giraph
a language for manipulating graphs

<table>
<thead>
<tr>
<th>Jessie Liu</th>
<th>Seth Benjamin</th>
<th>Daniel Benett</th>
<th>Jennifer Bi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
<td>Language Guru</td>
<td>System Architect</td>
<td>Tester</td>
</tr>
<tr>
<td>j112219</td>
<td>sjb2190</td>
<td>deb2174</td>
<td>jb3495</td>
</tr>
</tbody>
</table>

Fall 2017
## Contents

1 Introduction 3
  1.1 Motivation ........................................ 3

2 Language Tutorial 4
  2.1 Basic Syntax ....................................... 4
  2.2 Graphs, Nodes, Edges ............................... 4
  2.3 Graph Iterators .................................... 6
  2.4 Generics in Maps and Graphs ...................... 8

3 Language Manual 9
  3.1 Lexical Conventions .................................. 9
  3.2 Data Types ........................................... 9
    3.2.1 Primitive types .................................. 9
    3.2.2 Graph types ..................................... 9
    3.2.3 Maps ............................................. 10
  3.3 Operators and Expressions ......................... 10
    3.3.1 Variable Assignment ............................. 10
    3.3.2 Arithmetic Operators ............................ 10
    3.3.3 Logical and Relational Operators ............ 10
    3.3.4 Graph construction ............................ 10
    3.3.5 Methods .......................................... 11
  3.4 Control Flow ........................................ 12
    3.4.1 Conditionals ..................................... 12
    3.4.2 Loops ............................................ 13
  3.5 Scoping .............................................. 13
  3.6 Program Structure ................................... 13

4 Project Plan 14
  4.1 Work Process ........................................ 14
  4.2 Software Development Environment ..................... 14
  4.3 Programming Style Guide ............................. 14
  4.4 Project Timeline ..................................... 14
  4.5 Team Roles and Responsibilities .................... 15
  4.6 Project Log .......................................... 15

5 Architectural Design 16
  5.1 Components ......................................... 16
  5.2 Interfaces between Components ...................... 16
    5.2.1 Lexer (all contributed) .......................... 16
    5.2.2 Parser (Seth, Jennifer, Jessie) ............... 16
    5.2.3 Semantic checking (Jessie, Jennifer, Seth) .... 16
    5.2.4 Code generation (Seth, Daniel) ................ 16
    5.2.5 C Libraries (Daniel, Seth) .................... 17

6 Test Plan 18
  6.1 Unit testing ......................................... 18
  6.2 Regression testing ................................. 18
7 Lessons Learned
7.1 Danny ................................................................. 20
7.2 Jennifer .............................................................. 20
7.3 Jessie ................................................................. 20
7.4 Seth ................................................................. 20

8 Appendix
8.1 Scanner .............................................................. 22
8.2 Parser ............................................................... 23
8.3 Semantic Checking .................................................. 31
8.4 Code Generation .................................................... 60
8.5 C Libraries .......................................................... 80
8.6 Testing ............................................................... 104
8.7 Example code ....................................................... 108
8.8 Project Log .......................................................... 109
1 | Introduction

1.1 Motivation

Graphs are a fundamental method of representing the world we live in. Though graphs are composed of simple nodes and edges, they are powerful enough to be applied to real areas like economics and chemistry, with applications ranging from airline scheduling to linguistic modeling. However, graph creation and algorithms in common programming languages can be verbose and tedious to write. *giraph* is a programming language with the goal of simplifying the creation of graphs and the implementation of graph algorithms.
2.1 Basic Syntax

Our program uses C-like syntax. Executables run from their main function. All functions must have a return. This example program increments a counter 10 times and then prints the result:

```c
void main () {
    int counter = 0;
    while (counter < 10) {
        counter = counter + 1;
    }
    print (counter);
    return;
}
```

2.2 Graphs, Nodes, Edges

There are 4 graph types: graphs, digraphs, wegraphs, and wedigraphs.

<table>
<thead>
<tr>
<th></th>
<th>weighted?</th>
<th>directed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>wegraph</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>digraph</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>wedigraph</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Consider the following code snippet:

```c
graph<int> g = [A:1 -- B:2 -- C:3 ; D:4 -- A];
```

This creates a graph (unweighted and undirected) with the following structure, initializing nodes A, B, C, and D with int data values 1, 2, 3, and 4 (respectively):

To create a wegraph, use the following syntax:

```c
wegraph<int> g = [A:1 -{50} - B:2 -{8} - C:3 ; D:4 -{10} - A];
```

This creates a graph with the following structure:
To create a digraph, use the following syntax:

```
1 digraph<int> g = [A:1 -> B:2 -> C:3 ; D:4 <- A];
```

This creates a graph with the following structure:

```
A
  \-- B
     \- C
        \- D
```

To create a wedigraph, use the following syntax:

```
1 wedigraph<int> g = [A:1 -{50} - > B:2 -{8} - > C:3 ; D:4 <-{10} - A];
```

This creates a graph with the following structure:

```
A
  \-- B
     \- C
        \- D
```

In each of the above cases, the four nodes A, B, C, D are defined as part of the inline graph definition. It is also possible, however, to create nodes and set their data outside of graph definitions, as in this example:
After line 3, graph ‘g’ contains a single node ‘n’ with integer data 1.

If you assign a previously declared and assigned node in a graph definition, its old data will be overwritten, as in this example:

After line 3, graph ‘g’ contains a single node ‘n’ with integer data 5

Multiple graphs can share the same nodes. When a node’s data is updated, this update occurs across all graphs. Consider the following example:

After line 3, in both ‘g’ and ‘g_subgraph’, node A will have data 5.

It is also possible to add edges and set edge weights outside of the initial graph definition, as in the following snippet:

This creates a graph with the following structure:

2.3 Graph Iterators

Consider example:
void main () {
    int counter = 0;
    graph <int > g = [A:1 -- B:2 -- C:3];
    for_node(n : g) {
        counter = counter + n.data();
    }
    print(counter);
    return;
}

The above program will print 6. It iterates through each node in graph g, assigning each to variable n, and executing the for_node body with each n. The iteration order is consistent across program runs, but is not predictable from the structure of the graph.

for_edge

Consider example:

void main () {
    int counter = 0;
    wegraph <int > g = [A:1 -{1} - B:2 -{2} - C:3];
    for_edge(e : g) {
        counter = counter + e.weight();
    }
    print(counter);
    return;
}

The above program will print 3. It iterates through each edge in graph g, assigning each to variable e, and executing the for_edge body with each e. The iteration order is consistent across program runs, but is not predictable from the structure of the graph.

bfs

Consider example:

void main () {
    int counter = 0;
    digraph <int > g = [A:1 -> B:2 -> C:3];
    bfs(n : g ; B) {
        counter = counter + n.data();
    }
    print(counter);
    return;
}

The above program will print 5. It visits through each node in graph g in bfs order starting from node B, assigning each to variable n, and executing the bfs body with each n. In this example, node A is never visited.

dfs

The dfs iterator works just as the bfs iterator above, but visits nodes in dfs order.
2.4 Generics in Maps and Graphs

Maps
Maps pair nodes (serving as unique keys) to values. We may use maps to store and access data. Consider example:

```java
node<int> C;
map<int> m;

m.put(C, C.data());
printb(m.contains(C));
```

The program prints true, since the map contains our data.

Graphs
Generics allow for nesting in graphs. Consider example:

```java
graph<string> g = [A:"gir" -- B:"aph"];  
graph<graph<string>> G = [N:g];

for_node(ng : G){  
    for_node(n : ng.data()) {  
        prints(n.data());
    }
}
```

Here, the graph G contains a node which contains a graph (ng.data()), which we access and iterate over using for_node.
3 Language Manual

3.1 Lexical Conventions

Tokens fall into the following categories: identifiers, keywords, literals, expression operators. Whitespace (of any amount) may be used to separate tokens.

- Comments, single-line or multi-line, are indicated by !~ the comment characters ~!
- Identifiers are sequences of alphanumeric characters, including underscore: _. All identifiers must begin with an alphabetic character. Identifiers may not be the same as reserved keywords.
- Reserved keywords are

```
bool int float string
graph node wegraph digraph
wedigraph map void return
if then else for
while for_node for_edge bfs
dfs true false
```

3.2 Data Types

`giraph` is a statically typed language, and supports the following:

3.2.1 Primitive types

- `bool` - Boolean data, which can be `true` or `false`.
- `int` - Ints are signed 8-byte literals. Represents a number as a sequence of digits.
- `float` - Floats are signed single-precision floating point numbers.
- `string` - A sequence of ASCII characters. Literals are enclosed in double quotes: `string s = "graphs are cool"`

3.2.2 Graph types

A graph consists of nodes and optionally edges. The following distinct subtypes of `graph` are supported, with the following hierarchical structure:

- `graph` - A graph whose edges are unweighted and undirected
- `digraph` - A graph whose edges are directed
- `wegraph` - A graph whose edges are weighted with positive or negative integer weights
- `wedigraph` - A graph whose edges are directed and weighted with positive or negative weights

Graphs and nodes are generically implemented in `giraph`. The type of data the nodes in a graph can store may be any of the data types in the language. A graph can only have a single type, which must be specified (e.g. `digraph<int>` is the type of a directed graph whose nodes store ints). A graph’s data type can also be a generic type (e.g. `graph<graph<int>>`).
3.2.3 Maps
A map stores key-value pairs keyed on nodes. Maps are also generically implemented: keys are always nodes, but values can be any type (including generic types). Like graphs, map can only store a single type of value, which must be specified (e.g. \texttt{map<int>}). However, the keys of the map can be nodes with any type of data. For example, you can put a \texttt{node<string>} and a \texttt{node<int>} as keys in the same map.

3.3 Operators and Expressions

3.3.1 Variable Assignment
Variables are assigned using the $=$ operator. The left hand side must be an identifier while the right hand side must be a value or another identifier. The LHS and RHS must have the same type, as conversions or promotions are not supported. The variable assignment operator groups right-to-left.

3.3.2 Arithmetic Operators
Arithmetic operator precedence is as in standard PEMDAS. Operator binding is as follows:

Additive operators $+$, $-$ group left-to-right.
Multiplicative operators $\ast$, $\backslash$, $\%$ group left-to-right.
Parentheses have the highest precedence, and therefore can be used to override the default operator precedence.

3.3.3 Logical and Relational Operators
Relational operators ($<$, $>$, $\leq$, $\geq$) and logical operators ($\&\&$, $||$) also group left-to-right. So, a statement like $a < b \&\& b < c \&\& c < d$ can simply be accomplished with $a < b < c < d$. Expressions with relational and or logical operators return 0 or 1 for true or false respectively.

Equality operators also group left-to-right, but have lower precedence than relational ones. Logical operators have the lowest precedence of the three.

3.3.4 Graph construction
The subtype of graph and type of its nodes' data are designated when a new graph is declared. A graph can be initialized with a graph literal, which is enclosed in square brackets ([ ]). A graph literal consists of nodes, edges, and node initializations.

New nodes may be declared and optionally initialized in graphs by including a new identifier in a graph literal; this identifier becomes visible in the most local scoped block. Existing nodes may be added to graphs by referencing their identifier. A node may be referenced (but not initialized) multiple times within one graph construction.

Edges between nodes are defined through intuitive edge tokens, the available edges being undirected (\texttt{-{-}}), directed (\texttt{->}, \texttt{<-}, \texttt{<->}), weighted undirected (\texttt{-{0}--}), weighted directed (\texttt{-{0}->}). The type of edge corresponds to the subtype of graph and must be consistent across all edges in the graph expression. A graph literal without edges can be assigned to any of the graph subtypes.

Separate parts of the same graph can written in the same literal by separating them with semicolons; the final graph is the union of all parts. This can be useful for including separate components that are
not connected by edges in the same graph.

Nodes can be initialized with data using the : operator. A node identifier followed by a colon followed by an expression in a graph literal initializes the data in the node to be the value of the expression. The expression type for any node in a graph literal must match the data type of the graph it is being assigned to.

A graph literal must either contain at least one previously-declared node, or at least one initialized node.

For example, the following declarations/initializations are all separately valid:

```cpp
node<int> n;  // a node declaration, with no data initialized
graph<int> g = [A:1 -- B:2 -- C:3 -- A ; D:4 -- A ; E:5];
digraph<float> g = [A:1.0 <-> B:2.0 ; E:5.0 <-> A];
digraph<bool> g = [];  // creates an empty graph with no nodes
wegraph<string> g = [A:"hi" -{1}- B:"there"];
wedigraph<int> g = [A:1 -{1}-> B:2 <-{2}-> C:3 <-{3}-> D:4];
```

### 3.3.5 Methods

Methods can be invoked as follows.

```cpp
id . method()
```

Methods are the primary mode of working with graphs and the nodes/edges belonging to a graph. While a node may exist independently of graphs, construction as part of a graph is preferable. Edges may not be initialized independently of graphs, as they represent a relation between two nodes in a graph. Thus, nodes and edges are manipulated through built-in methods.

- **node<t>** - Nodes hold data within a graph and this data may be accessed and modified.
  - node.data() - returns data stored in node
  - node.set_data(t data) - stores data in node

- **edge** - An edge connects two nodes within a graph. While iterating over the edges in a graph with the for_edge iterator, the following methods can be called on an edge.
  - edge.from() - returns node edge is coming from. In an undirected graph, edge.from() != edge.to(), except in a self-loop. Beyond that, there is no guarantee as to which node will be returned by edge.from()
  - edge.to() - returns node edge is going to. In an undirected graph, edge.from() != edge.to(), except in a self-loop. Beyond that, there is no guarantee as to which node will be returned by edge.to()

In wegraphs and wedigraphs, the following methods may also be called on edges during for_edge iteration:
- * edge.weight() - returns weight of edge
- * edge.set_weight(int i) - sets weight of edge to int i.

- **graph<t>** and all subtypes - graph methods are as follows.
  - graph.add_node(node<t> node) - add node to graph. If node is already in graph, does nothing.
- `graph.add_edge(node<t> from_node, node<t> to_node)` - add edge between `from_node` and `to_node` to `graph`. If either of `from_node` or `to_node` are not already in `graph`, they are added, and then connected by an edge.

- `graph.remove_node(node<t> node)` - remove node from `graph`. If `node` is not already in `graph`, does nothing.

- `graph.remove_edge(node<t> from_node, node<t> to_node)` - remove edge between `from_node` and `to_node` from `graph`. If there is not an edge between `from_node` or `to_node` in `graph`, or if either node is not in the graph, does nothing.

- `graph.has_node(node<t> node)` - returns true if `graph` has node `node`, false otherwise.

- `graph.has_edge(node<t> from_node, node<t> to_node)` - returns true if `graph` has an edge from `from_node` to `to_node`, false otherwise.

- `graph.neighbors(node<t> node)` - returns edgeless graph containing all neighbors of `node` as nodes.

- `graph.print()` - prints `graph` nodes, node data, and adjacencies.

- `wegraph<t>` and `wegraph<t>` - in addition to the above, additional weighted graph methods are as follows.

  - `graph.add_edge(node<t> from_node, node<t> to_node, int weight)` - add edge with weight `weight` between `from_node` and `to_node` to `graph`. If either of `from_node` or `to_node` are not already in `graph`, they are added, and then connected by an edge. This method replaces the 2-argument `add_edge` listed above.

  - `graph.get_edge_weight(node<t> from_node, node<t> to_node)` - returns weight of the edge between `from_node` and `to_node` in `graph`.

  - `graph.set_edge_weight(node<t> from_node, node<t> to_node, int i)` - sets weight of the edge between `from_node` and `to_node` in `graph` to `i`

- `map<t>` - map methods are as follows.

  - `map.put(node<?> key, t val)` - add `key` to map with associated value `val`. If `key` is already in map, replace its old associated value with `val`. `key` can be a node of any type.

  - `map.get(t key)` - get the value associated with `key` from the map.

  - `map.contains(t key)` - returns true if `map` contains `key` and false if it does not. It is best to call this prior to get in order to insure that the map contains the desired `key`.

