070f64a05b2784cd5fb57
2068 Merge: d538667 3534164
2069 Author: hosannajfull <miramonte23@gmail.com>
2070 Date: Mon Dec 7 20:28:14 2015 -0500
```

```

2071
2072 Merge branch 'compile' of github.com:adamincera/dots into compile
2073
2074 commit d538667eb8eedd8190cf797a36a3ea96b88c40c2
2075 Author: hosannajfull <miramonte23@gmail.com>
2076 Date: Mon Dec 7 20:27:55 2015 -0500
2077
2078 rachel fixed dict literal kv1
2079
2080 commit 7854be57e956bd97b624ebe57a33dbf915164bfe
2081 Merge: e23e4ff 3534164
2082 Author: rgordon <rcgordon@umass.edu>
2083 Date: Mon Dec 7 20:18:39 2015 -0500
2084
2085 Merge branch 'compile' of https://github.com/adamincera/dots into compile
2086
2087 merge
2088
2089 commit e23e4ff78cdac6c9d0c4362cdd9bc85f10f29f6a
2090 Author: rgordon <rcgordon@umass.edu>
2091 Date: Mon Dec 7 20:18:24 2015 -0500
2092
2093 added new clib include libraries. made for n in node expr translation
2094 simpler
2095
2096 commit 353416473c41baca89e78d95469e76fffbe20d46
2097 Author: Adam Incera <aji2112@columbia.edu>
2098 Date: Mon Dec 7 20:16:47 2015 -0500
2099
2099 renamed translations.c/h to something more useful, list.c/h
2100
2101 commit a5b5365831a1e4c372890a4fc84d2940f21f2c3b
2102 Merge: 76ef60f 7f12856
2103 Author: hosannajfull <miramonte23@gmail.com>
2104 Date: Mon Dec 7 19:25:35 2015 -0500
2105
2106 you used to you used to
2107
2108 commit 76ef60f591f9c92117be7436c8b843d0ec7ce6f3
2109 Author: hosannajfull <miramonte23@gmail.com>
2110 Date: Mon Dec 7 19:17:55 2015 -0500
2111
2112 you used to you used to
2113
2114 commit 7f12856fa664c8237d18f982f77948a94aa3d33f
2115 Author: rgordon <rcgordon@umass.edu>
2116 Date: Sun Dec 6 20:31:02 2015 -0500
2117
2118 added ListLiterals as a type to Ast and Sast. moved check_list handling to
2119 occur at the ListLiteral level. removed (commented out) AssignList type
2120 of Ast and Sast and moved all handling into just simple Assign type. (

```

2121 Author: rgordon <rcgordon@umass.edu>  
2122 Date: Sun Dec 6 19:55:28 2015 -0500  
2123  
2124 implemented block to C. other small implementations  
2125  
2126 commit 474550cdadd1259037320beae65cf1018abd02b6  
2127 Author: rgordon <rcgordon@umass.edu>  
2128 Date: Sun Dec 6 19:33:46 2015 -0500  
2129  
2130 fixed Assign def in parser (didn't end in a SEMI). moved the addition of 'v  
' and 'f' for func and var names to be added in analyzer instead of  
translate (so that you can call specific function names / var names as  
well). implemented node declaration.  
2131  
2132 commit a6474d213ab3b621ec7cea7425eae043d3571367  
2133 Author: rgordon <rcgordon@umass.edu>  
2134 Date: Sun Dec 6 19:07:09 2015 -0500  
2135  
2136 implemented rest of For loop to C except for 'dict' data type. implemented  
while loop to C. added some simple test cases (still need expected  
output files)  
2137  
2138 commit 83570804fc36cef76890997df095a4418f2bf203  
2139 Author: rgordon <rcgordon@umass.edu>  
2140 Date: Sun Dec 6 18:36:36 2015 -0500  
2141  
2142 added clib/graph.h to the list of includes  
2143  
2144 commit 6c26461ac3e4e6bd2f381e18763c350c5a270624  
2145 Author: rgordon <rcgordon@umass.edu>  
2146 Date: Sun Dec 6 01:57:18 2015 -0500  
2147  
2148 begin fleshing out for loops to c  
2149  
2150 commit ec8d0fe6b2d00c844fd4e8823999814fb2123e2f  
2151 Merge: 7e2c025 28e5c4f  
2152 Author: rgordon <rcgordon@umass.edu>  
2153 Date: Sun Dec 6 01:09:14 2015 -0500  
2154  
2155 pushed to wrong branch. fixing  
2156  
2157 commit 7e2c0256afce14f33fd11e26c48c9e696831b429  
2158 Merge: f51e890 caa274e  
2159 Author: rgordon <rcgordon@umass.edu>  
2160 Date: Sun Dec 6 01:07:18 2015 -0500  
2161  
2162 merge conflicts  
2163  
2164 commit f51e89049dde73ead4ea9719c6c4c36c1edb6e45  
2165 Author: rgordon <rcgordon@umass.edu>  
2166 Date: Sun Dec 6 01:05:27 2015 -0500  
2167  
2168 parser rule for ListDecl was wrong  
2169