### 3.4 Control Flow

#### 3.4.1 Conditionals

If-else statements are allowed, in the following formats:

```plaintext
if (condition) {statements}
if (condition) {statements} else {statements}
if (condition) {statements} else if {statements} else {statements}
```

The else block is optional for any if statement, and any number of else if's can be appended to any if statement.
3.4.2 Loops

C-style while loops and for loops are provided, such as the following:

```
while (condition) {statements}
for (initialization; condition; update) {statements}
```

They can either be followed by a single statement to be looped, or by a sequence of statements enclosed within brackets. Graph-specific iteration over nodes and edges is also allowed, using "for each" loops, which take the following format:

```
for_node(node : graph) {statements}
for_edge(edge : graph) {statements}
```

These iterate over all the nodes/edges of graph respectively, executing the looped statements at every node/edge. For example, the following loop can be used to print the data at every node in some graph g:

```
for_node(n : g) {print(n.data());}
```

Each of for_node and for_edge iterate over their respective graph components in an unspecified order. However, one can also iterate over graph components in a specific order using the following loop constructions:

```
bfs(node : graph ; root) {statements}
dfs(node : graph ; root) {statements}
```

These iterate over the nodes of a graph using breadth-first search and depth-first search respectively, starting at root and executing the looped statements at every subsequently reached node.

3.5 Scoping

A variable exists within the scope in which it is declared. This scope could be global (outside of any function) or local, within a function, if block, then block, else block, any loop, or a plain scope delimited by braces: { !~this is a scope~! }.

3.6 Program Structure

Programs in giraph consist of a list of declarations which includes either variable or function declarations. The declarations must include a main() function which is the entry point of a compiled executable giraph program. Functions are defined with the following signatures:

```
return_type function_name(type arg, type arg, ...) {body}
```

Program execution begins in the main() method, and the program exits upon arriving at its return statement.
4 | Project Plan

4.1 Work Process

Our team met to work on the project around once a week at the beginning of the semester, and almost every day the last month. We met with our TA Lizzie before major deadlines (getting the LRM, hello world) along with a few other check ins to run ideas past her, discuss issues, and get encouragement. Our main form of communication was Facebook Messenger, over which we communicated almost every day and pretty much every hour in the final stages of the project. We also would meet to work together in groups, frequently drinking green tea in EC or cappuccinos in the CS lounge as we pushed code late into the night.

4.2 Software Development Environment

To build giraph, we used these languages and development tools:

- OCaml version 4.05.0: for scanning, parsing, and semantic checking
- C: for building graphs
- Makefile: for compiling and linking
- Git and Github for version control and hosting our git repository, respectively
- Bash Shell scripting: for automating testing

4.3 Programming Style Guide

Our team used a very simple style guide (which doesn’t quite warrant a whole page):

- Spaces, not tabs
- Functions and variables should be named with under_scores, not camelCase
- Indentation should be consistent
- Code should be as clear and concise as possible, with comments added for clarity

4.4 Project Timeline

10/16: Language Reference Manual submitted
10/30: Hello World program compiled
11/17: First parsed graph
12/2: Graphs in Codegen
12/16: Digraphs end to end
12/17: Semantic checking added - sast types integrated into codegen
12/18: Max Flow program works
12/19: Maps added end to end
12/20: Final presentation
4.5 Team Roles and Responsibilities

Though we each technically had an assigned role, boundaries between them faded as we continued to work on the project. We often found ourselves stepping into parts of different roles, picking up slack in places and also allowing other members of the team to help us out when we needed it. We all contributed to components of the language design, compiler, and test suite, and also made a point to check in with each other frequently.

Jessie Liu: Semantic checking, parser, lexer, tests
Seth Benjamin: Code generation, C libraries, parser, semantic checking, lexer, tests
Daniel Benett: C libraries, code generation, lexer, parser, tests
Jennifer Bi: Test suite, semantic checking, parser, lexer

4.6 Project Log

See appendix 8.8 for our project log.
5 | Architectural Design

5.1 Components

5.2 Interfaces between Components

5.2.1 Lexer (all contributed)
The lexer reads in a program string and returns tokens representing the program.

5.2.2 Parser (Seth, Jennifer, Jessie)
The parser reads in tokens from the lexer, and returns an abstract syntax tree. The structure of the tree is determined by the grammar. A program not accepted by the grammar is thus rejected in the parser. In our case, the parser did a lot of heavy-lifting for graph construction. In particular, the parser extracts node names, node initializations, and edge relationships and weights directly from graph literals. Additionally, determining the subtype of a graph literal (weightedness/directedness) is traditionally a type-related check and could have been the responsibility of the semantic checker. However, we found that we could make use of grammar rules to immediately enforce consistency in weightedness/directedness of edges. So, the AST produced by the parser has a subtype by the time it reaches semantic checking. The node initialization types are not checked by the parser.

5.2.3 Semantic checking (Jessie, Jennifer, Seth)
In semantic checking, we perform type checking and enforce static scoping. Our semantic checker keeps an environment variable that corresponds to each scoped block. The environment includes a symbol table for local variables, a map of visible functions, along with several other fields. The recursive nature of OCaml was conducive to using applicative order evaluation, in which the innermost expression were evaluated first, semantically checked, and its environment updated. As each token of the AST is checked, valid expressions are tagged with types, while anything invalid returns a Failure. These type-tagged expressions are then added to a SAST (semantically-checked AST).

5.2.4 Code generation (Seth, Daniel)
In code generation, we step through the SAST produced after semantic checking and converts it to an LLVM module, which is then outputted. We start at the top level of the SAST (containing a list of globals and a list of functions) and unpacking the heavily nested structure down to the terminals.
First, the globals are allocated and added to a symbol table, mapping their identifiers to their allocated registers. For each function, the formals are first processed and added to this symbol table. Then, we enter the body of the function, which is a scoped block. Whenever we enter a scoped block, we create a new symbol table, containing the outer block’s variables, as well as allocating and mapping any new variables declared within the block. Then, all of the statements in the block are mapped to LLVM instructions which are outputted in the module. The semantic type information included in the SAST is heavily used in codegen, especially with graphs. Given that the different graph types all have fairly similar user-facing functionalities (e.g. the methods we provide) that have different behaviors (e.g. add_edge(from, to) should add an undirected edge in an undirected graph, but a directed edge in a directed graph), codegen is able to solve this problem by using the type information to determine what instructions to output / what C functions to call. The code generation stage also declares function headers for all of the functions implemented in the C graph library, so that they appear in the outputted LLVM and can be linked from the library’s object files.

5.2.5 C Libraries (Daniel, Seth)

There are a series of C structs used to internally store graphs and maps. These most frequently take linked list form. C functions are called from code generation, taking in void pointers that, when casted and dereferenced, contain the structs and data for graphs and nodes. For some methods, such as graph.has_edge, code generation can simply pattern match on the SAST, call the proper C function, and use its return as necessary. For functionality such as the bfs and dfs iterators, LLVM code and C code are interlaced. Each iteration of the bfs/dfs loop requires predicate testing and assignment of the current node in the graph traversal, which is occurring in the C library, to the node variable, which is needed in the loop body’s statement list. As such, significant coordination between code generation and C library design was required.
6 Test Plan

We tested components separately in the early stages of development when we were still defining the stages of our compiler. Once the architecture was established and once our language had more functionality, regression testing became ideal. Unit testing served as a secondary debugging method.

6.1 Unit testing

- Lexer and parser

  We tested the parser by running programs with the parser trace option `OCAMLRUNPARAM=p` and manually comparing its output with expected output. Also useful for early stage parser testing were the debugging options in Menhir, an alternative LR(1) parser generator, which printed out a parse tree.

  `menhir -interpret -interpret-show-cst parser.mly`

6.2 Regression testing

A test script adapted from MicroC ran tests that should pass and tests that should fail, with a general rule that for each feature tested as a pass case, a corresponding fail case was also tested. This was useful for keeping semantic checking and code generation in sync, that is, at any point in development all code generation functionality would not go unchecked. The regression testing was useful for tracking ripple effects of small changes in semantic checking or codegen. Furthermore, testing provided a sanity check on basic control flow, variable and function declarations, primitive data assignment operations.

- Semantic checking

  Most semantic errors were caught by regression testing. Most of our target programs were simple to isolate certain features. A few additional test programs had more complexity to test that features could work together.

  `fail-graphdecl6.gir`: test to ensure checking against impossible graph declarations were prevented

  ```
  int main () {
    wedigraph g = [A:5 -{5}-> B:6 <-(4)- A];
    return 0;
  }
  
  Fatal error: exception Failure("graph literal cannot feature the same edge with different weights")
  ```

  `fail-bfs1.gir`: test to ensure checking against interference with breadth-first search

  ```
  int main () {
    graph g = [A:1 -- B:2 -- C:3 -- D:4 -- E:5 -- F:6 -- G:7];
    bfs(b : g ; A) {
      g.add_edge(A,b);
    }
    return 0;
  }
  
  Fatal error: exception Failure("concurrent modification of graph in bfs")
  ```

- Code generation

  `test-foredge2.gir`: one of the tests to ensure for_edge functions properly
```c
int main() {
    graph<int> g = [A:1 -- B:2; C:3 -- D:4];
    graph<int> g2 = [];
    for_edge(e : g) {
        g2.add_edge(e.from(), e.to());
    }
    for_edge(e : g2) {
        print(e.from().data());
        print(e.to().data());
    }
    return 0;
}
```

```
out:
2
1
4
3
```

test-neighbors3.gir: tests neighbors method in case where there are no neighbors (nothing should print)

```c
int main() {
    graph<int> g = [A:5];
    graph<int> g2 = g.neighbors(A);
    for_node(n : g2) {
        print(n.data());
    }
    return 0;
}
```

```
out:
```

test-bfs3.gir: tests bfs iteration in disconnected graph

```c
!~ should only print A and B, as C and D are not accessible from bfs root A ~!
int main() {
    graph<int> g = [A:1 -- B:2 ; C:3 -- D:4];
    bfs(b : g ; A) {
        print(b.data());
    }
    return 0;
}
```

```
out:
1
2
```

These tests are indicative of our testing style. test-foredge2.gir is an example of a test that thoroughly tests the core intended functionality of a feature (for-edge). test-neighbors3.gir is an example of a test that tests an edge case (when there are no neighbors). test-bfs3.gir is an example of a test that tests bfs in the specific case of disconnected graphs. We tried to write small, specific tests that thoroughly cover the cases a particular feature could encounter.
7 | Lessons Learned

Working on giraph was an intense learning experience for us in several different ways. Below are some lessons and advice we have for future groups.

7.1 Danny

Set yourself clear goals and then work at them in a quiet room until they have been accomplished. Communicate what you need from everybody else in order to implement your goal end-to-end. Talk out and debate tricky implementations and difficult design decisions. Don’t talk too long: eventually just make a decision. Ask for help when you get stuck. Test frequently and comprehensively. When you feel lazy and don’t want to write more code, bake your team cookies. The physical warmth of the fresh-out-the-oven cookies will combine with the sentimental warmth of the moment and will imbue your language with the sense of joy that all programming languages truly need to succeed. Make sure to continue work after baking and eating cookies.

7.2 Jennifer

I learned to embrace functional programming and forget about pointers and tons of control flow. It was convenient to be able to use variables and apply functions freely without cumbersome type declarations. I learned a lot about version control since it was my first time using github for development. I’m glad that our team communicated and consulted each other frequently—often we would start writing different pieces of code and find ourselves overlapping or thinking about the same problem. This made the process a lot more enjoyable. Testing became habitual bordering on compulsive, which I’m really glad for. Pushing and pulling on github was a lot easier knowing that it compiles and all the cases pass. My advice would be to start early and set more milestones between Hello World and the project due date—building a compiler is pretty much boundless with respect to adding functionality and optimization so you can literally never start too early.

7.3 Jessie

Firstly, I now have the utmost respect for the OCaml language. OCaml is the greatest for doing this kind of compiler building stuff. Scanning and parsing was intuitive given pattern matching, and utilizing OCaml’s natural scoping made semantic checking so so much easier. Don’t try to fight functional programming - I spent a good amount of time in an imperative mindset trying to write OCaml, and it was not fun. Secondly, test often, especially before pushing code. Thirdly, maintaining good communication, relations, and trust with your teammates is absolutely crucial - you want to know they’ll always have your back. And lastly, have fun. This is probably the coolest project I’ve worked on, ever!

7.4 Seth

1. Write design docs. You do not want to end up 70% of the way through the project and realize that your design was fundamentally flawed. You also do not want to waste time arguing with your teammates about your half-baked ideas. For me, writing design docs solved both these problems: it forced me to organize my thoughts coherently and consider edge cases before proposing an idea. It also produces a paper trail: 3 weeks down the line, when everything is breaking, the design doc is there to remind you why you thought x was a good idea.
2. Work incrementally. Get a super dumb and useless compiler working end-to-end first. Add a single new feature, test, repeat. This thing is way too big to be done in any other way.
3. The majority of my contribution to this language was written in a series of fear-driven code-vomiting
sessions between the hours of 2 and 7 am. I don’t recommend my approach per se, but it did the job. If you’re terrified that the project will crash and burn, you’ll go the extra mile to make sure it doesn’t.
8 | Appendix

8.1 Scanner

(* Authors:
Daniel Benett deb2174
Seth Benjamin sjb2190
Jennifer Bi jb3495
Jessie Liu jll2219 *)

(* Ocamllelex scanner for giraph *)

{ open Parser }

(* Definitions *)

let digit = ['0'-'9']
let decimal = ((digit+ ',' digit*) | (',' digit+))
let letter = ['a'-'z', 'A'-'Z']

(* Rules *)

rule token = parse
[ ' ' '	' '' '
' ] { token lexbuf } (* to ignore whitespace *)
| "!-" { comment lexbuf }
| ',,' { COMMA }
| ',.' { DOT }
| ';:' { SEMI }
| ':.' { COLON }
| '\\' { SINGLEQUOTE }
| '\"' { DOUBLEQUOTE }

(* scoping *)
| '(': { LPAREN }
| ')': { RPAREN }
| '{': { LBRACE }
| '}': { RBRACE }
| '[': { LBRACK }
| ']': { RBRACK }

(* keywords *)
| "for" { FOR }
| "while" { WHILE }
| "for_node" {FOR_NODE}
| "for_edge" {FOR_EDGE}
| "bfs" {BFS}
| "dfs" {DFS}
| "if" { IF }
| "then" { THEN }
| "else" { ELSE }
| "bool" { BOOL }
| "float" { FLOAT }
| "int" { INT }
| "string" { STRING }
| "graph" { GRAPH }
| "node" { NODE }
| "wgraph" { WEGRAPH }
| "digraph" { DIGRAPH }
| "wedigraph" { WEDIGRAPH }
8.2 Parser
%{ open Ast
open Prshelper %}

% token LPAREN RPAREN LBRACK RBRACK LBRACE RBRACE DOT COMMA SEMI COLON
% token PLUS MINUS TIMES DIVIDE ASSIGN NOT MOD
% token EQ NEQ LT LEQ GT GEQ AND OR
% token RETURN IF THEN ELSE FOR WHILE FOR_NODE FOR_EDGE BFS DFS BREAK CONTINUE
% token INT BOOL VOID FLOAT STRING NODE EDGE GRAPH WEGRAPH DIGRAPH WEDIGRAPH MAP
% token RARROW LARROW DIARROW
% token SINGLEQUOTE DOUBLEQUOTE
% token <int> INT_LIT
% token <float> FLOAT_LIT
% token <bool> BOOL_LIT
% token <string> STRING_LIT
% token <string> ID
% token EOF

% right SEMI

/* arithmetic ops*/
% right ASSIGN
% left PLUS MINUS
% left TIMES DIVIDE MOD
% left OR AND
% right NOT NEG