```
2170 commit caa274e0773697aae8dfb4c5652ba7224f5d989b
2171 Merge: a88b704 8ad664f
2172 Author: hosannajfull <miramonte23@gmail.com>
2173 Date: Sun Dec 6 00:50:37 2015 -0500
2174
2175     fixed the merge conflict
2176
2177 commit a88b7046b47f4cbf6357224307199ace0c553548
2178 Author: hosannajfull <miramonte23@gmail.com>
2179 Date: Sun Dec 6 00:30:32 2015 -0500
2180
2181     ratchel
2182
2183 commit 8ad664fc79e6d32c9d9f45322742913721f89c39
2184 Author: rgordon <rcgordon@umass.edu>
2185 Date: Sat Dec 5 23:44:11 2015 -0500
2186
2187     initial work on getting list to c translation
2188
2189 commit d9636bba3a702decf5f94d8c181f20f497f785bb
2190 Merge: f4ffe22 4166f5e
2191 Author: hosannajfull <miramonte23@gmail.com>
2192 Date: Sat Dec 5 22:16:49 2015 -0500
2193
2194     Merge branch 'compile' of github.com:adamincera/dots into compile
2195
2196 commit f4ffe22de05f00ce2b2a49d5dd66f6eca4e6b2c4
2197 Author: hosannajfull <miramonte23@gmail.com>
2198 Date: Sat Dec 5 22:16:41 2015 -0500
2199
2200     assignList
2201
2202 commit 4166f5ealf51a4acdf590e18f8ccfc8b7c99b48c
2203 Author: rgordon <rcgordon@umass.edu>
2204 Date: Sat Dec 5 21:27:00 2015 -0500
2205
2206     fixed various bugs in typeConverter
2207
2208 commit 0aade866d412c9c180319ec808097a2bb688f38f
2209 Author: Yumeng Liao <y.liao.2908@gmail.com>
2210 Date: Sat Dec 5 02:34:24 2015 -0500
2211
2212     added == semantic checking, fixed more spacing issues
2213
2214 commit 0662952396a30e386eaf46e23667407fc57c53b3
2215 Author: Yumeng Liao <y.liao.2908@gmail.com>
2216 Date: Sat Dec 5 02:20:57 2015 -0500
2217
2218     added some logical operations semantic checking (last commit meant to say
        checking, it's late), fixed spacing
2219
2220 commit 6d177409debad6f4baa9096cc892ff156b8a826
2221 Author: Yumeng Liao <y.liao.2908@gmail.com>
2222 Date: Sat Dec 5 02:10:07 2015 -0500
```

2223  
2224 added subtraction semantic parsing, made error messages for semantic  
parsing more helpful  
2225  
2226 commit 3eb962f34557f29865c099541a26b59bfa73a08a  
2227 Merge: 9ecf51c bb6f307  
2228 Author: rgordon <rcgordon@umass.edu>  
2229 Date: Fri Dec 4 22:04:38 2015 -0500  
2230  
2231 fixed merge conflicts. but it doesn't compile so we need to fix that  
2232  
2233 commit 9ecf51c03c742b205dcd278a37cd960f0473e637  
2234 Author: rgordon <rcgordon@umass.edu>  
2235 Date: Fri Dec 4 22:00:53 2015 -0500  
2236  
2237 fleshed out the sast to cast translation. fixed the compile file and fixed  
the Makefile to actually use it  
2238  
2239 commit bb6f30715f1b2d30cdf551763a1d9629f781ff6f  
2240 Author: hosannajfull <miramonte23@gmail.com>  
2241 Date: Fri Dec 4 22:00:45 2015 -0500  
2242  
2243 pattern matching for type checking  
2244  
2245 commit 9cac4d89a0623e6b819a9ea8db4777e52b75b517  
2246 Author: hosannajfull <miramonte23@gmail.com>  
2247 Date: Fri Dec 4 19:27:05 2015 -0500  
2248  
2249 Mango placeholder types in analyzer to get it to compile  
2250  
2251 commit 974efff87bff31cc6037411fe34ba91f35bad54a  
2252 Merge: b75b609 369140f  
2253 Author: hosannajfull <miramonte23@gmail.com>  
2254 Date: Fri Dec 4 17:43:29 2015 -0500  
2255  
2256 Merge branch 'compile' of github.com:adamincera/dots into compile  
2257  
2258 commit b75b6091351fa126f8517ef91bc5aadda31dbc63  
2259 Author: hosannajfull <miramonte23@gmail.com>  
2260 Date: Fri Dec 4 17:43:16 2015 -0500  
2261  
2262 comments on what needs to change  
2263  
2264 commit 369140fecf094e261c270a5cf7956db7c20887ed  
2265 Author: Yumeng Liao <yl2908@columbia.edu>  
2266 Date: Thu Dec 3 01:15:02 2015 -0500  
2267  
2268 Started fixing translate\_stmt in analyzer.ml  
2269  
2270 commit 1e77665dd8e4dcb297dccde534141321596c1736  
2271 Author: Yumeng Liao <yl2908@columbia.edu>  
2272 Date: Thu Dec 3 00:04:52 2015 -0500  
2273



```
2274     fixed translate.ml to take c statement lists rather than strings, next up
        is analyzer
2275
2276 commit a7f90380bbf338a7412d402f00fc7eddd147679d
2277 Author: Yumeng Liao <yl2908@columbia.edu>
2278 Date: Wed Dec 2 22:49:59 2015 -0500
2279
2280     added bit to translate whole program calling predefined functions, cleared
        up library stuf
2281
2282 commit 4304b02a6fe0ab5cbd2672a091356ab58bfe36be
2283 Author: Yumeng Liao <yl2908@columbia.edu>
2284 Date: Wed Dec 2 22:24:15 2015 -0500
2285
2286     added library translating function, still have to clarify it though
2287
2288 commit 852dbf204844b55e0f9f6e72a6dcf7eddc7bcca5
2289 Merge: e038853 eda78c0
2290 Author: rgordon <rcgordon@umass.edu>
2291 Date: Wed Dec 2 21:55:58 2015 -0500
2292
2293     Merge branch 'compile' of https://github.com/adamincera/dots into compile
2294
2295     merge
2296
2297 commit e03885327826c00a0e4f8829375f8140150ff544
2298 Author: rgordon <rcgordon@umass.edu>
2299 Date: Wed Dec 2 21:55:47 2015 -0500
2300
2301     initial work on creating the C syntax tree
2302
2303 commit eda78c06a1fb165251d6bd39fc9513ad6ccb6105
2304 Merge: aaeabf8 1517189
2305 Author: Adam Incera <aji2112@columbia.edu>
2306 Date: Wed Dec 2 21:24:54 2015 -0500
2307
2308     merging
2309
2310 commit aaeabf831de770eb3541ffc0c29a16edc73f3671
2311 Author: Adam Incera <aji2112@columbia.edu>
2312 Date: Wed Dec 2 21:23:28 2015 -0500
2313
2314     added list things
2315
2316 commit 151718960842f5c5fd54504c0a8dfa707a326296
2317 Author: rgordon <rcgordon@umass.edu>
2318 Date: Tue Dec 1 17:47:06 2015 -0500
2319
2320     removed separate rules for LogAnd and LogOr and put them into regular
        Binops (added an operator for each of them)
2321
2322 commit 9f80b6a3080dec6ae2291b33f92353856715c37e
2323 Author: rgordon <rcgordon@umass.edu>
2324 Date: Tue Dec 1 17:21:29 2015 -0500
```

2325  
2326 fixed expected output for string literal and num literal test cases (should  
have been nothing since there is no print statement). added output  
file extensions to gitignore

2327  
2328 commit 4b8473972e9710f220803e5b7961b02683c8d56f  
2329 Author: rgordon <rgordon@umass.edu>  
2330 Date: Tue Dec 1 17:11:54 2015 -0500

2331  
2332 the .gitignore incorrectly had 'gdc' inside. switched it to 'dotc'

2333  
2334 commit 5d0b6c889a2bf1a3430d659794525146a88cdf9b  
2335 Author: Yumeng Liao <yl2908@columbia.edu>  
2336 Date: Tue Dec 1 16:27:45 2015 -0500

2337  
2338 testing script now prints nice summary of passing tests in alphabetical  
order

2339  
2340 commit 2e62d11ec597c9e4cb31c54368288482ff510939  
2341 Author: Yumeng Liao <yl2908@columbia.edu>  
2342 Date: Tue Dec 1 16:12:14 2015 -0500

2343  
2344 added .out files for test script from tests written yesterday on simple  
list + dict decls

2345  
2346 commit 3da7cf18591da8b4df58010773b2f0454cd578a6  
2347 Author: Yumeng Liao <yl2908@columbia.edu>  
2348 Date: Tue Dec 1 16:09:38 2015 -0500

2349  
2350 made requirements.txt for testing script, updated test guide

2351  
2352 commit 81293310c62073cac8e1bdb545f85c7ebb5c88  
2353 Merge: 99ac50b 42324b2  
2354 Author: Yumeng Liao <yl2908@columbia.edu>  
2355 Date: Tue Dec 1 16:00:59 2015 -0500

2356  
2357 Merge branch 'compile' of <https://github.com/adamincera/dots> into compile

2358  
2359 commit 99ac50b4alde5ada95346b02715b6a109241a83a  
2360 Author: Yumeng Liao <yl2908@columbia.edu>  
2361 Date: Tue Dec 1 15:54:32 2015 -0500

2362  
2363 added test for list library functions

2364  
2365 commit 42324b22066955980dcc5a3e1a38fb7e47695481  
2366 Author: rgordon <rgordon@umass.edu>  
2367 Date: Mon Nov 30 21:59:53 2015 -0500

2368  
2369 Mango removed dict decl and made dict declaration vdecl

2370  
2371 commit 8e2961054a9c38dbd2f2c2cc7734d61fb67f6e40  
2372 Merge: b09db0b dc45603  
2373 Author: rgordon <rgordon@umass.edu>  
2374 Date: Mon Nov 30 21:22:54 2015 -0500