% nonassoc EQ NEQ
% nonassoc GEQ LEQ GT LT
% nonassoc NOELSE
% nonassoc ELSE

% left DOT

% right LARROW RARROW DIARROW EDGE
% nonassoc COLON
% start program
% type <Ast . program> program
%
program:
  decls EOF { $1 }
%
dcls:
  /* nothing */
  { [], [] } /* first list has vdecls, second has fdecls*/
  | decls vdecl { ($2 :: fst $1), snd $1 }
  | decls fdecl { fst $1 , ($2 :: snd $1) }

tdecl: typ ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
  { { f_typ = $1; f_name = $2; f_formals = $4; f_body = List . rev $7 } }

typ:
  INT { Int }
  | FLOAT { Float }

Jessie Liu jll2219 */
| BOOL  | { Bool } |
| VOID  | { Void } |
| STRING | { String } |
| NODE LT typ GT | { Node($3) } |
| GRAPH LT typ GT | { Graph($3) } |
| DIGRAPH LT typ GT | { Digraph($3) } |
| WEGRAPH LT typ GT | { Wegraph($3) } |
| WEDIGRAPH LT typ GT | { Wedigraph($3) } |
| MAP LT typ GT | { Map($3) } |

formals_opt: /* nothing */ { [] }
| formal_list | { List . rev $1 } |

formal_list: typ ID { ([$1, $2]) }
| formal_list COMMA typ ID | { ($3,$4) :: $1 } |

vdecl:
| typ ID SEMI | { ($1, $2) } |

stmt_list:
| /* nothing */ | { [] } |
| stmt_list stmt | { $2 :: $1 } |

stmt:
| expr SEMI | { Expr $1 } |
| typ ID SEMI | { Vdecl($1, $2, Noexpr) } |
| typ ID ASSIGN expr SEMI | { Vdecl($1, $2, Assign($2,$4)) } |
| RETURN SEMI | { Return Noexpr } |
| RETURN expr SEMI | { Return $2 } |
| LBRACE stmt_list RBRACE | { Block(List . rev $2) } |
| IF LPAREN expr RPAREN stmt % prec NOELSE | { If($3, $5, Block([])) } |
| IF LPAREN expr expr opt RPAREN stmt ELSE stmt | { If($3, $5, $7) } |
| FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt | { For($3, $5, $7, $9) } |
| FOR_NODE LPAREN ID COLON expr RPAREN stmt | { For_Node($3, $5, $7) } |
| FOR_EDGE LPAREN ID COLON expr RPAREN stmt | { For_Edge($3, $5, $7) } |
| BFS LPAREN ID COLON expr expr SEMI expr RPAREN stmt | { Bfs($3, $5, $7, $9) } |
| DFS LPAREN ID COLON expr expr SEMI expr RPAREN stmt | { Dfs($3, $5, $7, $9) } |
| WHILE LPAREN expr RPAREN stmt | { While($3, $5) } |
| BREAK SEMI | { Break } |
| CONTINUE SEMI | { Continue } |

expr:
| INT_LIT | { Int_Lit($1) } |
| BOOL_LIT | { Bool_Lit($1) } |
| STRING_LIT | { String_Lit($1) } |
| FLOAT_LIT | { Float_Lit($1) } |
| ID | { Id($1) } |
| expr PLUS expr | { Binop($1, Add, $3) } |
| expr MINUS expr | { Binop($1, Sub, $3) } |
| expr TIMES expr | { Binop($1, Mult, $3) } |
| expr DIVIDE expr | { Binop($1, Div, $3) } |
| expr MOD expr | { Binop($1, Mod, $3) } |
| expr EQ expr | { Binop($1, Eq, $3) } |
| expr NEQ expr | { Binop($1, Neq, $3) } |
| expr LEQ expr | { Binop($1, Leq, $3) } |
| expr LT expr | { Binop($1, Greater, $3) } |
expr AND expr { Binop($1, And, $3) }
expr OR expr { Binop($1, Or, $3) }
MINUS expr %prec NEG { Unop(Neg, $2) }
NOT expr { Unop(Not, $2) }
ID ASSIGN expr { Assign($1, $3) }
ID LPAREN actuals_opt RPAREN { Call($1, $3) }
expr DOT ID LPAREN actuals_opt RPAREN { Method($1, $3, $5) }
LPAREN expr RPAREN { $2 }
LBRACK graph_expr_opt RBRACK { match $2 with (n, e, n_i , is_d , is_w ) ->
  Graph_Lit(n, e, n_i , is_d, is_w) }

graph_expr_opt:
  /* nothing */ { [], [], [], false, false }
single_node_expr { match $1 with (n, e, n_i) -> (List.rev n, List.rev e, List.
    rev n_i, false, false) }
single_node_exprs_list { match $1 with (n, e, n_i) -> (List.rev n, List.rev e, List.
    rev n_i, false, false) }
ugraph_exprs_list { match $1 with (n, e, n_i) -> (List.rev n, List.rev e, List.
    rev n_i, true, false) }
digraph_exprs_list { match $1 with (n, e, n_i) -> (List.rev n, List.rev e, List.
    rev n_i, true, true) }
weigraph_exprs_list { match $1 with (n, e, n_i) -> (List.rev n, List.rev e, List.
    rev n_i, true, true) }
wedigraph_exprs_list { match $1 with (n, e, n_i) -> (List.rev n, List.rev e, List.
    rev n_i, true, true) }
single_node_exprs_list:
single_node_expr SEMI single_node_expr { merge_graph_exprs $1 $3 }
single_node_expr SEMI single_node_exprs_list { merge_graph_exprs $1 $3 }
ugraph_exprs_list:
ugraph_expr { $1 }
ugraph_expr_list SEMI ugraph_expr { merge_graph_exprs $1 $3 }
ugraph_expr_list SEMI single_node_expr { merge_graph_exprs $1 $3 }
single_node_expr ugraph_exprs_list { merge_graph_exprs $1 $3 }
digraph_exprs_list:
digraph_expr { $1 }
digraph_expr_list SEMI digraph_expr { merge_graph_exprs $1 $3 }
digraph_expr_list SEMI single_node_expr { merge_graph_exprs $1 $3 }
single_node_expr digraph_exprs_list { merge_graph_exprs $1 $3 }
ugraph_expr:
single_node_expr EDGE ID { update_graph $1 $3 (Int_Lit(0)) }
single_node_expr EDGE ID COLON expr { update_graph_e $1 $3 $5 (Int_Lit(0)) }
ugraph_expr EDGE ID { update_graph $1 $3 (Int_Lit(0)) }
ugraph_expr EDGE ID COLON expr { update_graph_e $1 $3 $5 (Int_Lit(0)) }
digraph_expr:
single_node_expr RARROW ID { update_digraph $1 $3 (Int_Lit(0)) 0 }
single_node_expr LARROW ID { update_digraph $1 $3 (Int_Lit(0)) 1 }
single_node_expr DIARROW ID { update_digraph_b $1 $3 (Int_Lit(0)) }
single_node_expr RARROW ID COLON expr { update_digraph_e $1 $3 $5 (Int_Lit(0)) 0 }
single_node_expr LARROW ID COLON expr { update_digraph_e $1 $3 $5 (Int_Lit(0)) 1 }
167 | single_node_expr DIARROW ID COLON expr { update_digraph_be $1 $3 $5 (Int_Lit(0)) }
168 | digraph_expr RARROW ID { update_digraph $1 $3 (Int_Lit(0)) 0 }
169 | digraph_expr LARROW ID { update_digraph $1 $3 (Int_Lit(0)) 1 }
170 | digraph_expr DIARROW ID { update_digraph_b $1 $3 (Int_Lit(0)) }
171 | digraph_expr RARROW ID COLON expr { update_digraph_e $1 $3 $5 (Int_Lit(0)) 0 }
172 | digraph_expr LARROW ID COLON expr { update_digraph_e $1 $3 $5 (Int_Lit(0)) 1 }
173 | digraph_expr DIARROW ID COLON expr { update_digraph_be $1 $3 $5 (Int_Lit(0)) }
174
175 /* weighted graphs */
176 wegraph_exprs_list:
177  wegraph_expr { $1 }
178 | wegraph_exprs_list SEMI wegraph_expr { merge_graph_exprs $1 $3 }
179 | wegraph_exprs_list SEMI single_node_expr { merge_graph_exprs $1 $3 }
180 | single_node_expr SEMI wegraph_exprs_list { merge_graph_exprs $1 $3 }
181
182 wegraph_expr:
183  single_node_expr MINUS LBRACE expr RBRACE MINUS ID %prec EDGE { update_graph $1 $7 $4 }
184 | single_node_expr MINUS LBRACE expr RBRACE MINUS ID COLON expr %prec EDGE { update_graph_e $1 $7 $9 $4 }
185 | wegraph_expr MINUS LBRACE expr RBRACE MINUS ID %prec EDGE { update_graph $1 $7 $4 }
186 | wegraph_expr MINUS LBRACE expr RBRACE MINUS ID COLON expr %prec EDGE { update_graph_e $1 $7 $9 $4 }
187
188 wedigraph_exprs_list:
189  wedigraph_expr { $1 }
190 | wedigraph_exprs_list SEMI wedigraph_expr { merge_graph_exprs $1 $3 }
191 | wedigraph_exprs_list SEMI single_node_expr { merge_graph_exprs $1 $3 }
192 | single_node_expr SEMI wedigraph_exprs_list { merge_graph_exprs $1 $3 }
193
194 wedigraph_expr:
195  single_node_expr MINUS LBRACE expr RBRACE RARROW MINUS ID %prec EDGE { update_digraph $1 $7 $4 0 }
196 | single_node_expr LARROW LBRACE expr RBRACE MINUS ID %prec EDGE { update_digraph $1 $7 $4 1 }
197 | single_node_expr LARROW LBRACE expr RBRACE RARROW MINUS ID %prec EDGE { update_digraph $1 $7 $4 0 }
198 | single_node_expr MINUS LBRACE expr RBRACE RARROW ID COLON expr %prec EDGE { update_digraph_e $1 $7 $9 $4 0 }
199 | single_node_expr LARROW LBRACE expr RBRACE RARROW ID COLON expr %prec EDGE { update_digraph_e $1 $7 $9 $4 1 }
200 | single_node_expr LARROW LBRACE expr RBRACE RARROW ID COLON expr %prec EDGE { update_digraph_be $1 $7 $9 $4 0 }
201 | wedigraph_expr MINUS LBRACE expr RBRACE RARROW ID %prec EDGE { update_digraph $1 $7 $4 0 }
202 | wedigraph_expr LARROW LBRACE expr RBRACE MINUS ID %prec EDGE { update_digraph $1 $7 $4 1 }
203 | wedigraph_expr LARROW LBRACE expr RBRACE RARROW ID %prec EDGE { update_digraph $1 $7 $4 0 }
204 | wedigraph_expr MINUS LBRACE expr RBRACE RARROW ID COLON expr %prec EDGE { update_digraph_e $1 $7 $9 $4 0 }
205 | wedigraph_expr LARROW LBRACE expr RBRACE RARROW ID COLON expr %prec EDGE { update_digraph_e $1 $7 $9 $4 1 }
206 | wedigraph_expr LARROW LBRACE expr RBRACE RARROW ID COLON expr %prec EDGE { update_digraph_be $1 $7 $9 $4 0 }
```ml
let merge_graph_exprs (n1, e1, n_i1) (n2, e2, n_i2) =
  (* essentially, take the union of node/edge/node_init lists. *)
  let add_if_missing list elem = if (List.mem elem list) then
    list
  else
    elem :: list
  in (List.fold_left add_if_missing n1 (List.rev n2),
    List.fold_left add_if_missing e1 (List.rev e2),
    List.fold_left add_if_missing n_i1 (List.rev n_i2))

let update_graph graph n weight = match graph with
  (nodes, edges, nodes_init) ->
    let nodes = if (List.mem n nodes) then (* if next node is already in this
      (* if next node is already in this graph, *)
      (n :: List.filter (fun x -> x <> n) nodes) (* move to front of nodelist
        so edges work *)
    else
      n :: nodes (* otherwise just add to front *)
    and edges =
      let new_edge = ((List.hd nodes), n, weight)
      and new_edge_rev = (n, (List.hd nodes), weight) in
      (* only add this edge if it's not already there *)
      if (List.mem new_edge edges || List.mem new_edge_rev edges) then
        edges
      else if (new_edge_rev = new_edge) then (* check for self-loop *)
        new_edge :: edges
      else
        new_edge_rev :: new_edge :: edges (* add in both directions for undir.
      graph *)
    in (nodes, edges, nodes_init)
```
let update_graph_e graph n expr weight = match graph with
  (nodes, edges, nodes_init) ->
  let nodes = if (List.mem n nodes) then (* if next node is already in this
  graph, *)
    (n :: List.filter (fun x -> x <> n) nodes) (* move to front of nodelist
  so edges work *)
  else
    n :: nodes (* otherwise just add to front *)
  and edges =
    let new_edge = ((List.hd nodes), n, weight)
    and new_edge_rev = (n, (List.hd nodes), weight) in
    (* only add this edge if it's not already there *)
    if (List.mem new_edge edges || List.mem new_edge_rev edges) then
      edges
    else if (new_edge_rev = new_edge) then (* check for self-loop *)
      new_edge :: edges
    else
      new_edge_rev :: new_edge :: edges (* add in both directions for undir.
  graph *)
  and nodes_init = (n, expr) :: nodes_init (* add node name/data pair to
  nodes_init *)
  in (nodes, edges, nodes_init)

let update_digraph graph n weight l = match graph with
  (nodes, edges, nodes_init) ->
  let nodes = if (List.mem n nodes) then (* if next node is already in this
  graph, *)
    (n :: List.filter (fun x -> x <> n) nodes) (* move to front of nodelist
  so edges work *)
  else
    n :: nodes (* otherwise just add to front *)
  and edges =
    let new_edge = if l == 0 then
      ((List.hd nodes), n, weight)
    else
      (n, List.hd nodes, weight) in
    (* only add this edge if it's not already there *)
    if (List.mem new_edge edges) then
      edges
    else
      new_edge :: edges
  in (nodes, edges, nodes_init)

let update_digraph_e graph n expr weight l = match graph with
  (nodes, edges, nodes_init) ->
  let nodes = if (List.mem n nodes) then (* if next node is already in this
  graph, *)
    (n :: List.filter (fun x -> x <> n) nodes) (* move to front of nodelist
  so edges work *)
  else
    n :: nodes (* otherwise just add to front *)
  and edges =
    let new_edge = if l == 0 then
      if (List.mem new_edge edges) then
82  ((List.hd nodes), n, weight)
83  else
84    (n, List.hd nodes, weight) in
85    (* only add this edge if it's not already there *)
86    if (List.mem new_edge edges) then
87      edges
88    else
89      new_edge :: edges
90      and nodes_init = (n, expr) :: nodes_init (* add node name/data pair to
91      nodes_init *)
92    in (nodes, edges, nodes_init)
93  let update_digraph_b graph n weight = match graph with
94    (nodes, edges, nodes_init) ->
95    let nodes = if (List.mem n nodes) then (* if next node is already in this
96      graph, *)
97      (n :: List.filter (fun x -> x <> n) nodes) (* move to front of nodelist
98      so edges work *)
99    else
100      n :: nodes (* otherwise just add to front *)
101      and edges =
102        let new_edge = ((List.hd nodes), n, weight) and new_edge_rev = (n, (List.hd nodes), weight) in
103        (* only add this edge if it's not already there *)
104        if (List.mem new_edge edges && List.mem new_edge_rev edges) then
105          edges
106        else if (List.mem new_edge edges) then
107          new_edge_rev :: edges
108        else if (List.mem new_edge_rev edges || new_edge_rev = new_edge) then
109          new_edge :: edges
110        else
111          new_edge :: new_edge_rev :: edges
112      in (nodes, edges, nodes_init)
113  let update_digraph_be graph n expr weight = match graph with
114    (nodes, edges, nodes_init) ->
115    let nodes = if (List.mem n nodes) then (* if next node is already in this
116      graph, *)
117      (n :: List.filter (fun x -> x <> n) nodes) (* move to front of nodelist
118      so edges work *)
119    else
120      n :: nodes (* otherwise just add to front *)
121    and edges =
122      let new_edge = ((List.hd nodes), n, weight) and new_edge_rev = (n, (List.hd nodes), weight) in
123      (* only add this edge if it's not already there *)
124      if (List.mem new_edge edges && List.mem new_edge_rev edges) then
125        edges
126      else if (List.mem new_edge edges) then
127        new_edge_rev :: edges
128      else if (List.mem new_edge_rev edges || new_edge_rev = new_edge) then
129        new_edge :: edges
130      else
131        new_edge :: new_edge_rev :: edges
132      and nodes_init = (n, expr) :: nodes_init (* add node name/data pair to
133      nodes_init *)
134      in (nodes, edges, nodes_init)
8.3 Semantic Checking

(* Authors: *)
Daniel Benett deb2174
Seth Benjamin sjb2190
Jennifer Bi jb3495
Jessie Liu jll2219

(*) type binop = Add | Sub | Mul | Div | Mod | Eq | Neq |
| Less | Leq | Greater | Geq | And | Or