```
2375
2376 Merge branch 'compile' of https://github.com/adamincera/dots into compile
2377
2378 merge
2379
2380 commit b09db0bb0ff8ad88d7b3d101a238e13713472aee
2381 Author: rgordon <rcgordon@umass.edu>
2382 Date: Mon Nov 30 21:22:29 2015 -0500
2383
2384 added code for translating a list declaration. BUT it still needs the C
      snippet. also changed permissions of runtest.py and testall.py to be
      executable. added new simple list test case. TODO: add expected output
2385
2386 commit dc45603dc4d9bc5f81fa184b4c7d150f3097f834
2387 Merge: 3845a2c 0b4375a
2388 Author: Adam Incera <aji2112@columbia.edu>
2389 Date: Mon Nov 30 20:08:34 2015 -0500
2390
2391 rachel pushed stuff :(
2392
2393 commit 3845a2c583314788279fb8d3f0d888c9494e3113
2394 Author: Adam Incera <aji2112@columbia.edu>
2395 Date: Mon Nov 30 20:07:53 2015 -0500
2396
2397 added snippet for copy
2398
2399 commit 0b4375aa0a87d3171f472e525263268e40dad2ba
2400 Merge: 15ad512 a6542ab
2401 Author: rgordon <rcgordon@umass.edu>
2402 Date: Mon Nov 30 20:00:04 2015 -0500
2403
2404 Merge branch 'compile' of https://github.com/adamincera/dots into compile
2405
2406 merge
2407
2408 commit 15ad51228551265137b92e09d8e02f8b8ce9c345
2409 Author: rgordon <rcgordon@umass.edu>
2410 Date: Mon Nov 30 19:59:57 2015 -0500
2411
2412 added comments. changed DictAssign to contain 2 exprs
2413
2414 commit a6542ab60c7924ca40c35ea96f64a57da7d98535
2415 Merge: fdf472e eba852f
2416 Author: Adam Incera <aji2112@columbia.edu>
2417 Date: Mon Nov 30 18:11:15 2015 -0500
2418
2419 pulling clib into compile
2420
2421 commit fdf472e316bc4ba01d0ad1749488c52e788cedbd
2422 Merge: ed5e1bb 701c540
2423 Author: Adam Incera <aji2112@columbia.edu>
2424 Date: Mon Nov 30 18:10:40 2015 -0500
2425
2426 pulling compile
```

2427  
2428 commit 701c5404a5faad318a20b83bc3a3d40962f7460f  
2429 Author: rgordon <rcgordon@umass.edu>  
2430 Date: Mon Nov 30 12:22:43 2015 -0500  
2431  
2432 finished / fixed Vdecl handling in the new format  
2433  
2434 commit 28e5c4f81a6bfc1996ccb7da8a33121f5509277a  
2435 Author: rgordon <rcgordon@umass.edu>  
2436 Date: Mon Nov 30 11:38:50 2015 -0500  
2437  
2438 changed final report Makefile to refer to the correct file  
2439  
2440 commit eba852f71c2a21b682eeef38c9f42c5da4e33207  
2441 Author: Adam Incera <aji2112@columbia.edu>  
2442 Date: Mon Nov 30 05:59:36 2015 -0500  
2443  
2444 added copy function, updated snippets  
2445  
2446 commit a01fa3fb3978348009678cf9c15a59f69a926493  
2447 Author: Adam Incera <aji2112@columbia.edu>  
2448 Date: Mon Nov 30 05:20:26 2015 -0500  
2449  
2450 fixed plus and plus\_equals functions  
2451  
2452 commit c2c57edc6a152d5151369a746fda7a323dd0a7a9  
2453 Author: Yumeng Liao <y.liao.2908@gmail.com>  
2454 Date: Sat Nov 28 23:50:12 2015 -0500  
2455  
2456 Made folder final-report with rough draft .tex file with basic sections for  
our final report  
2457  
2458 commit 91f6fc3860d7463cc424bd94924395ed7b3412b3  
2459 Author: rgordon <rcgordon@umass.edu>  
2460 Date: Fri Nov 27 23:56:58 2015 -0500  
2461  
2462 removed last reference to old hard coded variable tables. removed  
extraneous comment  
2463  
2464 commit b9c22a26e980fb8226119590bfbd542630a47d6c  
2465 Author: rgordon <rcgordon@umass.edu>  
2466 Date: Fri Nov 27 23:47:46 2015 -0500  
2467  
2468 variable declaration now works with the new environment setup. the print  
function still doesn't work. unclear whether the scope variables are  
truly updated  
2469  
2470 commit e263632981d5e3c38dbala619ff6c7dd1e113fef  
2471 Author: rgordon <rcgordon@umass.edu>  
2472 Date: Fri Nov 27 23:40:25 2015 -0500  
2473  
2474 altered analyzer.ml to add functions that convert the Ast to the Sast.  
Fixed incorrect types in Sast. Created a translation environment  
variable that can be used in semantic analysis.

2475  
2476 commit 02ef7cc396bd9e1f2606f5a6299b2c8206a148ab  
2477 Merge: ad45044 92df7d0  
2478 Author: rgordon <rcgordon@umass.edu>  
2479 Date: Fri Nov 27 10:54:24 2015 -0500  
2480  
2481 resolved merge conflict  
2482  
2483 commit 92df7d0d9a148d74f1bef3db2b27cc63cbf18807  
2484 Author: hosannajfull <miramonte23@gmail.com>  
2485 Date: Tue Nov 24 12:52:12 2015 -0500  
2486  
2487 env var integration:  
2488  
2489 commit ad45044715a11abe389ebf0a6cdceff619a924f7  
2490 Merge: 977cabb d495509  
2491 Author: rgordon <rcgordon@umass.edu>  
2492 Date: Tue Nov 24 11:18:20 2015 -0500  
2493  
2494 merged hosanna's changes  
2495  
2496 commit 977cabb8367842f870a34b6b69572e3674d9b46b  
2497 Author: rgordon <rcgordon@umass.edu>  
2498 Date: Tue Nov 24 11:14:43 2015 -0500  
2499  
2500 additional work on creating env var. new test cases  
2501  
2502 commit d49550956b2f133f35e1e96edcfd7424d6699d77  
2503 Author: hosannajfull <miramonte23@gmail.com>  
2504 Date: Tue Nov 24 10:30:55 2015 -0500  
2505  
2506 files merged  
2507  
2508 commit cf1e0bee61b1886bef74c2fb9c455d90305e7b67  
2509 Merge: ldc5b83 82761b5  
2510 Author: hosannajfull <miramonte23@gmail.com>  
2511 Date: Mon Nov 23 20:25:20 2015 -0500  
2512  
2513 merged all files  
2514  
2515 commit ldc5b8330463c839816e729b2f348cbe8fdcf262  
2516 Author: hosannajfull <miramonte23@gmail.com>  
2517 Date: Mon Nov 23 20:11:25 2015 -0500  
2518  
2519 sast  
2520  
2521 commit ed5e1bb5198f1657db9a6e939b25408e14c7dc17  
2522 Author: Adam Incera <aji2112@columbia.edu>  
2523 Date: Sun Nov 22 17:43:22 2015 -0500  
2524  
2525 wrote some snippets for translation  
2526  
2527 commit 82761b54a30454e7502440490f92a0a034a37e24  
2528 Author: Yumeng Liao <yl2908@columbia.edu>

2529 Date: Sat Nov 21 23:54:06 2015 -0500  
2530  
2531 added some dict tests  
2532  
2533 commit 380ff23d6dfd8c7034fe5dc72d635fe79a245486  
2534 Author: Yumeng Liao <yl2908@columbia.edu>  
2535 Date: Sat Nov 21 21:38:48 2015 -0500  
2536  
2537 list declaration, assignment, access tests  
2538  
2539 commit 3681084fcf74ef5086f559d476b917141f7072be  
2540 Author: rgordon <rcgordon@umass.edu>  
2541 Date: Mon Nov 16 10:19:56 2015 -0500  
2542  
2543 changed num literal parsing in scanner to include floats. removed  
unnecessary comment from analyzer.ml. altered num-assign test to  
assign a negative float to the variable. renamed testdots.py -> runtest  
.py. renamed tests.py -> testall.py.  
2544  
2545 commit d64c91c1d6cf8a245d1544a93a23f3890b74c881  
2546 Author: rgordon <rcgordon@umass.edu>  
2547 Date: Mon Nov 16 01:06:14 2015 -0500  
2548  
2549 fixed parsing rule for assignment operator. added test case for assigning a  
num (which passes - yay). fleshed out translation of Assign expr  
2550  
2551 commit 1e255c94d891c1f74a67c8433ab6db3fb462c72f  
2552 Author: rgordon <rcgordon@umass.edu>  
2553 Date: Mon Nov 16 00:46:45 2015 -0500  
2554  
2555 added placeholders for each case for translate\_expr and translate\_stmt to  
get rid of all those annoying warnings  
2556  
2557 commit b280b35987fc3804ee2343f08e1dfef7fa2d658b  
2558 Author: rgordon <rcgordon@umass.edu>  
2559 Date: Mon Nov 16 00:30:57 2015 -0500  
2560  
2561 cleaned up print parsing a little by adding a function that figures out  
what format string type each expression is  
2562  
2563 commit 1fdfb02362e6ffffaf41709076176c49df8efea04  
2564 Author: rgordon <rcgordon@umass.edu>  
2565 Date: Mon Nov 16 00:17:02 2015 -0500  
2566  
2567 altered .gitignore to only ignore those particular file extensions for  
filesi in the documentation folder. before it was ignoring .out files  
in dtest, which we DO want to be able to commit to the repo  
2568  
2569 commit ae6f91880717bb83bafde40b418456c196af73be  
2570 Author: rgordon <rcgordon@umass.edu>  
2571 Date: Mon Nov 16 00:13:06 2015 -0500  
2572  
2573 added check to testdots.py to make sure the .out file exists for a  
particular test case. altered way function call translation works:

```

    moved logic from translate.ml into analyzer.ml in the translate_expr
    function; changed it to printf with a format string so that things that
    aren't of type stirng can be printed
2574
2575 commit 8bb993ec94c2e35fe89f29e8066cc18e25312c26
2576 Author: rgordon <rcgordon@umass.edu>
2577 Date: Sun Nov 15 23:03:18 2015 -0500
2578
2579     altered variable declaration so that it maps local variables to indices and
        declares l1, l2, .... ln instead of using the original variable name
2580
2581 commit 785c00c266bba9928af7d67f3ad49af2cb620a2b
2582 Author: rgordon <rcgordon@umass.edu>
2583 Date: Sun Nov 15 22:42:37 2015 -0500
2584
2585     fixed adding variables to locals_indexes (involved turned locals_indexes
        into a ref). added placeholders for unhandled cases in ast.ml
2586
2587 commit 5b95741260579dfa9ebed11534fcad55ab69c376
2588 Author: rgordon <rcgordon@umass.edu>
2589 Date: Sun Nov 15 22:03:56 2015 -0500
2590
2591     changed it so that executables are compiled to files ending in '.exec' so
        that then the clean flag can properly remove those files. also changed
        it so that executables are compiled into the test directory instead of
        straight into src
2592
2593 commit 3b3b362812a06df371599e7e24be4fd2daff1e26
2594 Author: rgordon <rcgordon@umass.edu>
2595 Date: Sun Nov 15 21:56:04 2015 -0500
2596
2597     added rule to Makefile to delete *.outgdc files. changed argument parsing
        in testdots.py to use argparse module. fixed testdots.py so that it
        doesn't run the non-existent exeutable if compilation failed. fixed
        expected output for multi-hello-world.dots
2598
2599 commit 704d7c3f3533b7c6cde378c877e1e9c60aac8fe1
2600 Author: rgordon <rcgordon@umass.edu>
2601 Date: Sun Nov 15 21:30:24 2015 -0500
2602
2603     added rule to clean that removes all .c files in dtest
2604
2605 commit c956fdd02fdc3667565233bba8af514674a4080d
2606 Merge: 84337d9 b40061d
2607 Author: rgordon <rcgordon@umass.edu>
2608 Date: Sun Nov 15 21:29:10 2015 -0500
2609
2610     integrated parser branch into compile branch
2611
2612 commit b40061d38da265f9a6fae510fde6948ac43d05ee
2613 Author: rgordon <rcgordon@umass.edu>
2614 Date: Sun Nov 15 18:36:59 2015 -0500
2615
2616     got rid of random executables

```

```
2617
2618 commit a5421a1c36064766427f6d458f94cf26120f1220
2619 Author: rgordon <rcgordon@umass.edu>
2620 Date: Sun Nov 15 18:32:59 2015 -0500
2621
2622     added -B flag to diff command that fixes tests. changed testdots.py so that
           the flag -c will remove files; otherwise all output files are kept at
           the end of the test
2623
2624 commit c3f960bccaccb99b933a56f9b89929dcd6286928
2625 Author: Hosanna <miramonte23@gmail.com>
2626 Date: Sun Nov 15 18:23:53 2015 -0500
2627
2628     wording change
2629
2630 commit f4b3ed3c7890de962e8b8ee630a6269fbb0aada1
2631 Author: Adam Incera <aji2112@columbia.edu>
2632 Date: Sun Nov 15 18:21:32 2015 -0500
2633
2634     added range function in translations.h
2635
2636 commit d99bc7ff95f490db2b12fb97ac5f88aaca2ee788
2637 Author: Hosanna <miramonte23@gmail.com>
2638 Date: Sun Nov 15 18:08:38 2015 -0500
2639
2640     set -e added
2641
2642 commit 29f9e165d2b8074a6d5955a52c85bce12365345d
2643 Author: Yumeng Liao <yl2908@columbia.edu>
2644 Date: Sun Nov 15 17:44:09 2015 -0500
2645
2646     fixed testing script diff checking
2647
2648 commit 84337d9cdd9545e043b480f08442b2354b46f60c
2649 Author: Hosanna <miramonte23@gmail.com>
2650 Date: Sun Nov 15 17:41:04 2015 -0500
2651
2652     progress on vdecl
2653
2654 commit ff31564d51e4f26eab7d43f9a33df0023a161ac2
2655 Author: Yumeng Liao <yl2908@columbia.edu>
2656 Date: Sun Nov 15 17:32:01 2015 -0500
2657
2658     test harness works but output file is being poopy
2659
2660 commit 0441d1ddb364bee8251369400784309e319c4af9
2661 Author: Yumeng Liao <yl2908@columbia.edu>
2662 Date: Sun Nov 15 16:48:35 2015 -0500
2663
2664     changed the makefile to NOT accidentally delete all of our c libraries
           .....
2665
2666 commit 071e1ca4c36f2c73f2b8e385202d32ebbc1d28b6
2667 Merge: 3937078 b2044a2
```



```
2668 Author: rgordon <rcgordon@umass.edu>
2669 Date: Sun Nov 15 16:47:06 2015 -0500
2670
2671 Merge branch 'parser' of https://github.com/adamincera/dots into parser
2672
2673 pull merge
2674
2675 commit b2044a278c7aa17c49b92a375ef0d2ba746d429a
2676 Author: Yumeng Liao <yl2908@columbia.edu>
2677 Date: Sun Nov 15 16:31:00 2015 -0500
2678
2679 accidentally deleted our shell script...
2680
2681 commit 2cee4ced9cce0b744298aecfd99594c994c88f15
2682 Author: Yumeng Liao <yl2908@columbia.edu>
2683 Date: Sun Nov 15 16:13:23 2015 -0500
2684
2685 make clean removes executables too
2686
2687 commit c3daddf6a5e209aa7feale743b212f13d97bca2a
2688 Author: Yumeng Liao <yl2908@columbia.edu>
2689 Date: Sun Nov 15 16:11:34 2015 -0500
2690
2691 made .gitignore more comprehensive
2692
2693 commit fc1c7db32cdc94cfd8a45e23d23fd0aa6a80da53
2694 Author: Hosanna <miramonte23@gmail.com>
2695 Date: Sun Nov 15 15:42:10 2015 -0500
2696
2697 working built in funky funcs
2698
2699 commit 3937078f2fafdd69590015df521c3366d9cc4d00
2700 Merge: ccb3212 f500303
2701 Author: rgordon <rcgordon@umass.edu>
2702 Date: Sun Nov 15 15:18:43 2015 -0500
2703
2704 Merge branch 'clib' into parser
2705
2706 commit ccb3212c429b77717e9f176324f2bc003b20edf5
2707 Author: Hosanna <miramonte23@gmail.com>
2708 Date: Sun Nov 15 15:17:44 2015 -0500
2709
2710 deleted interpret microC copy
2711
2712 commit 5a9dc78ab564eea5110b41969a5560b18e0b8dc2
2713 Author: rgordon <rcgordon@umass.edu>
2714 Date: Sat Nov 14 16:41:05 2015 -0500
2715
2716 fixes to shell script and hello world test files
2717
2718 commit f5003033ddd4b56572f74e97343a332504ff7d3a
2719 Author: Adam Incera <aji2112@columbia.edu>
2720 Date: Sat Nov 14 16:39:43 2015 -0500
2721
```