(*) type unop = Neg | Not

(*) type typ = Int | Float | Bool | Void | String
   | Graph of typ
   | Digraph of typ
   | Wedgraph of typ
   | Wedigraph of typ
   | Node of typ
   | Edge of typ
   | Wedge of typ
   | Diwedge of typ
   | Map of typ

(*) type bind = typ * string

(*) type expr =
   | Id of string
   | Binop of expr * binop * expr
   | Unop of unop * expr
   | Assign of string * expr
   | Call of string * expr list
   | Method of expr * string * expr list
   | Bool_Lit of bool
   | Int_Lit of int
   | Float_Lit of float
   | String_Lit of string
   (* first bool is true if graph is directed; second bool is true if graph is
   weighted *)
   | Graph_Lit of string list * (string * string * expr) list * (string * expr)
   list * bool * bool
   | Noexpr

(*) type stmt =
   | Block of stmt list
   | If of expr * stmt * stmt
   | For of expr * expr * expr * stmt
   | While of expr * stmt
   | For_Node of string * expr * stmt
   | For_Edge of string * expr * stmt
   | Bfs of string * expr * expr * stmt
   | Dfs of string * expr * expr * stmt
<table>
<thead>
<tr>
<th>Break</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continue</td>
</tr>
<tr>
<td>Expr of expr</td>
</tr>
<tr>
<td>Vdecl of typ * string * expr</td>
</tr>
<tr>
<td>Return of expr</td>
</tr>
</tbody>
</table>

```ocaml
let string_of_op = function
    Add -> "+
    | Sub -> "-
    | Mul -> "*
    | Div -> "/
    | Mod -> "\n    | Eq -> "==
    | Neq -> "!="
    | Less -> "<
    | Leq -> "<=
    | Greater -> ">
    | Geq -> ">=
    | And -> "&&
    | Or -> "||

let string_of_uop = function
    Neg -> "-
    | Not -> "!

let rec string_of_typ = function
    Int -> "int"
    | Float -> "float"
    | Bool -> "bool"
    | String -> "str"
    | Node(t) -> "node<" ^ string_of_typ t ^ ">
    | Graph(t) -> "graph<" ^ string_of_typ t ^ ">
    | Digraph(t) -> "digraph<" ^ string_of_typ t ^ ">
    | Wegraph(t) -> "wedge<" ^ string_of_typ t ^ ">
    | Wedigraph(t) -> "wedigraph<" ^ string_of_typ t ^ ">
    | Edge(t) -> "edge<" ^ string_of_typ t ^ ">
    | Wedge(t) -> "wedge<" ^ string_of_typ t ^ ">
    | Divedge(t) -> "diedge<" ^ string_of_typ t ^ ">
    | Map(t) -> "map<" ^ string_of_typ t ^ ">
    | Void -> "void"
```

let rec string_of_expr = function
  | Bool_Lit(true) -> "true"
  | Bool_Lit(false) -> "false"
  | Int_Lit(1) -> string_of_int 1
  | String_Lit(l) -> l
  | Float_Lit(l) -> string_of_float l
  | Id(s) -> s
  | Binop(e1, o, e2) -> string_of_expr e1 ~ " " ~ string_of_op o ~ " " ~ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ~ string_of_expr e
  | Assign(v, e) -> v ~ " = " ~ string_of_expr e
  | Call(f, el) -> f ~ "(" ~ String.concat " " ~ (List.map string_of_expr el) ~ ")"
  | Method(e, m, el) -> string_of_expr e ~ "." ~ m ~ "(" ~ String.concat " " ~ (List.map string_of_expr el) ~ ")"
  | Graph_Lit(node_l, edge_l, node_init_l, _, _) -> "[
    " ~ String.concat ", " ~ (List.map (fun (f,t,w) -> "(" ~ f ~ ", " ~ t ~ " ~ " ~ string_of_expr w ~ ")") edge_l) ~ "]"
  |
  | Noexpr -> ""

let rec string_of_stmt = function
  | Block(stmts) -> "{\n" ~ String.concat " " ~ (List.map string_of_stmt stmts) ~ "}\n"
  | Expr(e) -> string_of_expr e ~ ";\n"
  | Vdecl(t, id, Noexpr) -> string_of_typ t ~ " " ~ id ~ ";\n"
  | Vdecl(t, id, e) -> string_of_typ t ~ " " ~ string_of_expr e ~ ";\n"
  | Return(e) -> "return " ~ string_of_expr e ~ ";\n"
  | If(e, s, Block([])) -> "if (" ~ string_of_expr e ~ ")\n" ~ string_of_stmt s
  | If(e, s1, s2) -> "if (" ~ string_of_expr e ~ ")\n" ~ string_of_stmt s1 ~ " else\n" ~ string_of_stmt s2
  | For(e1, e2, e3, s) -> "for (" ~ string_of_expr e1 ~ ") ~ " ~ string_of_expr e2 ~ " ~ string_of Expr e3 ~ ") ~ string_of_stmt s
  | While(e, s) -> "while (" ~ string_of_expr e ~ ") ~ string_of_stmt s
  | For_Node(n, g, sl) -> "for_node (" ~ n ~ ") ~ string_of_stmt s1
  | For_Edge(e, g, sl) -> "for_edge (" ~ e ~ ") ~ string_of_stmt s1
  | Bfs(n, g, src, sl) -> "bfs (" ~ n ~ ") ~ string_of_stmt s1

let string_of_vdecl (t, id) = string_of_typ t ~ " " ~ id ~ ";\n"

let string_of_fdecl fdecl = string_of_typ fdecl.f_typ ~ " " ~
  fdecl.f_name ~ "(" ~ String.concat " " ~ (List.map snd fdecl.f_formals) ~ ")\n" ~
  String.concat "" ~ (List.map string_of_stmt fdecl.f_body) ~

let string_of_program (vars, funcs) = String.concat "" ~ (List.map string_of_vdecl vars) ~ "\n" ~
  String.concat "" ~ (List.map string_of_fdecl funcs)
(* Authors: *)
Seth Benjamin sjb2190
Jennifer Bi jb3495
Jessie Liu jll2219
(*)

(* Semantically checked AST *)

open Ast

type sexpr =
  SId of string * typ
  | SBinop of sexpr * binop * sexpr * typ
  | SUNop of unop * sexpr * typ
  | SAssign of string * sexpr * typ
  | SMethod of sexpr * string * sexpr list * typ
  | SCall of string * sexpr list * typ
  | SBool_Lit of bool
  | SInt_Lit of int
  | SFloat_Lit of float
  | SString_Lit of string
  | SGraph_Lit of string list * (string * string * sexpr) list * (string * sexpr) list * typ * typ
  | SNoexpr

  type svdecl = {
  sv_name : string;
  sv_type : typ;
  sv_init : sexpr;
}

| type sstmt =
  SBlock of sstmt list
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt
  | SFor_Node of string * sexpr * sstmt
  | SFor_Edge of string * sexpr * sstmt
  | SBfs of string * sexpr * sexpr * sstmt
  | SDfs of string * sexpr * sexpr * sstmt
  | SBreak
  | SContinue
  | SExpr of sexpr * typ
  | SVdecl of typ * string * sexpr
  | SReturn of sexpr
type sfdecl = {
  sf_typ : typ;
  sf_name : string;
  sf_formals : bind list;
  sf_body : sstmt list;
}

type sprogram = bind list * sfdecl list

(* Pretty-printing functions *)

let rec string_of_sexpr = function
  SBool_Lit (true) -> "true"
| SBool_Lit (false) -> "false"
| SInt_Lit (l) -> string_of_int l
| SString_Lit (l) -> l
| SFloat_Lit (l) -> string_of_float l
| SId (s, t) -> s ^ ":" ^ string_of_typ t
| SBinop (e1, o, e2, t) ->
  string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^
  string_of_sexpr e2 ^ ":" ^ string_of_typ t
| SUnop (o, e, t) -> string_of_uop o ^ string_of_sexpr e ^ ":" ^ string_of_typ t
| SAssign (v, e, t) -> v ^ " = " ^ string_of_sexpr e ^ ":" ^ string_of_typ t
| SCall (f, el, t) ->
  f ^ "(" ^ String . concat ", " (List . map string_of_sexpr el) ^ ")" ^
  string_of_typ t
| SGraph_Lit (node_l, edge_l, node_init_l, g_typ, n_typ) ->
  "[" ^ String . concat ", " node_l ^ "] " ^
  "[" ^ String . concat ", " (List . map (fun (f, t, w) -> "(" ^ f ^ ", " ^ t ^ ", " ^
  string_of_sexpr w ^ ")") edge_l ^ "] " ^
  "[" ^ String . concat ", " (List . map (fun (n, e) -> "(" ^ n ^ ", " ^
  string_of_sexpr e ^ ")") node_init_l ^ "] : " ^ string_of_typ g_typ ^ ", " ^
  string_of_typ n_typ
| SMethod (i, m, e, r_typ) -> string_of_sexpr i ^ "." ^ m ^ ") ^
  "(" ^ String . concat ", " (List . map string_of_sexpr e) ^ ")" ^
  string_of_typ r_typ
| SNoexpr -> ""

let rec string_of_sstmt = function
  SBlock (stmts) ->
  "{" ^ String . concat ", " (List . map string_of_sstmt stmts) ^ "}\n"
| SExpr (expr, t) -> string_of_sexpr expr ^ ";\n"
| SVdecl (t, id, SNoexpr) -> string_of_typ t ^ " " ^ id ^ ";\n"
| SVdecl (t, a_expr) -> string_of_typ t ^ " " ^ string_of_sexpr a_expr ^ ";\n"
| SReturn (expr) -> "return " ^ string_of_sexpr expr ^ ";\n"
| SIf (e, s, SBlock ([])) -> "if (" ^ string_of_sexpr e ^ ")\n"
| SIf (e, s1, s2) -> "if (" ^ string_of_sexpr e ^ ")\n"
| SFor (e1, e2, e3, s) ->
  "for (" ^ string_of_sexpr e1 ^ ") " ^ string_of_sexpr e2 ^ ") " ^
  string_of_sexpr e3 ^ ") " ^ string_of_sstmt s
| SWhile (e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s
| SFor_Node (n, g, sl) -> "for_node (" ^ n ^ " " ^ string_of_sexpr g ^ ") " ^
  string_of_sstmt s

let rec string_of_sstmt = string_of_sstmt

let rec string_of_sprogram = string_of_sprogram
let string_of_svdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";
"  
let string_of_sfdecl fdecl = 
  string_of_typ fdecl.sf_typ ^ " " ^ 
  fdecl.sf_name ^ "(" ^ String.concat ", " ^ (List.map snd fdecl.sf_formals) ^ 
  ")
\n" ^ 
  String.concat "" ^ (List.map string_of_sstmt fdecl.sf_body) ^ 
  "\n"

let string_of_sprogram (vars, funcs) = 
  String.concat "" ^ (List.map string_of_svdecl vars) ^ "\n" ^ 
  String.concat "\n" ^ (List.map string_of_sfdecl funcs)
Method (e, "print", e_lst) -> (check_graphmtd e "print" 0 e_lst Void env)
Method (e, "add_node", e_lst) -> (check_graphmtd e "add_node" 1 e_lst Void env)
Method (e, "remove_node", e_lst) -> (check_graphmtd e "remove_node" 1 e_lst Void env)
Method (e, "has_node", e_lst) -> (check_graphmtd e "has_node" 1 e_lst Bool env)
Method (e, "add_edge", [f;t]) -> (check_graphmtd e "add_edge" 2 [f;t] Void env)
Method (e, "add_edge", [f;t;w]) -> (check_graphmtd e "add_edge" 3 [f;t;w] Void env)
Method (e, "remove_edge", e_lst) -> (check_graphmtd e "remove_edge" 2 e_lst Void env)
Method (e, "has_edge", e_lst) -> (check_graphmtd e "has_edge" 2 e_lst Bool env)
Method (e, "neighbors", e_lst) -> (check_graphmtd e "neighbors" 1 e_lst (Graph(Int)) env)
Method (e, "get_edge_weight", e_lst) -> (check_graphmtd e "get_edge_weight" 2 e_lst Int env)
Method (e, "set_edge_weight", e_lst) -> (check_graphmtd e "set_edge_weight" 3 e_lst Int env)
Method (e, "put", e_lst) -> (check_mapmtd e "put" 2 e_lst env)
Method (e, "get", e_lst) -> (check_mapmtd e "get" 1 e_lst env)
Method (e, "contains", e_lst) -> (check_mapmtd e "contains" 1 e_lst env)
Method (e, s2, e_lst) -> (report_meth_not_found s2)
Call ("print", e_lst) -> (check_print e_lst env)
Call ("printb", e_lst) -> (check_print e_lst env)
Call ("prints", e_lst) -> (check_print e_lst env)
Call (str, e_lst) -> (check_call str e_lst env)
Int_Lit (i) -> (SInt_Lit (i), env)
Float_Lit (f) -> (SFloat_Lit (f), env)
String_Lit (str) -> (SString_Lit (str), env)
Bool_Lit (bool) -> (SBool_Lit (bool), env)
Graph_Lit (str_lst, ed_lst, n_lst, is_d, is_w) ->
(check_graph str_lst ed_lst n_lst is_d is_w env)
Noexpr -> (SNoexpr, env)

and convert_expr_list expr_lst env =
mismatch expr_lst with
[ ] -> [], env
| _ ->
  let rec add_sexpr acc ex_lst env = match ex_lst with
  [ ] -> acc, env
  | e :: e_lst ->
    let sexpr, new_env = convert_expr e env in
    let new_acc = sexpr::acc in add_sexpr new_acc e_lst new_env
  in
  let sexpr_lst, nenv = add_sexpr [] expr_lst env in
  let new_env = {
    env_name = env.env_name;
    env_return_type = env.env_return_type;
    env_fmap = env.env_fmap;
    env_sfmap = nenv.env_sfmap;
    env_globals = env.env_globals;
    env_flocals = env.env_flocals;
    env_fformals = env.env_fformals;
    }
and get_sexpr_type sexpr = match sexpr with
   | SId(_, typ) -> typ
   | SBinop(_, _, _, typ) -> typ
   | SUnop(_, _, typ) -> typ
   | SAssign(_, _, typ) -> typ
   | SCall(_, _, typ) -> typ
   | SMethod(_,_,_, typ) -> typ
   | SBool_Lit(_) -> Bool
   | SInt_Lit(_) -> Int
   | SFloat_Lit(_) -> Float
   | SString_Lit(_) -> String
   | SGraph_Lit(_,_,_,subtype,_) -> subtype
   | SNoexpr -> Void

and get_sexpr_lst_type sexpr_lst =
   List.map (fun sexpr -> get_sexpr_type sexpr) sexpr_lst

(* s is the id being checked *)

and check_id str env =
   try let typ = StringMap.find str env.env_flocals in
     SId(str, typ) with Not_found ->
   (try let typ = StringMap.find str env.env_fformals in
     SId(str, typ) with Not_found ->
   (try let typ = StringMap.find str env.env_globals in
     SId(str, typ) with Not_found ->
     raise (Failure("undeclared identifier " ^ str))))

and check_id_typ str typ env =
   if not (StringMap.mem str env.env_flocals || StringMap.mem str env.env_fformals
      || StringMap.mem str env.envGlobals)
   then raise(Failure("node " ^ str ^ " is undefined"))
   else
     let lval = try StringMap.find str env.env_flocals with
       Not_found -> (try StringMap.find str env.env_fformals with
                      Not_found -> StringMap.find str env.env_globals) in
     if lval <> typ then raise(Failure("variable " ^ str ^ " is of type " ^
                               string_of_typ lval ^ " when " ^
                               string_of_typ typ ^ " was expected"));

and check_binop e1 op e2 env =
   let (s1, _) = convert_expr e1 env
   and (s2, _) = convert_expr e2 env in
   let t1 = get_sexpr_type s1 and t2 = get_sexpr_type s2 in
   (match op with
     | Add ->
       (match t1, t2 with
         Int, Int      -> SBinop(s1, op, s2, Int)
     | _ -> raise (Failure("addition is not defined for type " ^
                           string_of_typ t1 ^ " and " ^
                           string_of_typ t2))))
check_unop op e env =  
  let (s, _) = convert_expr e env in  
  let t = get_sexpr_type s in  
  match op with  
    Neg when (t = Int || t = Float) -> SUnop (op, s, t)  
    Not when t = Bool -> SUnop (op, s, t)  
    _ -> report_bad_unop op t

and check_print e_lst env =  
  if ((List.length e_lst) != 1) then raise(Failure("wrong number of arguments") )  
  else  
    let (s, nenv) = convert_expr (List.hd e_lst) env in  
    let t = get_sexpr_type s in  
    let strcall =  
      match t with  
        Int -> "print"  
        String -> "prints"  
        Bool -> "printb"  
        _ -> raise(Failure("type " ^ string_of_typ t ^ " is unsupported for this function"))  
      in  
        SCall (strcall, [s], Void), nenv

and check_assign str e env =  
  let (r, env) = convert_expr e env in  
  let rvaluet = get_sexpr_type r in  
  (* check if the id has been declared *)  
  if StringMap.mem str env.env_flocals || StringMap.mem str env.env_fformals ||
StringMap.mem str env.env_globals then
    let lvaluet = try StringMap.find str env.env_flocals with
      Not_found -> (try StringMap.find str env.env_fformals with
        Not_found -> StringMap.find str env.env_globals) in
    (* if types match *)
    if lvaluet = rvaluet then SAssign(str, r, lvaluet), env
    else
      (* The parser always says an edgeless graph literal is of type Graph,
        but it is a valid rvalue for Digraph, Wgraph, and Wedigraph too -
        if this is why lvaluet <> rvaluet, this is fine.
        Similarly, semant always says the graph literal is of type Graph<
        Void>,
        but it is always a valid rvalue for any type of Graph with any data
        type. *)
      (match (lvaluet, r) with
        (Digraph(t1), SGraph_Lit(_, [], _, Graph(t2), _)) when t1 = t2 ->
          SAssign(str, r, lvaluet), env
        | (Wgraph(t1), SGraph_Lit(_, [], _, Graph(t2), _)) when t1 = t2 ->
          SAssign(str, r, lvaluet), env
        | (Wedigraph(t1), SGraph_Lit(_, [], _, Graph(t2), _)) when t1 = t2 ->
          SAssign(str, r, lvaluet), env
        | (Graph(_), SGraph_Lit(_, [], _, Graph(Void), _))
        | (Digraph(_), SGraph_Lit(_, [], _, Graph(Void), _))
        | (Wgraph(_), SGraph_Lit(_, [], _, Graph(Void), _))
        | (Wedigraph(_), SGraph_Lit(_, [], _, Graph(Void), _)) ->
          SAssign(str, r, lvaluet), env
        | _ -> report_bad_assign lvaluet rvaluet
      )
    else report_undeclared_id_assign str

and check_data e e_lst env =
  let len = List.length e_lst in
  if (len != 0) then raise(Failure("data takes 0 arguments but " ~
    string_of_int len ~ " arguments given"))
  else
    let (s, env) = convert_expr e env in
    let t = get_sexpr_type s in
    match t with
      Node(dt) -> SMethod(s, "data", [], dt), env
      | _ -> raise(Failure("data() called on type " ~ string_of_typ t ~ " when
          node was expected"))
  and check_sdata e e_lst env =
    let len = List.length e_lst in
    if (len != 1) then raise(Failure("set_data() takes 1 arguments but " ~
      string_of_int len ~ " arguments given"))
    else
      let (id, env) = convert_expr e env in
      let id_t = get_sexpr_type id in
      let (arg, env) = convert_expr (List.hd e_lst) env in
      let arg_t = get_sexpr_type arg in
      match (id_t, arg_t) with
        (Node(dt), at) when dt = at -> SMethod(id, "set_data", [arg], Void), env
        | (Node(dt), at) -> raise(Failure("set_data() called on type " ~
            string_of_typ id_t ~ " with parameter " ~

40
and check_graphmtd g name args e_lst ret_typ env =
    let id, env = convert_expr g env in
    let t = get_sexpr_type id in
    let data_type = match t with
        Graph (dt) -> dt
      | Digraph (dt) -> dt
      | Wedigraph (dt) -> dt
      | _ ->
        raise (Failure (name ^ " called on type " ^ string_of_typ t ^ " when graph was expected")) in
    let len = List.length e_lst in
    if (len != args) then raise (Failure (name ^ " takes " ^ string_of_int args ^ " arguments but " ^ string_of_int len ^ " arguments given"));
    let sexpr , env =
        match args with
        0 (* print *) -> (match data_type with
            Int | Float | Bool | String -> (SMethod (id, name, [], ret_typ)), env
        | _ -> raise (Failure ("print cannot be called on graphs with generic
data types")))
      | 1 -> ( let e1 = (List.hd e_lst) in
              let (ex, env1) = convert_expr e1 env in
              let t1 = get_sexpr_type ex in
              if (t1 <> Node (data_type))
                  then raise (Failure ("graph method " ^ name ^ " may not be called on
graph of type " ^ string_of_typ t ^ " with parameter " ^ string_of_typ t1));
              let new_env = {
                  env_name = env.env_name;
                  env_return_type = env.env_return_type;
                  env_fmap = env.env_fmap;
                  env_sfmap = env.env_sfmap;
                  env_globals = env.env_globals;
                  env_flocals = env.env_flocals;
                  env_fformals = env.env_fformals;
                  env_in_loop = env.env_in_loop;
              } in
              let ret_typ = if (ret_typ = Graph (Int)) then Graph (data_type) (*
                  neighbors *)
                  else ret_typ in
                  (SMethod (id, name, [ex], ret_typ)), new_env)
        | 2 ->
          let e1 = (List.hd e_lst) and e2 = (List.nth e_lst 1) in
          let (ex, env1) = convert_expr e1 env in
          let (ex2, env2) = convert_expr e2 env1 in
          let t1 = get_sexpr_type ex and t2 = get_sexpr_type ex2 in
          if (t1 <> Node (data_type) || t2 <> Node (data_type))
              then raise (Failure ("graph method " ^ name ^ " may not be called on graph of
type " ^ string_of_typ t ^ " with parameters " ^ string_of_typ t1 ^ ", " ^ string_of_typ t2 ^ ")

string_of_typ at " when "
string_of_typ dt " was expected")
| (_, _) -> raise (Failure ("set_data() called on type " ^ string_of_typ id_t " when node was expected"))
string_of_typ t2));
(* make sure this method can be called on this type *)
if (name = "get_edge_weight" && (t = Graph(data_type) || t = Digraph(data_type)))
  then raise(Failure(name ~ " may not be called on unweighted graphs"));
if (name = "add_edge" && (t = Wegraph(data_type) || t = Wedigraph(data_type)))
  then raise(Failure(name ~ " may not be called on weighted graphs without a weight argument"));

let new_env = {
  env_name = env.env_name;
  env_return_type = env.env_return_type;
  env_fmap = env.env_fmap;
  env_sfmap = env2.env_sfmap;
  envGlobals = env.envGlobals;
  env_flocals = env.env_flocals;
  env_fformals = env.env_fformals;
  env_in_loop = env.env_in_loop;
} in
(SMethod(id, name, [ex; ex2], ret_typ)), new_env | 3 ->
  let e1 = (List.hd e_lst)
  and e2 = (List.nth e_lst 1)
  and e3 = (List.nth e_lst 2) in
  let (ex,env1) = convert_expr e1 env in
  let (ex2,env2) = convert_expr e2 env1 in
  let (ex3,env3) = convert_expr e3 env2 in
  let t1 = get_sexpr_type ex
  and t2 = get_sexpr_type ex2
  and t3 = get_sexpr_type ex3 in
  if (t1 <> Node(data_type) || t2 <> Node(data_type) || t3 <> Int)
    then raise(Failure("graph method " ^ name ^ " may not be called on graph of type " ^ string_of_typ t ^ ", " ^ string_of_typ t2 ^ ", " ^ string_of_typ t3));
(* all 3-argument graph methods can only be called on we(di)graphs *)
if (t = Graph(data_type) || t = Digraph(data_type)) then
  (if (name = "add_edge") then
    raise(Failure(name ~ " may not be called on unweighted graphs with a weight argument"));
    raise(Failure(name ~ " may not be called on unweighted graphs"));)
  let new_env = {
    env_name = env.env_name;
    env_return_type = env.env_return_type;
    env_fmap = env.env_fmap;
    env_sfmap = env3.env_sfmap;
    envGlobals = env.envGlobals;
    env_flocals = env.env_flocals;
    env_fformals = env.env_fformals;
    env_in_loop = env.env_in_loop;
  } in
  (SMethod(id, name, [ex; ex2; ex3], ret_typ)), new_env
in sexpr, env

and check_edgemtd e n e_lst env =
        let len = List.length e_lst in

42
let rec add_sexpr acc e_lst env = match e_lst with
  [] -> acc, env
| e :: e_lst ->
  let se, new_env = convert_expr e env in
  let new_acc = se :: acc in add_sexpr new_acc e_lst new_env
in
let e_lst_checked, nenv = add_sexpr [] e_lst env in
let new_env = {
  env_name = env.env_name;
  env_return_type = env.env_return_type;
  env_fmap = env.env_fmap;
  env_sfmap = nenv.env_sfmap;
  env_globals = env.env_globals;
  env_flocals = env.env_flocals;
  env_fformals = env.env_fformals;
  env_in_loop = env.env_in_loop;
}

let correct_len = match n with "set_weight" -> 1 | _ -> 0 in
if (len != correct_len) then
  raise (Failure (n ^ " takes " ^ string_of_int correct_len ^ " arguments but " ^ string_of_int len ^ " arguments given"))
else
  let se, nenv = convert_expr e new_env in
  let t = get_sexpr_type se in
  let new_env = {
    env_name = env.env_name;
    env_return_type = env.env_return_type;
    env_fmap = env.env_fmap;
    env_sfmap = nenv.env_sfmap;
    env_globals = env.env_globals;
    env_flocals = env.env_flocals;
    env_fformals = env.env_fformals;
    env_in_loop = env.env_in_loop;
  }

  match t with
  Diwedge (dt) | Wedge (dt) ->
    (match n with
      "from" -> SMethod (se, n, List.rev e_lst_checked, Node (dt)), new_env
      | "to" -> SMethod (se, n, List.rev e_lst_checked, Node (dt)), new_env
      | "weight" -> SMethod (se, n, List.rev e_lst_checked, Int), new_env
      | "set_weight" -> SMethod (se, n, List.rev e_lst_checked, Void), new_env
    )
  | Edge (dt) ->
    (match n with
      "from" -> SMethod (se, n, List.rev e_lst_checked, Node (dt)), new_env
      | "to" -> SMethod (se, n, List.rev e_lst_checked, Node (dt)), new_env
      | "weight" -> raise (Failure ("weight() cannot be called on edges of unweighted graphs"));
      | "set_weight" -> raise (Failure ("set_weight() cannot be called on edges of unweighted graphs"));
      | _ -> raise (Failure ("Edge method " ^ n ^ " called on type " ^ string_of_typ t));
    )
  and check_mapmtd m name args e_lst env =
  let id, env = convert_expr m env in
let t = get_sexpr_type id in
let value_typ = match t with
  Map (v) -> v
| _ -> raise (Failure (name ^ " called on type " ^ string_of_typ t ^ " when map was expected"))
in
let len = List.length e_lst in
if (len <> args) then raise (Failure (name ^ " takes " ^ string_of_int args ^ " arguments but " ^ string_of_int len ^ " arguments given"));

let sexpr, env =
  match args with
  1 -> ( (* get(k), contains(k) *)
    let e1 = (List.hd e_lst) in
    let (ex, nenv) = convert_expr e1 env in
    let t1 = get_sexpr_type ex in
    (match t1 with
      Node (dt) -> ()
    | _ -> raise (Failure ("map method " ^ name ^ " may not be called on type " ^ string_of_typ t1)));
    let new_env = {
      env_name = env.env_name;
      env_return_type = env.env_return_type;
      env_fmap = env.env_fmap;
      env_sfmap = nenv.env_sfmap;
      env_globals = env.env_globals;
      env_flocals = env.env_flocals;
      env_fformals = env.env_fformals;
      env_in_loop = env.env_in_loop;
    } in
    let ret_typ = (match name with "contains" -> Bool | _ -> value_typ) in
    (SMethod (id, name, [ex], ret_typ)), new_env)
  | 2 -> (* put(k,v) *)
    let e1 = (List.hd e_lst) and e2 = (List.nth e_lst 1) in
    let (ex, env1) = convert_expr e1 env in
    let (ex2, env2) = convert_expr e2 env1 in
    let t1 = get_sexpr_type ex and t2 = get_sexpr_type ex2 in
    (match t1 with
      Node (dt) -> ()
    | _ -> raise (Failure ("map method " ^ name ^ " may not be called with key type " ^ string_of_typ t1)));
    if (t2 <> value_typ)
      then raise (Failure ("map method " ^ name ^ " called with value type " ^ string_of_typ t2 ^ " on map of type " ^ string_of_typ value_typ ));
    let new_env = {
      env_name = env.env_name;
      env_return_type = env.env_return_type;
      env_fmap = env.env_fmap;
      env_sfmap = env2.env_sfmap;
      envGlobals = env.env_globals;
      envGlobals = env.env_globals;
      envFlocals = env.env_flocals;
      envFformals = env.env_fformals;
      envIn_loop = env.env_in_loop;
    } in
    (SMethod (id, name, [ex; ex2], Void)), new_env}
in sexpr, env

(* TODO *)
and check_graph str_lst ed_lst n_lst is_d is_w env =
  let graph_type data_type = match (is_d, is_w) with
  (true, true) -> Wedigraph(data_type)
  | (true, false) -> Digraph(data_type)
  | (false, true) -> Wegraph(data_type)
  | (false, false) -> Graph(data_type)
  in
  (* convert each weight expression *)
  let ed_lst_checked = List.map (fun (f,t,w) -> (f, t, fst (convert_expr w env))) ed_lst in
  let weight_types = List.map (fun (_,_,w_sexpr) -> get_sexpr_type w_sexpr) ed_lst_checked in
  List.iter (fun t -> if t <> Int then
    raise (Failure("edge weights must be of type int"))) weight_types;
  (* make sure no nodes are initialized more than once *)
  ignore(List.fold_left (fun m (n,_) -> if StringMap.mem n m then
    raise(Failure("graph literal cannot initialize the same node more than once"))
  else StringMap.add n true m) StringMap.empty n_lst);
  (* make sure no edge appears more than once (may happen with we(di)graphs) *)
  ignore(List.fold_left (fun m (f,t,_) -> if StringMap.mem (f ^ "+" ^ t) m then
    raise(Failure("graph literal cannot feature the same edge with different weights"))
  else StringMap.add (f ^ "+" ^ t) true m) StringMap.empty ed_lst);
  match str_lst with
  [] -> SGraph_Lit([], [], [], Graph(Void), Void), env
  | _ ->
    let is_declared env str = (StringMap.mem env.env_flocals ||
      StringMap.mem env.env_fformals ||
      StringMap.mem env.env_globals) in
    let declare env str =
      if (StringMap.mem env.env_flocals) then
        StringMap.find env.env_flocals
      else if (StringMap.mem env.env_fformals) then
        StringMap.find env.env_fformals
      else
        StringMap.find env.env_globals
    in
    match node_type with Node(t) -> t | _ -> node_type
    in
    let declared = List.filter (is_declared env) str_lst in
    let declared_types = List.map (get_declared_type env) declared in
    let rec check_consistent types = (match types with
      [] -> true
      | [..] -> true
      | hd :: tl -> (hd = List.hd tl) && (check_consistent tl))
    in
    if (not (check_consistent declared_types)) then
      raise(Failure("all nodes in graph literal must have the same data type"))
    ;
    (match (declared_types, n_lst) with
[] , [] -> raise(Failure("graph literal must contain at least one
previously " ^
"declared node or at least one initialized node
"));
| _, [] -> let t = List.hd declared_types in
    let newenv = List.fold_left (fun x y -> let (_, z) = check_vdecl (Node(t
)) y Noexpr true x in z) env str_lst in
    SGraph_Lit(str_lst, ed_lst_checked, [], (graph_type t), t), newenv
| [], _ -> let (s,_) = convert_expr (snd (List.hd n_lst)) env in
    let t = get_sexp_type s in
    if t <> s then raise (Failure("node type mismatch of " ^
"string_of_typ t " and " ^ string_of_typ t\n"))) n_lst;
    let newenv = List.fold_left (fun x y -> let (_, z) = check_vdecl (Node(t
)) y Noexpr true x in z) env str_lst in
    let nodes = List.map (fun (x,y) -> let (s,_) = convert_expr y newenv in
                      (x,s)) n_lst
    in
    (SGraph_Lit(str_lst, ed_lst_checked, nodes, (graph_type t), t), newenv)
| _, _ ->
    let n_lst_types = List.map (fun n -> (get_sexp_type (fst (convert_expr
(snd n) env))))) n_lst in
    if (check_consistent (declared_types @ n_lst_types)) then
        let t = List.hd declared_types in
        let newenv = List.fold_left (fun x y -> let (_, z) = check_vdecl (Node(t
)) y Noexpr true x in z) env str_lst in
        let nodes = List.map (fun (x,y) -> let (s,_) = convert_expr y newenv
                      in (x,s)) n_lst
        in
        (SGraph_Lit(str_lst, ed_lst_checked, nodes, (graph_type t), t),
         newenv)
    else
        raise(Failure("all nodes in graph literal must have the same data type
"));
and check_call str e_lst env =
(* can't call main *)
if str == "main" then raise (Failure("can't make call to main"))
(* check if function can be found*)
else if not (StringMap.mem str env.env_fmap) then report_function_not_found
  str
else
(* semantically check all the arguments*)
    let checked_args, env = convert_expr_list e_lst env in