```
2722     restructured node removal, added edge removal
2723
2724 commit 8828505f14ec1db4cf282a1e039d9cf12837dab2
2725 Merge: bc82f3b 461e632
2726 Author: rgordon <rcgordon@umass.edu>
2727 Date: Sat Nov 14 16:11:03 2015 -0500
2728
2729     Merge branch 'parser' of https://github.com/adamincera/dots into parser
2730
2731     aaaaahhhhh merges
2732
2733 commit bc82f3bfd0f39a2183f69d7ed71ee7d6012dfac6
2734 Author: rgordon <rcgordon@umass.edu>
2735 Date: Sat Nov 14 16:10:47 2015 -0500
2736
2737     OMMMG HELLO WORLD WORKS
2738
2739 commit 461e632be28d37408b8f14560d210bd284b2106f
2740 Author: Yumeng Liao <yl2908@columbia.edu>
2741 Date: Sat Nov 14 15:11:01 2015 -0500
2742
2743     test automation script and some tests and out files, option to print diff
2744     with -k flag
2745
2746 commit fbdd4595a863649ea36f01ab0939510e3987e8ba
2747 Merge: 78b3b2a ca78e91
2748 Author: rgordon <rcgordon@umass.edu>
2749 Date: Sat Nov 14 14:58:19 2015 -0500
2750
2751     Merge branch 'parser' of https://github.com/adamincera/dots into parser
2752
2753     whoops didn't pull
2754
2755 commit 78b3b2a801723cf33e2014fdd8556e2a17915580
2756 Author: rgordon <rcgordon@umass.edu>
2757 Date: Sat Nov 14 14:58:09 2015 -0500
2758
2759     fixed function declaration parsing and pretty printing
2760
2761 commit ca78e9192ff7440561213a32031c0465561f2f1d
2762 Author: Yumeng Liao <yl2908@columbia.edu>
2763 Date: Sat Nov 14 14:38:27 2015 -0500
2764
2765     hello world endgame tests
2766
2767 commit 1589aaae506dc06ec6be7d4204cbc46e81ae116c
2768 Author: rgordon <rcgordon@umass.edu>
2769 Date: Sat Nov 14 14:05:26 2015 -0500
2770
2771     fixed parser definition of an fdecl. started writing compiler
2772
2773 commit 9077169727517d3c4ac94fe63e52cbc911b7fa65
2774 Author: rgordon <rcgordon@umass.edu>
2775 Date: Sat Nov 14 12:13:10 2015 -0500
```

2775  
2776 added differentiation between string literals and num literals.  
2777  
2778 commit 7b5cabcb0d45e47a0195f693b233315a4096e399  
2779 Author: rgordon <rcgordon@umass.edu>  
2780 Date: Sat Nov 14 12:01:01 2015 -0500  
2781  
2782 fixed string literal parsing  
2783  
2784 commit 6bed058244e6c677d7348cbfdb9387bcfb461e75  
2785 Author: Adam Incera <aji2112@columbia.edu>  
2786 Date: Sat Nov 14 03:49:29 2015 -0500  
2787  
2788 fixed subtract! was actually just yet another bug with remove(). also  
deleted .swp file that snuck in the last commit.  
2789  
2790 commit 88045d86d7d288a5d123d1f15749a34011fe5918  
2791 Author: Adam Incera <aji2112@columbia.edu>  
2792 Date: Sat Nov 14 03:39:53 2015 -0500  
2793  
2794 accidentally committed .o files and executable :(  
2795  
2796 commit debb0afcd7f243fa0db73a8b88d138098a123f58  
2797 Author: Adam Incera <aji2112@columbia.edu>  
2798 Date: Sat Nov 14 03:34:01 2015 -0500  
2799  
2800 c library! all functions seem to be running without memory leaks except  
subtract  
2801  
2802 commit 3bc1b163c9ce6093e95e2b3d5ba74b58517416d0  
2803 Author: rgordon <rcgordon@umass.edu>  
2804 Date: Fri Nov 13 18:14:07 2015 -0500  
2805  
2806 removed vars field from program type  
2807  
2808 commit fded5410d7d3f3cbf97c309c46cc60cc8d296985  
2809 Author: rgordon <rcgordon@umass.edu>  
2810 Date: Fri Nov 13 18:00:39 2015 -0500  
2811  
2812 altered variable declaration so that it's now just another kind of  
statement. added types to ast.ml to support this change. fixes to  
pretty printer  
2813  
2814 commit 12f44aed7f27792e86b30e0bea79470c5e0911d74  
2815 Merge: 08b8e03 0c28e1b  
2816 Author: rgordon <rcgordon@umass.edu>  
2817 Date: Fri Nov 13 14:52:29 2015 -0500  
2818  
2819 Merge branch 'master' into parser  
2820  
2821 integrating updates to lang-ref-man from master branch into parser branch  
2822  
2823 commit 0c28e1bdfd2eb9bcfc74e051dd183465502cfbcd  
2824 Author: rgordon <rcgordon@umass.edu>

2825 Date: Fri Nov 13 14:50:38 2015 -0500  
2826  
2827 re-added the list and dict section to the lang ref manual. PLEASE don't  
overwrite/delete these changes again. --> proof of the need to double  
check git diff before committing  
2828  
2829 commit 08b8e03718fae7421ee8b01c7bbb0b3beba84e0f  
2830 Author: Hosanna <miramonte23@gmail.com>  
2831 Date: Thu Nov 12 23:42:28 2015 -0500  
2832  
2833 pretty printer with types  
2834  
2835 commit 2a6e5d26ddc27ac9be159f3c68cb6a232ac3c0d3  
2836 Author: Hosanna <miramonte23@gmail.com>  
2837 Date: Thu Nov 12 12:52:49 2015 -0500  
2838  
2839 commenting the parser so we know what we need tuples of  
2840  
2841 commit 20f46314c2d4fbda1401438678da20d4a406eae2  
2842 Author: rgordon <rcgordon@umass.edu>  
2843 Date: Wed Nov 11 11:54:38 2015 -0500  
2844  
2845 fixed Ast.expr / string error. fixed reduce / reduce conflict. can now feed  
dots files to ./dotc  
2846  
2847 commit 6785a4f2cc625656664f9b805c9428835db4c79e  
2848 Author: Hosanna <miramonte23@gmail.com>  
2849 Date: Mon Nov 9 19:29:22 2015 -0500  
2850  
2851 added up to date compilation efforts  
2852  
2853 commit ec029757f4f8f90110182fe4ec29fe82c063f583  
2854 Author: Hosanna <miramonte23@gmail.com>  
2855 Date: Mon Nov 9 18:02:35 2015 -0500  
2856  
2857 compiler  
2858  
2859 commit b53b098c17fb51acdcbf4b5b7dc85f4487461d36  
2860 Merge: 921f9b0 cf233e0  
2861 Author: rgordon <rcgordon@umass.edu>  
2862 Date: Fri Nov 6 11:54:49 2015 -0500  
2863  
2864 Merge branch 'parser' of <https://github.com/adamincera/dots> into parser  
2865  
2866 merge  
2867  
2868 commit 921f9b06078ebddd7847cc72db32f6d1745ff4a9  
2869 Author: rgordon <rcgordon@umass.edu>  
2870 Date: Fri Nov 6 11:54:36 2015 -0500  
2871  
2872 added an analyzer.ml file to act as our compiler file. fixed some syntax  
errors in Ast.ml.  
2873  
2874 commit cf233e03b463ffad062fd243462b92c26bdcf470

```
2875 Author: Hosanna <miramonte23@gmail.com>
2876 Date: Wed Nov 4 14:36:04 2015 -0500
2877
2878     added a baseline interpreter
2879
2880 commit bf58fd6c1e116697951f3bea33880f1daf0afdab
2881 Merge: c7de03b f94390c
2882 Author: Hosanna <miramonte23@gmail.com>
2883 Date: Wed Nov 4 14:29:18 2015 -0500
2884
2885     Merge branch 'parser' of https://github.com/adamincera/dots into parser
2886
2887 commit c7de03b76bdc1b0d029249c8f6db856386d6f56f
2888 Author: Hosanna <miramonte23@gmail.com>
2889 Date: Wed Nov 4 14:29:00 2015 -0500
2890
2891     added microC version of our code
2892
2893 commit f94390c2063798a81039588751208dcf44bfed93
2894 Author: Yumeng Liao <yl2908@columbia.edu>
2895 Date: Mon Nov 2 22:34:44 2015 -0500
2896
2897     added method to write tests that should fail and print that it should have
        failed but passed
2898
2899 commit df650eeddb0244037cb4291fa1bed7edeb8406e7
2900 Author: Yumeng Liao <yl2908@columbia.edu>
2901 Date: Mon Nov 2 21:57:54 2015 -0500
2902
2903     worked out a few list and dict tests
2904
2905 commit 29021b3f4c78356aa75b0634681295904022e597
2906 Merge: a2de0d7 aa14b69
2907 Author: rgordon <rcgordon@umass.edu>
2908 Date: Mon Nov 2 21:50:46 2015 -0500
2909
2910     Merge branch 'parser' of https://github.com/adamincera/dots into parser
2911
2912     merge
2913
2914 commit a2de0d72728744b73412f9edbea98de147490761
2915 Author: rgordon <rcgordon@umass.edu>
2916 Date: Mon Nov 2 21:50:32 2015 -0500
2917
2918     changed list decl assignment to require a actuals_list instead of
        formals_list, meaning that expressions can be assigned to lists.
2919
2920 commit aa14b69aac7dae922f6596b5759476ef76ae8edb
2921 Merge: 683a340 a6549cc
2922 Author: Yumeng Liao <yl2908@columbia.edu>
2923 Date: Mon Nov 2 20:40:38 2015 -0500
2924
2925     Merge branch 'parser' of https://github.com/adamincera/dots into parser
2926
```