(* get the types of the args*)
    let arg_types = get_sexpr_lst_type (checked_args) in

if not (StringMap.mem str env.env_sfmap) then
    let fdecl = StringMap.find str env.env_fmap
    in
    let nenv =
      {
      env_name = env.env_name;
      env_return_type = env.env_return_type;
      env_fmap = env.env_fmap;
 46
env_sfmap = (convert_fdecl fdecl . f_name fdecl . f_formals env).

env_globals = env.env_globals;
env_flocals = env.env_flocals;
env_fformals = env.env_fformals;
env_in_loop = env.env_in_loop;

} in

let sfdecl = StringMap . find str nenv . env_sfmap in try

(* confirm types match *)
List.iter2 (fun t1 (t2 , _) -> if t1 <> t2 then report_typ_args t2 t1 else ()) arg_types sfdecl . sf_formals;
let sexpr_lst , env = convert_expr_list e_lst env in
SCall(str , sexpr_lst , sfdecl . sf_typ ), nenv with

(* wrong number of arguments *)
Invalid_argument _ -> raise (Failure ("expected " ^ string_of_int (List.length sfdecl . sf_formals) ^ "arguments when " ^ string_of_int (List.length e_lst) ^ " arguments were provided"))

else let sfdecl = StringMap . find str env . env_sfmap in try

(* confirm types match *)
List.iter2 (fun t1 (t2 , _) -> if t1 <> t2 then report_typ_args t2 t1 else ()) arg_types sfdecl . sf_formals;
let sexpr_lst , env = convert_expr_list e_lst env in
SCall(str , sexpr_lst , sfdecl . sf_typ ), env with

(* wrong number of arguments *)
Invalid_argument _ -> raise (Failure ("expected " ^ string_of_int (List.length sfdecl . sf_formals) ^ "arguments when " ^ string_of_int (List.length e_lst) ^ " arguments were provided"))

and convert_stmt stmt env = match stmt with
Block (s_lst) -> (check_block s_lst env)
| If (e , s1 , s2) -> (check_if e s1 s2 env)
| For (e1 , e2 , e3 , s) -> (check_for e1 e2 e3 s env)
| While (e , s) -> (check_while e s env)
| For_Node (str , e, s) -> (check_for_node str e s env)
| For_Edge (str , e, s) -> (check_for_edge str e s env)
| Bfs (str , e1 , e2 , s) -> (check_bfs str e1 e2 s env)
| Dfs (str , e1 , e2 , s) -> (check_dfs str e1 e2 s env)
| Break -> (check_break env , env)
| Continue -> (check_continue env , env)
| Expr (e) -> (check_expr_stmt e env)
| Vdecl (t , str , e) -> (check_vdecl t str e false env)
| Return (e) -> (check_return e env)

and check_block s_lst env = match s_lst with
[] -> SBlock[[SExpr(SNoexpr , Void)]], env
| _ -> (*check every statement, and put those checked statements in a list *)

let rec add_sstmt acc stmt_lst env = match stmt_lst with
[] -> acc, env
| Return _ :: _ :: _ -> raise (Failure("nothing may follow a return"))
| st :: st_lst ->
  let sstmt, new_env = convert_stmt st env in
  let new_acc = sstmt::acc in add_sstmt new_acc st_lst
new_env
in
  let sblock, nenv = add_sstmt [] s_lst env in
  let new_env = {
    env_name = env.env_name;
    env_return_type = env.env_return_type;
    env_fmap = env.env_fmap;
    env_sfmap = nenv.env_sfmap;
    env_globals = env.env_globals;
    env_flocals = env.env_flocals;
    env_fformals = env.env_fformals;
    env_in_loop = env.env_in_loop;
  }
  in
  (SBlock(List.rev sblock), new_env)
and check_if cond is es env =
(* semantically check the condition *)
  let scond, env = convert_expr cond env in
  (match get_sexpr_type scond with
  Bool ->
    let (sis, env) = convert_stmt is env in
    let (ses, env) = convert_stmt es env in
    SIf(scond, sis, ses), env
  | _ -> raise(Failure("Expected boolean expression")))
and check_for e1 e2 e3 s env =
(* a new var may be added to locals there*)
  let (sel, env1) = convert_expr e1 env in
  let (se2, env2) = convert_expr e2 env1 in
  let (se3, env3) = convert_expr e3 env2 in
  let new_env = {
    env_name = env3.env_name;
    env_return_type = env3.env_return_type;
    env_fmap = env3.env_fmap;
    env_sfmap = env3.env_sfmap;
    env_globals = env3.env_globals;
    env_flocals = env3.env_flocals;
    env_fformals = env3.env_fformals;
    env_in_loop = true;
  }
  in
  let (for_body, nenv) = convert_stmt s new_env in
(* check if you have a return in a for *)
  match for_body with
  SBlock(slst) ->
let rets = List.filter (fun x -> match x with
  | SReturn(expr) -> true
  | _ -> false) slst

  in
  if List.length rets != 0 then raise(Failure("cannot return in for loop "));

let nnenv =
{
  env_name = env.env_name;
  env_return_type = env.env_return_type;
  env_fmap = env.env_fmap;
  env_sfmap = nenv.env_sfmap;
  env_globals = env.env_globals;
  env_flocals = env.env_flocals;
  env_fformals = env.env_fformals;
  env_in_loop = env.env_in_loop;
}

in
if (get_sexpr_type se2) = Bool then
  SFor(se1, se2, se3, for_body), nnenv
else raise(Failure("Expected boolean expression"))

and check_while e s env =
let (se, nenv) = convert_expr e env in
let new_env =
{
  env_name = nenv.env_name;
  env_return_type = nenv.env_return_type;
  env_fmap = nenv.env_fmap;
  env_sfmap = nenv.env_sfmap;
  env_globals = nenv.env_globals;
  env_flocals = nenv.env_flocals;
  env_fformals = nenv.env_fformals;
  env_in_loop = true;
}

in
let (while_body, nenv) = convert_stmt s new_env in

let nnenv =
{
  env_name = env.env_name;
  env_return_type = env.env_return_type;
  env_fmap = env.env_fmap;
  env_sfmap = nenv.env_sfmap;
  env_globals = env.env_globals;
  env_flocals = env.env_flocals;
  env_fformals = env.env_fformals;
  env_in_loop = env.env_in_loop;
}

in
if (get_sexpr_type se) = Bool then
  SWhile(se, while_body), nnenv
else raise(Failure("Expected boolean expression"))
and check_for_node str e s env =
 (* This is jank as hell but it's the last day so whatever *)
 let graph_type = get_sexpr_type (fst (convert_expr e env)) in
 let node_type = match graph_type with
 Wedigraph(dt) | Wegraph(dt) | Graph(dt) | Digraph(dt) -> Node(dt)
 | _ -> raise(Failure("type " ^ string_of_type graph_type ^
" may not be iterated with for_node")); (* TODO: write
a test for this *)

let flocals = StringMap.add str node_type env.env_flocals in
let new_env =
{
  env_name = env.env_name;
  env_return_type = env.env_return_type;
  env_fmap = env.env_fmap;
  env_sfmap = env.env_sfmap;
  env_globals = env.env_globals;
  env_flocals = flocals;
  env_fformals = env.env_fformals;
  env_in_loop = env.env_in_loop;
}
in
let (se , senv ) = convert_expr e new_env in
let gname = match se with (* cannot call the following methods on a NAMED
graph *)
 (* unnamed graph is safe since it cannot modify the graph itself *)
| SId (str , Graph(_)) -> str
| SId (str , Digraph(_)) -> str
| SId (str , Wedigraph(_)) -> str
| SId (str , Wegraph(_)) -> str
| _ -> "" (*not sure if empty string is problematic *)

let for_body = convert_stmt s senv in
let.chk = match for_body with
| SBlock (lst) -> match lst with
| [] -> ();
| _ -> (* non empty statement lst, check em*)
| _ -> ();
let chkcall x = match x with
| SExpr(SMethod(g,fname,_,_),_) -> (match g,fname with
  SId(s,_) ,"add_node" -> if s = gname then
  report_concurrent_mod "for_node"
  | SId(s,_) ,"remove_node" -> if s = gname then
  report_concurrent_mod "for_node"
  | _ -> ();;

let nenv =
{
  env_name = env.env_name;
  env_return_type = env.env_return_type;
  env_fmap = env.env_fmap;
  env_sfmap = env.env_sfmap;
  env_globals = env.env_globals;
  env_flocals = env.env_flocals;
}
env_fformals = env.env_fformals;
env_in_loop = env.env_in_loop;
}
in
SFor_Node(str, se, for_body), nenv

and check_for_edge str e s env =
  let graph_type = get_sexpr_type (fst (convert_expr e env)) in
  let edge_type = match graph_type with
    Wedigraph(dt) -> Diwedge(dt)
    | Wgraph(dt) -> Wedge(dt)
    | Digraph(dt) -> Edge(dt)
    | _ -> raise (Failure("type " ^ string_of_typ graph_type ^
      " may not be iterated with for_edge"); (* TODO: write
a test for this *)
  in
  let flocals = StringMap.add str edge_type env.env_flocals in
  let new_env =
  {
    env_name = env.env_name;
    env_return_type = env.env_return_type;
    env fmap = env.env fmap;
    env sfmap = env.env sfmap;
    env_globals = env.env globals;
    env flocals = flocals;
    env_fformals = env.env_fformals;
    env_in_loop = env.env_in_loop;
  }
in
  let (se, senv) = convert_expr e new_env in
  let gname = match se with (* cannot call the following methods on a NAMED
  graph *)
    (* unnamed graph is safe since it cannot modify the graph itself *)
    SId(str, Graph(_)) -> str
    | SId(str, Digraph(_)) -> str
    | SId(str, Wedigraph(_)) -> str
    | SId(str, Wgraph(_)) -> str
    | _ -> "" (* not sure if empty string is problematic *)
  in
  let (for_body, senv) = convert_stmt s senv in
  let chk = match for_body with
    SBlock(lst) -> match lst with
    [] -> ();
    _ -> (* non empty statement lst, check em *)
    let chkcall x = match x with
      SExpr(SMethod(g, fname, _, _), _) -> (match g, fname with
        SId(s, _), "add_edge" -> if s = gname then
          report_concurrent_mod "for_edge"
        | SId(s, _), "remove_edge" -> if s = gname then
          report_concurrent_mod "for_edge"
        | SId(s, _), "add_node" -> if s = gname then
          report_concurrent_mod "for_edge"
        | SId(s, _), "remove_node" -> if s = gname then
          report_concurrent_mod "for_edge"
        | _ -> ();)
    in

51
List.iter chkcall lst

| _ -> ()
in
let nenv =
{
  env_name = env.env_name;
  env_return_type = env.env_return_type;
  env_fmap = env.env_fmap;
  env_sfmap = env.env_sfmap;
  env_globals = env.env_globals;
  env_flocals = env.env_flocals;
  env_fformals = env.env_fformals;
  env_in_loop = env.env_in_loop;
}
in
SFor_Edge(str, se, for_body),nenv

and check_bfs str e1 e2 s env =
let graph_type = get_sexpr_type (fst (convert_expr e1 env)) in
let node_type = match graph_type with
  Wedigraph(dt) | Wegraph(dt) | Graph(dt) | Digraph(dt) -> Node(dt)
| _ -> raise(Failure("type " ^ string_of_typ graph_type ^
  " may not be iterated with bfs"));
in
let flocals = StringMap.add str node_type env.env_flocals in
let new_env =
{
  env_name = env.env_name;
  env_return_type = env.env_return_type;
  env_fmap = env.env_fmap;
  env_sfmap = env.env_sfmap;
  env_globals = env.env_globals;
  env_flocals = flocals;
  env_fformals = env.env_fformals;
  env_in_loop = env.env_in_loop;
}
in
let (se1, senv1) = convert_expr e1 new_env in
let (se2, senv2) = convert_expr e2 senv1 in
let source_type = get_sexpr_type se2 in
(match source_type with
  Node(_) -> ()
| _ -> raise(Failure("source expr in bfs must be of type node")));
let gname = match se1 with (* cannot call the following methods on a NAMED graph*)
  SID(str, Graph(_)) -> str
| SID(str, Digraph(_)) -> str
| SID(str, Wedigraph(_)) -> str
| SID(str, Wegraph(_)) -> str
| _ -> "" (*not sure if empty string is problematic *)
in
let (bfs_body, senv2) = convert_stmt s senv2 in
let chk = match bfs_body with
  SBlock(lst) -> match lst with
  [] -> ();
| _ -> (* non empty statement lst, check em*)

52
let chkcall x = match x with
  SExpr(SMethod(g, fname,_,_),_) -> (match g, fname with
    SId(s,_) -> "add_edge" -> if s = gname then
      report_concurrent_mod "bfs"
    | SId(s,_) -> "remove_edge" -> if s = gname then
      report_concurrent_mod "bfs"
    | SId(s,_) -> "add_node" -> if s = gname then
      report_concurrent_mod "bfs"
    | SId(s,_) -> "remove_node" -> if s = gname then
      report_concurrent_mod "bfs"
    | _ -> ()
  | _ -> ();
) in List.iter chkcall lst

let nenv =
  {
    env_name = env.env_name;
    env_return_type = env.env_return_type;
    env_fmap = env.env_fmap;
    env_sfmap = senv2.env_sfmap;
    envGlobals = env.envGlobals;
    env_flocals = env.env_flocals;
    env_fformals = env.env_fformals;
    env_in_loop = env.env_in_loop;
  }

in SBfs(str, se1, se2, bfs_body), nenv

and check_dfs str e1 e2 s env =
  let graph_type = get_sexpr_type (fst (convert_expr e1 env)) in
  let node_type = match graph_type with
    Wedigraph(dt) | Wegraph(dt) | Graph(dt) | Digraph(dt) -> Node(dt)
    | _ -> raise (Failure("type " ^ string_of_typ graph_type ^
                          " may not be iterated with dfs"));
  in let flocals = StringMap.add str node_type env.env_flocals in
  let new_env =
    {
      env_name = env.env_name;
      env_return_type = env.env_return_type;
      env_fmap = env.env_fmap;
      env_sfmap = env.env_sfmap;
      envGlobals = env.envGlobals;
      env_flocals = flocals;
      env_fformals = env.env_fformals;
      env_in_loop = env.env_in_loop;
    }

  in let (se1, senv1) = convert_expr e1 new_env in
    let (se2, senv2) = convert_expr e2 senv1 in
    let source_type = get_sexpr_type se2 in
    (match source_type with
      Node( _) -> ()
    | _ -> raise(Failure("source expr in dfs must be of type node")));
let gname = match se1 with (* cannot call the following methods on a NAMED graph *)
  | SId(str, Graph(_)) -> str
  | SId(str, Digraph(_)) -> str
  | SId(str, Wedigraph(_)) -> str
  | SId(str, Wegraph(_)) -> str
  | _ -> "" (*not sure if empty string is problematic *)
in
let (dfs_body, senv2) = convert_stmt s senv2 in
let chk = match dfs_body with
  | SBlock(lst) -> match lst with
    | [] -> ();
    | _ -> (* non empty statement lst, check em*)
    let chkcall x = match x with
      | SExpr(SMethod(g, fname, _, _), _) -> (match g, fname with
        | SId(s, _) , "add_edge" -> if s = gname then
          report_concurrent_mod "dfs"
        | SId(s, _) , "remove_edge" -> if s = gname then
          report_concurrent_mod "dfs"
        | SId(s, _) , "add_node" -> if s = gname then
          report_concurrent_mod "dfs"
        | SId(s, _) , "remove_node" -> if s = gname then
          report_concurrent_mod "dfs"
        | _ -> ()
      )
    in
  in
  List.iter chkcall lst
in
let nenv =
{ env_name = env.env_name;
  env_return_type = env.env_return_type;
  env_fmap = env.env_fmap;
  env_sfmap = senv2.env_sfmap;
  env_globals = env.env_globals;
  env_flocals = env.env_flocals;
  env_fformals = env.env_fformals;
  env_in_loop = env.env_in_loop;
}
in
SDfs(str, se1, se2, dfs_body), nenv

and check_break env =
  if env.env_in_loop then
    SBreak
  else raise(Failure("can’t break outside of a loop"))

and check_continue env =
  if env.env_in_loop then
    SContinue
  else raise(Failure("can’t continue outside of a loop"))

and check_expr_stmt e env =
  let (se, nenv) = convert_expr e env in
let typ = get_sexpr_type se in (SExpr(se, typ), nenv)

and convert_fdecl fname fformals env =
  let fdecl = StringMap.find fname env.env_fmap in
  report_duplicate (fun n -> match n with
    _, str -> "duplicate fformal " ^ str) fformals;

  let formals_to_map m formal =
    match formal with
    (t, str) -> match t with
      Void -> raise (Failure("cannot declare " ^ str ^ " as type void"))
    | Map(Void) | Graph(Void) | Digraph(Void) | Wegraph(Void) | Wedigraph(Void) ->
      raise (Failure("cannot have formal with type " ^ string_of_typ t))
    | _ -> StringMap.add str t m

  in

  let formals = List.fold_left formals_to_map StringMap.empty fformals in

  let _ = match fdecl.f_typ with
    Map(Void) | Graph(Void) | Digraph(Void) | Wegraph(Void) | Wedigraph(Void) ->
      raise (Failure("cannot return " ^ string_of_typ fdecl.f_typ))
    | _ -> ()

  in

  let env = {
    env_name = fname;
    env_return_type = fdecl.f_typ;
    env_fmap = env.env_fmap;
    env_sfmap = env.env_sfmap;
    env_globals = env.env_globals;
    env_flocals = StringMap.empty; (* locals should be empty at start check*)
    env_fformals = formals;
    env_in_loop = env.env_in_loop
  }

  in

  (* Semantically check all statements of the body *)
  let (stmts, nenv) = convert_stmt (Block fdecl.f_body) env
  in

  let check_ret = match stmts with
    SBlock(lst) -> match lst with
      [] -> raise (Failure("missing return in " ^ fdecl.f_name));
    | _ -> ()

    let rets = List.filter (fun x -> match x with
      SReturn(expr) -> true
    | _ -> false ) lst in

    if List.length rets > 1 then
      raise (Failure("misplaced return in " ^ fdecl.f_name))
    else
      match (List.nth lst ((List.length lst)-1)) with
      SReturn(sexpr) -> ()
let _ -> raise(Failure("missing return in " ^ fdecl.f_name))

let sfdecl = {
  sf_typ = env.env_return_type;
  sf_name = fdecl.f_name;
  sf_formals = fformals; (* skips check? *)
  sf_body = match sstmts with SBlock(s1) -> s1 | _ -> [] ;
}

let new_env = {
  env_name = fname;
  env_return_type = fdecl.f_typ;
  env_fmap = env.env_fmap;
  env_sfmap = StringMap.add fname sfdecl nenv.env_sfmap;
  env_globals = env.env_globals;
  env_flocals = env.env_flocals;
  env_fformals = formals;
  env_in_loop = env.env_in_loop
}

let new_env =

and check_vdecl t str e from_graph_lit env =
  let t_is_node = match t with Node(_) -> true | _ -> false in
  if (StringMap.mem str env.env_flocals || StringMap.mem str env.env_fformals
  || StringMap.mem str env.env_globals)
    then
      (* if this vdecl is from a graph literal and we’ve already declared str
         as this type of node, this is fine - otherwise, reject *)
      if (from_graph_lit && t_is_node) then
        (if (StringMap.mem str env.env_flocals) then
          (if (StringMap.find str env.env_flocals <> t) then
            raise(Failure("cannot reinitialize existing variable"))
          else raise(Failure("cannot reinitialize existing variable")))
        else (if (StringMap.mem str env.env_fformals) then
          (if (StringMap.find str env.env_fformals <> t) then
            raise(Failure("cannot reinitialize existing variable"))
          else raise(Failure("cannot reinitialize existing variable")))
        else raise(Failure("cannot reinitialize existing variable")))
    else raise(Failure("cannot reinitialize existing variable"));

let _ = match t with
  Void -> raise(Failure("cannot declare " ^ str ^ " as type void"))
  | Map(Void) | Graph(Void) | Digraph(Void) | Wegraph(Void) | Wedigraph(Void)
    -> raise(Failure("cannot declare variable " ^ str ^ " with type " ^
                    string_of_typ t))
  | _ -> ()
in

let flocals = StringMap.add str t env.env_flocals in
let new_env =
  { env_name = env.env_name;
    env_return_type = env.env_return_type;
    env_fmap = env.env_fmap;
env_sfmap = env.env_sfmap;
envGlobals = env.envGlobals;
env_flocals = flocals;
env_fformals = env.env_fformals;
env_in_loop = env.env_in_loop;
}

let (se, nenv) = convert_expr e new_env
in
let typ = get_sexpr_type se
in
if typ <> Void then
  (if t <> typ then (raise (Failure("expression type mismatch " ^
string_of_typ t ~ " and " ~ string_of_typ typ)))
  else (SVdecl(t, str, se), nenv))
else
  (SVdecl(t, str, se), nenv)

and check_return e env =
let se, new_env = convert_expr e env in
let typ = get_sexpr_type se in
if typ = env.env_return_type then
  let nenv =
  {
    env_name = env.env_name;
    env_return_type = env.env_return_type;
    env_fmap = env.env_fmap;
    env_sfmap = new_env.env_sfmap;
    envGlobals = env.envGlobals;
    env_flocals = env.env_flocals;
    env_fformals = env.env_fformals;
    env_in_loop = env.env_in_loop;
  }
  in
  SReturn (se), nenv
else raise (Failure("expected return type " ^ string_of_typ env.
env_return_type
  ~ " but got return type " ^ string_of_typ typ))

let convert_ast globals fdecls fmap =
let _ = try StringMap.find "main" fmap with
  Not_found -> raise (Failure("missing main"))
in
report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

let convert_globals m global =
match global with
  (typ, str) -> if (typ <> Void) then StringMap.add str typ m
  else raise (Failure("global " ^ str " cannot have a void type"))
in
(* semantically checked globals in a map *)
let globals_map = List.fold_left convert_globals StringMap.empty globals
1128 (* check for duplicate functions *)
1129 report_duplicate ( fun f -> "duplicate function " ^ f.f_name ) fdecls;
1130
1131 (*List.iter (fun x -> convert_fdecl x x.f_name env) fdecls; *)
1132
1133 let env = {
1134   env_name = "main";
1135   env_return_type = Int;
1136   env_fmap = fmap;
1137   env_sfmap = StringMap.empty;
1138   env Globals = globals_map;
1139   env_flocals = StringMap.empty;
1140   env_fformals = StringMap.empty;
1141   env_flocals = false;
1142 }
1143
1144 (* this is the environment with all the sfdecls, stemming from main *)
1145 let sfdecl_env = convert_fdecl "main" [] env in
1146 let sfdecls = List.rev(List.fold_left (fun lst (_, sfdecl ) -> sfdecl :: lst)
1147   [] (StringMap.bindings sfdecl_env.env_sfmap))
1148 in (globals, sfdecls)
1149
1150 let build_fmap fdecls =
1151 (* built in *)
1152 let built_in_decls = StringMap.add "print"
1153   { f_typ = Void; f_name = "print"; f_formals = [(Int, "x")]
1154     f_body = [] } (StringMap.add "printb"
1155     { f_typ = Void; f_name = "printb"; f_formals = [(Bool, "x")]
1156     f_body = [] } (StringMap.singleton "prints"
1157     { f_typ = Void; f_name = "prints"; f_formals = [(String, "x")]
1158       f_body = [] } ))
1159 in
1160
1161 let check_fdecls map fdecl =
1162   if StringMap.mem fdecl.f_name map then
1163     raise (Failure ("duplicate function " ^ fdecl.f_name))
1164   else if StringMap.mem fdecl.f_name built_in_decls then
1165     raise (Failure ("reserved function name " ^ fdecl.f_name))
1166   else StringMap.add fdecl.f_name fdecl map
1167 in
1168 List.fold_left (fun map fdecl -> check_fdecls map fdecl) built_in_decls
1169 fdecls
1170
1171 let check ast = match ast with
1172 (globals, fdecls) ->
1173 let fmap = build_fmap fdecls in
1174 let sast = convert_ast globals fdecls fmap
1175 in
1176 sast

Semantic Checking: semant.ml
(* Authors: *)
Jennifer Bi jb3495
Jessie Liu jll2219
*)
open Ast
open Sast

(* Raise an error if the given list has a duplicate *)
let report_duplicate exceptf lst =
  let rec helper = function
    n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
    | _ :: t -> helper t
    | [] -> ()
  in helper (List.sort compare lst)

and report_undeclared_id_assign str =
  raise (Failure ("assign to undeclared identifier " ^ str))

(* prob will want to change these to actual exceptions *)
and report_bad_binop t1 op t2 =
  raise (Failure ("illegal binary operator " ^
    string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
    string_of_typ t2))

and report_bad_unop uop t =
  raise (Failure ("illegal unary operator " ^
    string_of_uop uop ^ " " ^ string_of_typ t))

and report_bad_assign lt rt =
  raise (Failure ("illegal assignment " ^
    string_of_typ lt ^ " = " ^ string_of_typ rt))

and check_not_void exceptf = function
  (Void, n) -> raise (Failure (exceptf n))
  | _ -> ()

and report_function_not_found func =
  raise (Failure ("function " ^ "not found"))

and report_num_args args provided =
  raise (Failure ("expected " ^ string_of_int args ^ "arguments when " ^
    string_of_int provided ^ "arguments were provided"))

and report_typ_args ot nt =
  raise (Failure ("expected " ^ string_of_typ ot ^ " when " ^
    string_of_typ nt ^ "type was provided"))

and report_meth_not_found s2 =
  raise (Failure ("method " ^ s2 ^ "not found"))

and report_concurrent_mod s =
  raise(Failure("concurrent modification of graph in " ^ s))

Semantic Checking: exception.ml
8.4 Code Generation

(* Authors:  
Daniel Benett deb2174  
Seth Benjamin sjb2190  
Jennifer Bi jb3495  
Jessie Liu jll2219  
*)

module L = Llvm
module A = Ast
module S = Sast

module StringMap = Map.Make(String)

let translate (globals, functions) =
    let context = L.global_context () in
    let the_module = L.create_module context "Giraph"
    and i32_t = L.i32_type context
    and i8_t = L.i8_type context
    and i1_t = L.i1_type context
    and str_t = L.pointer_type (L.i8_type context)
    and float_t = L.float_type context
    and void_t = L.void_type context
    and void_ptr_t = L.pointer_type (L.void_type context)
    and i32_ptr_t = L.pointer_type (L.i32_type context) in

    let ltype_of_typ = function
        A.Int -> i32_t
        | A.Bool -> i1_t
        | A.Float -> float_t
        | A.String -> str_t
        | A.Node(_) -> void_ptr_t
        | A.Graph(_) -> void_ptr_t
        | A.Digraph(_) -> void_ptr_t
        | A.Wedigraph(_) -> void_ptr_t
        | A.Edge(_) -> void_ptr_t
        | A.Wedge(_) -> void_ptr_t
        | A.Diedge(_) -> void_ptr_t
        | A.Map(_) -> void_ptr_t
        | A.Void -> void_t
 (* TODO: add wedge, handle generics *)
 in

    let get_sexpr_type sexpr = match sexpr with
        S.SId(_, typ) -> typ
        | S.SBinop(_, _, _, typ) -> typ
        | S.SUnop(_, _, typ) -> typ
        | S.SAssign(_, _, typ) -> typ
        | S.SCall(_, _, typ) -> typ
        | S.SMethod(_,_,_, typ) -> typ
        | S.SBool_Lit(_) -> Bool
        | S.SInt_Lit(_) -> Int
        | S.SFloat_Lit(_) -> Float
        | S.SString_Lit(_) -> String

let global_vars =
  let global_var m (t, n) =
    let init = L. const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty globals

(* Declare printf(), which the print built-in function will call *)
let printf_t = L.var_arg_function_type i32_t [L.pointer_type i8_t] in
let printf_func = L.declare_function "printf" printf_t the_module in

(* Declare functions that will be called to construct graphs *)
let new_graph_t = L.function_type void_ptr_t [||] in
let new_graph_func = L.declare_function "new_graph" new_graph_t the_module in

let add_vertex_t = L.function_type void_ptr_t [||] in
let add_vertex_func = L.declare_function "add_vertex" add_vertex_t the_module in

let add_edge_t = L.function_type void_t [||] in
let add_edge_func = L.declare_function "add_edge" add_edge_t the_module in

let add_wedge_t = L.function_type void_t [||] in
let add_wedge_func = L.declare_function "add_wedge" add_wedge_t the_module in

let new_data_t = L.function_type void_ptr_t [||] in
let new_data_func = L.declare_function "new_data" new_data_t the_module in

let set_data_int_t = L.function_type void_t [||] in
let set_data_int_func = L.declare_function "set_data_int" set_data_int_t the_module in

let set_data_float_t = L.function_type void_t [||] in
let set_data_float_func = L.declare_function "set_data_float" set_data_float_t the_module in

let set_data_char_ptr_t = L.function_type void_t [||] in
let set_data_char_ptr_func = L.declare_function "set_data_char_ptr" set_data_char_ptr_t the_module in

let set_data_void_ptr_t = L.function_type void_t [||] in
let set_data_void_ptr_func = L.declare_function "set_data_void_ptr" set_data_void_ptr_t the_module in

let get_data_int_t = L.function_type i32_t [||] in
let get_data_int_func = L.declare_function "get_data_int" get_data_int_t the_module in

let get_data_float_t = L.function_type float_t [||] in
let get_data_float_func = L.declare_function "get_data_float" get_data_float_t the_module in
let get_data_char_ptr_t = L.function_type str_t [] void_ptr_t in
let get_data_char_ptr_func = L.declare_function "get_data_char_ptr"
  get_data_char_ptr_t the_module in

let get_data_void_ptr_t = L.function_type void_ptr_t [] void_ptr_t in
let get_data_void_ptr_func = L.declare_function "get_data_void_ptr"
  get_data_void_ptr_t the_module in

(* Declare functions that will be called for for_node *)
let num_vertices_t = L.function_type i32_t [] void_ptr_t in
let num_vertices_func = L.declare_function "num_vertices" num_vertices_t
  the_module in

let get_head_vertex_t = L.function_type void_ptr_t [] void_ptr_t in
let get_head_vertex_func = L.declare_function "get_head_vertex"
  get_head_vertex_t the_module in

let get_next_vertex_t = L.function_type void_ptr_t [] void_ptr_t in
let get_next_vertex_func = L.declare_function "get_next_vertex"
  get_next_vertex_t the_module in

let get_data_from_vertex_t = L.function_type void_ptr_t [] void_ptr_t in
let get_data_from_vertex_func = L.declare_function "get_data_from_vertex"
  get_data_from_vertex_t the_module in