```
2927 commit 683a340ab4030fdd155d22942f3315047dbe4f42
2928 Author: Yumeng Liao <yl2908@columbia.edu>
2929 Date: Mon Nov 2 20:39:50 2015 -0500
2930
2931     Tried to keep tests informative but ran into some issues with tokens, talk
        to Rachel
2932
2933 commit a6549cce9f339bd298a8871848ee952cd460b7c4
2934 Author: rgordon <rcgordon@umass.edu>
2935 Date: Mon Nov 2 18:42:35 2015 -0500
2936
2937     allowed key of assigned dict declaration to be a literal
2938
2939 commit 0ef877da379e979c1383a67295798485737b6384
2940 Author: Yumeng Liao <yl2908@columbia.edu>
2941 Date: Mon Nov 2 17:36:24 2015 -0500
2942
2943     improved testing script to use python subprocess and various useful flags,
        documented in test_guide.txt
2944
2945 commit 9bb86de36eaeb5489b2861fb649499bebc1133ae
2946 Author: Yumeng Liao <yl2908@columbia.edu>
2947 Date: Mon Nov 2 15:01:05 2015 -0500
2948
2949     Moved test scripts to upper directory
2950
2951 commit 673062fe62cacc663258855efa1333225e605698
2952 Merge: 3c53ac7 e97c03c
2953 Author: Hosanna <miramonte23@gmail.com>
2954 Date: Mon Nov 2 11:21:16 2015 -0500
2955
2956     Merge branch 'parser' of https://github.com/adamincera/dots into parser
2957
2958 commit 3c53ac74db4e3538b503f54f1bd3e7398ec5bdb0
2959 Author: Hosanna <miramonte23@gmail.com>
2960 Date: Mon Nov 2 11:20:52 2015 -0500
2961
2962     compiler and the C code equiv to byte code
2963
2964 commit e97c03cebeae4a5378c98f44306537a2e322faac
2965 Author: rgordon <rcgordon@umass.edu>
2966 Date: Mon Nov 2 10:52:29 2015 -0500
2967
2968     added syntax for && and || symbols. fixed FOR and WHILE loop syntax
2969
2970 commit 1492a8856137a1f8ac5a79e9edb07716c1a8ecad
2971 Author: rgordon <rcgordon@umass.edu>
2972 Date: Mon Nov 2 10:24:28 2015 -0500
2973
2974     fixed syntax of FOR and WHILE loops. addressed all todo comments. added
        rule comments to the expr definition in ast.ml. fixed dict declaration
        assignment (now accepts id { id : expr, id : expr.... } etc.
2975
2976 commit 7059eccd39eab9075dadd63ea8b7e1e990abb428
```

```
2977 Merge: df03698 bb0ebb4
2978 Author: Hosanna <miramonte23@gmail.com>
2979 Date: Mon Nov 2 10:16:07 2015 -0500
2980
2981 Merge branch 'parser' of https://github.com/adamincera/dots into parser
2982
2983 commit bb0ebb49bc4701ab9e58d6973f954245d59c232a
2984 Author: Yumeng Liao <yl2908@columbia.edu>
2985 Date: Mon Nov 2 02:45:15 2015 -0500
2986
2987 after setting up VM and testing, fixed so testing system actually works now
    ...
2988
2989 commit 16930f181d84a284adf3f633f9724c661cc9d663
2990 Author: Yumeng Liao <y.liao.2908@gmail.com>
2991 Date: Sun Nov 1 23:39:58 2015 -0500
2992
2993 Wrote python shell script maker that combines tests from all 4 txt files
    into tests.sh to be run. See instructions inside .txt for acceptable
    input. Could possibly be used in other tests later.
2994
2995 commit 4ed51c78e3607f83709cad061a50a1f06d7023d8
2996 Merge: fa153c1 b7d616d
2997 Author: Yumeng Liao <y.liao.2908@gmail.com>
2998 Date: Sun Nov 1 22:58:16 2015 -0500
2999
3000 Merge branch 'parser' of https://github.com/adamincera/dots into parser
3001
3002 commit fa153c19b5d438d0bd976c0775bae79bf431a79a
3003 Merge: bce2ffe e9cea73
3004 Author: Yumeng Liao <y.liao.2908@gmail.com>
3005 Date: Sun Nov 1 22:57:39 2015 -0500
3006
3007 Merge branch 'master' into parser
3008
3009 commit b7d616d2a94bf2952f6c8fe11b1567a447b5c1b7
3010 Author: rgordon <rcgordon@umass.edu>
3011 Date: Sat Oct 31 11:52:23 2015 -0400
3012
3013 fixed dict and list declarations to require a *data_type* inside the type
    indicator (before it required ID's, which is wrong). also added member
    access for lists and dicts to the expr category (i.e. random access for
    dicts and lists via brackets)
3014
3015 commit df03698c0e146fca61e72e9c5e75a87b9f2c74a7
3016 Merge: 9d5cd99 2431ac5
3017 Author: Hosanna <miramonte23@gmail.com>
3018 Date: Wed Oct 28 18:37:27 2015 -0400
3019
3020 Merge branch 'master' of https://github.com/adamincera/dots into parser
3021
3022 commit d455d5b5effabb5502661e7e88ea36886d989659
3023 Merge: ac3904c 2431ac5
3024 Author: rgordon <rcgordon@umass.edu>
```

```
3025 Date: Wed Oct 28 18:32:41 2015 -0400
3026
3027 Merge branch 'master' into parser
3028
3029 merging change from master into parser branch. (updated lrm files)
3030
3031 commit ac3904cccab6634da34eb291323eb1b926b5c187
3032 Author: rgordon <rcgordon@umass.edu>
3033 Date: Tue Oct 27 12:19:40 2015 -0400
3034
3035 added rules for declaring lists and dicts. what goes within [ ] or { } for
assignment still needs to be reworked. also added folder of text files
of token sequences that should be accepted in menhir
3036
3037 commit 9d5cd99922da2b7f196d98b95c03fb88e443a213
3038 Merge: 1dbdcdb fe6a0b7
3039 Author: Hosanna <miramonte23@gmail.com>
3040 Date: Tue Oct 27 11:19:18 2015 -0400
3041
3042 Merge branch 'parser' of https://github.com/adamincera/dots into parser
3043
3044 commit fe6a0b7aad6aa626d13dcb673db2e86498a47dd6
3045 Author: rgordon <rcgordon@umass.edu>
3046 Date: Tue Oct 27 11:18:43 2015 -0400
3047
3048 fixed problem where program was never accepted
3049
3050 commit 2431ac59cefad1ecf2590abe9b2808685a2b6b07
3051 Author: rgordon <rcgordon@umass.edu>
3052 Date: Mon Oct 26 07:54:01 2015 -0400
3053
3054 additional edits
3055
3056 commit e9cea73d337580171ea5ff7f7239d5fe9f2630b8
3057 Author: Adam Incera <aji2112@columbia.edu>
3058 Date: Mon Oct 26 00:24:16 2015 -0400
3059
3060 cleaned up LRM, added logo to cover page
3061
3062 commit bce2ffe2076e070eb3ad9be1c3e210b3f6d0c3a7
3063 Author: rgordon <rcgordon@umass.edu>
3064 Date: Thu Oct 22 14:11:03 2015 -0400
3065
3066 old versions of all files were based on outdated version of microc compiler
. updated each file with new version of microc compiler. added function
for concatenating lists and altered the program rule to use that since
all variable declarations now return lists.
3067
3068 commit 77d47745262d1f0e1b719eelf5847894d4616631
3069 Author: rgordon <rcgordon@umass.edu>
3070 Date: Thu Oct 22 10:18:19 2015 -0400
3071
3072 changed definition of program symbol to be an object with a variable list,
function list, and command list (aka stmt list). added the '.' token to
```



```
the scanner and added rules for called a member variable or member
function of an object to the expr rule
3073
3074 commit 327b31c16e686fd804abab45b788ef19f990638e
3075 Author: rgordon <rcgordon@umass.edu>
3076 Date: Thu Oct 22 09:26:19 2015 -0400
3077
3078 previous fix forgot to change the patterns of the prefix definitions to
refer to itself --> corrected. also fixed chaining of graph
declarations
3079
3080 commit 1d0ac252d61979690ee196f67fef2e1bad7868bb
3081 Author: rgordon <rcgordon@umass.edu>
3082 Date: Thu Oct 22 09:23:40 2015 -0400
3083
3084 fixed chaining declarations of primitives and nodes
3085
3086 commit 53c49b72410109ceef3ed6cba051910106764156
3087 Author: rgordon <rcgordon@umass.edu>
3088 Date: Thu Oct 22 00:11:50 2015 -0400
3089
3090 minor edits
3091
3092 commit 1dbdcdbd96053cbe89328ace904feaalf2b99efa5
3093 Author: rgordon <rcgordon@umass.edu>
3094 Date: Thu Oct 22 00:01:26 2015 -0400
3095
3096 minor edits
3097
3098 commit 817cc418b1994be088fffc9c3c3403bed4c1d220
3099 Merge: 121d493 246b3a0
3100 Author: Adam Incera <aji2112@columbia.edu>
3101 Date: Wed Oct 21 23:50:33 2015 -0400
3102
3103 merged, removed pdf from repo
3104
3105 commit 121d493e3496111690be4d8bc2e34326e846aaaf
3106 Author: Adam Incera <aji2112@columbia.edu>
3107 Date: Wed Oct 21 23:48:24 2015 -0400
3108
3109 added section 1 and section 2.2 to LRM
3110
3111 commit 246b3a06f87df1641c25b79bf3cae23f5c00e4e7
3112 Merge: fdlfbff e965c02
3113 Author: Hosanna <miramonte23@gmail.com>
3114 Date: Wed Oct 21 23:43:02 2015 -0400
3115
3116 Merge branch 'master' of https://github.com/adamincera/dots
3117
3118 commit fdlfbff6b0b33aa68f41e1e4e25458c39bffb823
3119 Author: Hosanna <miramonte23@gmail.com>
3120 Date: Wed Oct 21 23:42:42 2015 -0400
3121
3122 hosanna
```

```
3123     's contriubtion
3124
3125 commit e965c028d19b707bcd5f42b5f980c2e3daafc77e
3126 Author: rgordon <rcgordon@umass.edu>
3127 Date: Wed Oct 21 23:41:38 2015 -0400
3128
3129     added built-in function subsection
3130
3131 commit 3ffcd8ad51551cf2f850a5eecef59465a7bc66be
3132 Author: rgordon <rcgordon@umass.edu>
3133 Date: Wed Oct 21 22:10:10 2015 -0400
3134
3135     slight change to name of rule
3136
3137 commit 61a87f635d17ab5ff8768ffeba5d610c2917df7f
3138 Author: rgordon <rcgordon@umass.edu>
3139 Date: Wed Oct 21 21:38:32 2015 -0400
3140
3141     additional changes; added rules for edge operations
3142
3143 commit 5db14b42a4ece367a051e8954093572d0b4615fb
3144 Author: Yumeng Liao <y.liao.2908@gmail.com>
3145 Date: Wed Oct 21 21:07:25 2015 -0400
3146
3147     added right graph
3148
3149 commit d716ac0736f352a4b71944f6a7c4f84a7d851022
3150 Author: rgordon <rcgordon@umass.edu>
3151 Date: Wed Oct 21 21:01:08 2015 -0400
3152
3153     1st attempts at making scanner and parser for our language. for testing
3154     whether the grammar is unambiguous
3155
3156 commit 7f980d206d07c1f0bc6faf1bce6684efc3cdefa4
3157 Author: Yumeng Liao <y.liao.2908@gmail.com>
3158 Date: Mon Oct 19 21:25:19 2015 -0400
3159
3160     Finished section on statements.
3161
3162 commit 7f2ee8c9ae79ab53a71202161deb2e4f42031389
3163 Author: Yumeng Liao <y.liao.2908@gmail.com>
3164 Date: Mon Oct 19 20:18:38 2015 -0400
3165
3166     added whitespace section, small tweaks to scoping
3167
3168 commit 3556afc508aa4b733ff08cee243b4416113619d3
3169 Author: rgordon <rcgordon@umass.edu>
3170 Date: Mon Oct 19 00:54:40 2015 -0400
3171
3172     added sections for list and dict data types
3173
3174 commit 4c462efa38de3182801882aca408fd799056d35d
3175 Author: Yumeng Liao <y.liao.2908@gmail.com>
3176 Date: Sun Oct 18 17:39:43 2015 -0400
```

3176  
3177 Tweaked "scope" more to my liking with a more comprehensive definition.  
3178  
3179 commit c51c53eaf772bbd049e18b07d37c5d1f13425751  
3180 Author: Yumeng Liao <y.liao.2908@gmail.com>  
3181 Date: Sun Oct 18 17:15:25 2015 -0400  
3182  
3183 added scope section in Latex  
3184  
3185 commit ed5c07efdb001cd3c6d81faec72fcb74d8b10799  
3186 Author: rgordon <rcgordon@umass.edu>  
3187 Date: Sun Oct 18 14:54:50 2015 -0400  
3188  
3189 created initial latex file for the language reference manual, including  
adding necessary headers and package imports. created Makefile for  
easily compiling and cleaning up the tex file. added primitive type,  
functions, and program structure sections to the doc.  
3190  
3191 commit 0846e4b9748002a74f1bbc7219ec59d98c88cbfe  
3192 Author: Adam Incera <aji2112@columbia.edu>  
3193 Date: Thu Oct 8 20:19:31 2015 -0400  
3194  
3195 moved proposal into documentation directory, added latex compilation script  
3196  
3197 commit a6b182e60d3366ee7cfd9f20ded2dc340a3422be  
3198 Merge: 36f25c8 0e4350a  
3199 Author: Hosanna <miramonte23@gmail.com>  
3200 Date: Thu Oct 8 20:00:13 2015 -0400  
3201  
3202 hosanna was here  
3203  
3204 commit 36f25c8db74c8497ce0cc4774dd616209d4b4c3f  
3205 Author: Hosanna <miramonte23@gmail.com>  
3206 Date: Thu Oct 8 19:59:02 2015 -0400  
3207  
3208 hosanna was here  
3209  
3210 commit 8fadefb68aa67f9d9ef976b3074a51794415285e  
3211 Author: rgord <rcg2130@columbia.edu>  
3212 Date: Fri Oct 9 07:46:04 2015 -0400  
3213  
3214 transferred latex project files for lang proposal from sharelatex to repo  
3215  
3216 commit 0e4350a21eb3b9eafe27dde4b3b3811bdf21cec9  
3217 Author: Yumeng Liao <y.liao.2908@gmail.com>  
3218 Date: Thu Oct 8 19:57:30 2015 -0400  
3219  
3220 testing  
3221  
3222 commit 2306aed112353a1dd50308f26cdd6c83de7cbbff  
3223 Author: Adam Incera <aji2112@columbia.edu>  
3224 Date: Sun Sep 20 21:34:23 2015 -0400  
3225

Listing 41: git log