(* Declare functions corresponding to graph methods *)
let add_vertex_if_not_t = L.function_type void_t [] void_ptr_t ; void_ptr_t in
let add_vertex_if_not_func = L.declare_function "add_vertex_if_not_present"
  add_vertex_if_not_t the_module in

let remove_vertex_t = L.function_type void_t [] void_ptr_t ; void_ptr_t in
let remove_vertex_func = L.declare_function "remove_vertex" remove_vertex_t
  the_module in

let has_vertex_t = L.function_type i32_t [] void_ptr_t ; void_ptr_t in
let has_vertex_func = L.declare_function "has_vertex" has_vertex_t the_module in

let add_edge_method_t = L.function_type void_t [] void_ptr_t ; void_ptr_t ;
void_ptr_t in
let add_edge_method_func = L.declare_function "add_edge_method"
  add_edge_method_t the_module in

let add_wedge_method_t = L.function_type void_t [] void_ptr_t ; void_ptr_t ;
void_ptr_t ; i32_t in
let add_wedge_method_func = L.declare_function "add_wedge_method"
  add_wedge_method_t the_module in

let remove_edge_t = L.function_type void_t [] void_ptr_t ; void_ptr_t ;
void_ptr_t in
let remove_edge_func = L.declare_function "remove_edge" remove_edge_t
  the_module in

let has_edge_t = L.function_type i32_t [] void_ptr_t ; void_ptr_t ; void_ptr_t
let has_edge_func = L.declare_function "has_edge" has_edge_t the_module in

let graph_neighbors_t = L.function_type void_ptr_t [ | void_ptr_t ; void_ptr_t | ] in
let graph_neighbors_func = L.declare_function "graph_neighbors"
  graph_neighbors_t the_module in

let get_edge_weight_t = L.function_type i32_t [ | void_ptr_t ; void_ptr_t ;
  void_ptr_t | ] in
let get_edge_weight_func = L.declare_function "graph_get_edge_weight"
  get_edge_weight_t the_module in

let set_edge_weight_t = L.function_type void_t [ | void_ptr_t ; void_ptr_t ;
  void_ptr_t ; i32_t | ] in
let set_edge_weight_func = L.declare_function "graph_set_edge_weight"
  set_edge_weight_t the_module in

let set_undirected_edge_weight_t = L.function_type void_t [ | void_ptr_t ;
  void_ptr_t ; void_ptr_t ; i32_t | ] in
let set_undirected_edge_weight_func = L.declare_function "
  graph_set_undirected_edge_weight" set_undirected_edge_weight_t the_module in

let print_int_t = L.function_type void_t [ | void_ptr_t | ] in
let print_int_func = L.declare_function "print_int" print_int_t the_module in

let print_float_t = L.function_type void_t [ | void_ptr_t | ] in
let print_float_func = L.declare_function "print_float" print_float_t
  the_module in

let print_bool_t = L.function_type void_t [ | void_ptr_t | ] in
let print_bool_func = L.declare_function "print_bool" print_bool_t the_module in

let print_char_ptr_t = L.function_type void_t [ | void_ptr_t | ] in
let print_char_ptr_func = L.declare_function "print_char_ptr" print_char_ptr_t
  the_module in

let print_unweighted_int_t = L.function_type void_t [ | void_ptr_t | ] in
let print_unweighted_int_func = L.declare_function "print_unweighted_int"
  print_unweighted_int_t the_module in

let print_unweighted_float_t = L.function_type void_t [ | void_ptr_t | ] in
let print_unweighted_float_func = L.declare_function "print_unweighted_float"
  print_unweighted_float_t the_module in

let print_unweighted_char_ptr_t = L.function_type void_t [ | void_ptr_t | ] in
let print_unweighted_char_ptr_func = L.declare_function "
  print_unweighted_char_ptr" print_unweighted_char_ptr_t the_module in

let print_unweighted_bool_t = L.function_type void_t [ | void_ptr_t | ] in
let print_unweighted_bool_func = L.declare_function "print_unweighted_bool"
  print_unweighted_bool_t the_module in

(* Declare functions that will be called for bfs and dfs on graphs*)
let find_vertex_t = L.function_type void_ptr_t [ | void_ptr_t ; void_ptr_t | ] in
let find_vertex_func = L.declare_function "find_vertex" find_vertex_t
  the_module in
let get_visited_array_t = L. function_type void_ptr_t [| void_ptr_t |] in
let get_visited_array_func = L.declare_function "get_visited_array"
    get_visited_array_t the_module in

let get_bfs_queue_t = L. function_type void_ptr_t [| void_ptr_t ; void_ptr_t |] in
let get_bfs_queue_func = L.declare_function "get_bfs_queue" get_bfs_queue_t
    the_module in

let get_dfs_stack_t = L. function_type void_ptr_t [| void_ptr_t ; void_ptr_t |] in
let get_dfs_stack_func = L.declare_function "get_dfs_stack" get_dfs_stack_t
    the_module in

let get_next_bfs_vertex_t = L. function_type void_ptr_t [| void_ptr_t ;
    void_ptr_t |] in
let get_next_bfs_vertex_func = L.declare_function "get_next_bfs_vertex"
    get_next_bfs_vertex_t the_module in

let get_next_dfs_vertex_t = L. function_type void_ptr_t [| void_ptr_t ;
    void_ptr_t |] in
let get_next_dfs_vertex_func = L.declare_function "get_next_dfs_vertex"
    get_next_dfs_vertex_t the_module in

let bfs_done_t = L. function_type i32_t [| void_ptr_t |] in
let bfs_done_func = L.declare_function "bfs_done" bfs_done_t the_module in

let dfs_done_t = L. function_type i32_t [| void_ptr_t |] in
let dfs_done_func = L.declare_function "dfs_done" dfs_done_t the_module in

(* Declare functions that will be used for edge creation and for_edge *)
let edge_from_t = L. function_type void_ptr_t [| void_ptr_t |] in
let edge_from_func = L.declare_function "edge_from" edge_from_t the_module in

let edge_to_t = L. function_type void_ptr_t [| void_ptr_t |] in
let edge_to_func = L.declare_function "edge_to" edge_to_t the_module in

let edge_weight_t = L. function_type i32_t [| void_ptr_t |] in
let edge_weight_func = L.declare_function "edge_weight" edge_weight_t
    the_module in

let edge_set_weight_t = L. function_type void_t [| void_ptr_t ; i32_t |] in
let edge_set_weight_func = L.declare_function "edge_set_weight"
    edge_set_weight_t the_module in

let undirected_edge_set_weight_t = L. function_type void_t [| void_ptr_t ; i32_t |
    i32_t |] in
let undirected_edge_set_weight_func = L.declare_function "undirected_edge_set_weight"
    undirected_edge_set_weight_t the_module in

let construct_edge_list_t = L. function_type void_ptr_t [| void_ptr_t |] in
let construct_edge_list_func = L.declare_function "construct_edge_list"
    construct_edge_list_t the_module in

let construct_undirected_edge_list_t = L. function_type void_ptr_t [| void_ptr_t |] in
let construct_undirected_edge_list_func = L.declare_function "construct_undirected_edge_list" construct_undirected_edge_list_t the_module in

let num_edges_t = L.function_type i32_t [| void_ptr_t |] in
let num_edges_func = L.declare_function "num_edges" num_edges_t the_module in

let get_next_edge_t = L.function_type void_ptr_t [| void_ptr_t |] in
let get_next_edge_func = L.declare_function "get_next_edge" get_next_edge_t the_module in

(* Declare functions that will be called for maps *)
let make_map_t = L.function_type void_ptr_t [||] in
let make_map_func = L.declare_function "make_map" make_map_t the_module in

let map_contains_t = L.function_type i32_t [| void_ptr_t ; void_ptr_t |] in
let map_contains_func = L.declare_function "contains_key" map_contains_t the_module in

let map_put_void_ptr_t = L.function_type void_t [| void_ptr_t ; void_ptr_t ; void_ptr_t |] in
let map_put_void_ptr_func = L.declare_function "put" map_put_void_ptr_t the_module in

let map_put_int_t = L.function_type void_t [| void_ptr_t ; void_ptr_t ; i32_t |] in
let map_put_int_func = L.declare_function "put_int" map_put_int_t the_module in

(* TODO: remove when nodes become generic *)
(* let map_put_int_ptr_t = L.function_type void_t [| void_ptr_t ; void_ptr_t ; i32_ptr_t |] in
let map_put_int_ptr_func = L.declare_function "put_int_ptr" map_put_int_ptr_t the_module in *)

let map_put_char_ptr_t = L.function_type void_t [| void_ptr_t ; void_ptr_t ; str_t |] in
let map_put_char_ptr_func = L.declare_function "put_char_ptr" map_put_char_ptr_t the_module in

let map_put_float_t = L.function_type void_t [| void_ptr_t ; void_ptr_t ; float_t |] in
let map_put_float_func = L.declare_function "put_float" map_put_float_t the_module in

let map_get_void_ptr_t = L.function_type void_ptr_t [| void_ptr_t ; void_ptr_t |] in
let map_get_void_ptr_func = L.declare_function "get" map_get_void_ptr_t the_module in

let map_get_int_t = L.function_type i32_t [| void_ptr_t ; void_ptr_t |] in
let map_get_int_func = L.declare_function "get_int" map_get_int_t the_module in

(* TODO: remove when nodes become generic *)
(* let map_get_int_ptr_t = L.function_type i32_ptr_t [| void_ptr_t ; void_ptr_t |] in
let map_get_int_ptr_func = L.declare_function "get_int_ptr" map_get_int_ptr_t the_module in *)
let map_get_char_ptr_t = L.function_type str_t [| void_ptr_t ; void_ptr_t |] in
let map_get_char_ptr_func = L.declare_function "get_char_ptr"
mmap_get_char_ptr_t the_module in

let map_get_float_t = L.function_type float_t [| void_ptr_t ; void_ptr_t |] in
let map_get_float_func = L.declare_function "get_float" map_get_float_t
the_module in

let function_decls =
    let function_decl m fdecl =
        let name = fdecl .S. sf_name
        and formal_types =
            Array . of_list ( List . map ( fun (t,_) -> ltype_of_typ t) fdecl .S. sf_formals)
        in let ftype = L. function_type ( ltype_of_typ fdecl .S. sf_typ) formal_types
            in
                StringMap.add name (L. define_function name ftype the_module , fdecl ) m in
        List . fold_left function_decl StringMap.empty functions in

(* Fill in the body of the given function *)
let build_function_body fdecl =
    let (the_function , _) = StringMap.find fdecl .S. sf_name function_decls in
    let builder = L. builder_at_end context (L. entry_block the_function ) in

    let int_format_str = L. build_global_stringptr "%d\n" "fmt" builder and
    string_format_str = L. build_global_stringptr "%s\n" "fmt" builder in

(* Allocate formal arguments on the stack , initialize their value , and
remember their values in a map also containing all global vars. *)
let globals_and_formals =
    let add_formal m (t, n) p = L. set_value_name n p;
    let local = L. buildalloca (ltype_of_typ t) n builder in
    ignore (L. buildstore p local builder);
    StringMap.add n local m
    in
    List . fold_left2 add_formal global_vars fdecl .S. sf_formals
    (Array . to_list (L. params the_function) ) in

(* Construct the locally declared variables for a block. Allocate each on the
stack , initialize their value , if appropriate, and remember their values in
the map m, passed as an argument. This is called every time A.Block is
processed, allowing every bracketed block to have its own scope. *)
let add_local_vars m builder sl =
    (* When initializing graphs with graph literals, nodes do not have to be declared; e.g. "graph g = [A -- B]" implicitly declares nodes A and B (unless
they have been already declared explicitly or in a previous graph). Thus,
whenever we encounter a graph literal, we have to construct all the new nodes
it uses as local variables. The following function handles this. *)
let rec add_nodes_from_graph_lits m expr = match expr with
  S.SGraph_Lit(nodes, edges, _, _, _) ->
  let add_node m node =
    if (StringMap.mem node m) then
      m
    else
      let local_node_var = L.buildalloca (ltype_of_typ (A.Node(A.Int)))
    node builder in
    let new_data_ptr = L.buildcall new_data_func ||| "tmp_data"
  builder in
    ignore(L.buildstore new_data_ptr local_node_var builder);
  StringMap.add node local_node_var m
  in
  List.fold_left add_node m nodes
  (* for any sexpr containing sub-sexprs, recursively call on those *)
  | S.SAssign(_, e, _) -> add_nodes_from_graph_lits m e
  | S.SMethod(e, _, el, _) ->
    List.fold_left (fun mp ex -> add_nodes_from_graph_lits mp ex) (add_nodes_from_graph_lits m e) el
  | S.SCall(_, el, _) -> List.fold_left add_nodes_from_graph_lits m el
  | S.SUnop(_, e, _) -> add_nodes_from_graph_lits m e
  | S.SBinop(e1, _, e2, _) -> List.fold_left add_nodes_from_graph_lits m [ e1; e2]
  | _ -> m (* ignore expressions without subexpressions *)
  in

  (* find all local variables declared in block *)
  (* TODO: does this need to be recursive? *)
  (* see: if (true) printb([A].has_node(A)) vs if (true) {printb([A].has_node(A))};
    either make recursive, or make [A].has_node(A) not work - probably the second *)
  let add_local m stmt = match stmt with
    S.SVdecl(t, n, e) ->
      let m = add_nodes_from_graph_lits m e in
      let local_var = L.buildalloca (ltype_of_typ t) n builder in
      (* if we're declaring a node (and not immediately initializing it to another node)
        we need to call new_data() from C to get a unique data pointer and store it in
        the allocated register *)
      (match t with
        A.Node(_) -> if e == S.SNoexpr then
          let new_data_ptr = L.buildcall new_data_func ||| "tmp_data"
        builder in
            ignore(L.buildstore new_data_ptr local_var builder);
        else ()
        (* Same thing if we declare a map: call make_map() to construct the map *)
        | A.Map(_) -> if e == S.SNoexpr then
          let map_ptr = L.buildcall make_map_func ||| "tmp_map" builder in
            ignore(L.buildstore map_ptr local_var builder);
else ()
| _ -> ();
(* add new variable to m *)
StringMap.add n local_var m
(* non-vdecl stmts might have graph literals, so match on those to get them *)
| S. SExpr(e, _)
| S. SIf(e, _, _)
| S. SWhile(e, _)
| S. SFor_Node(_, e, _)
| S. SFor_Edge(_, e, _)
| S. SReturn(e) -> add_nodes_from_graph_lits m e
| S. SBfs(_, e1, e2, _) -> List.fold_left add_nodes_from_graph_lits m [e1; e2]
| S. SDfs(_, e1, e2, _) -> List.fold_left add_nodes_from_graph_lits m [e1; e2; e3]
| _ -> m
in
List.fold_left add_local m sl (* return value of add_local_vars *)
in
(* Given a symbol table "vars", return the value for a variable or formal argument in the table *)
let lookup vars n = StringMap.find n vars in

(* Construct code for an expression; return its value *)
let rec expr vars builder = function
S. SInt_Lit i -> L. const_int i32_t i
| S. SBool_Lit b -> L. const_int i1_t (if b then 1 else 0)
| S. SNoexpr -> L. const_int i32_t 0
| S. SId (s, _) -> L. build_load (lookup vars s) s builder
| S. SString_Lit s -> L. build_global_stringptr s "str" builder
| S. SFloat_Lit f -> L. const_float float_t f
| S. SBinop (e1, op, e2, _) ->
    let e1' = expr vars builder e1
    and e2' = expr vars builder e2 in
    let e_type = get_sexp_type e1 in
    match e_type with
    | A. Float -> (match op with
        | A. Add -> L. build_fadd
        | A. Sub -> L. build_fsub
        | A. Mult -> L. build_fmul
        | A. Div -> L. build_fdiv
        | A. Mod -> L. build_frem (* TODO: look what this actually means *)
        | A. And -> L. build_and
        | A. Or -> L. build_or
        | A. Eq -> L. build_fcmp L. Fcmp.Eq
        | A. Neq -> L. build_fcmp L. Fcmp.Neq
        | A. Less -> L. build_fcmp L. Fcmp.Lt
        | A. Leq -> L. build_fcmp L. Fcmp.Leq
        | A. Greater -> L. build_fcmp L. Fcmp.Greater
        | A. Geq -> L. build_fcmp L. Fcmp.Geq)
    | _ -> (match op with
        | A. Add -> L. build_add

68
| A. Sub  -> L. build_sub          |
| A. Mult -> L. build_mul          |
| A. Div  -> L. build_sdiv         |
| A. Mod  -> L. build_srem         |
| A. And  -> L. build_and          |
| A. Or   -> L. build_or           |
| A. Eq   -> L. build_icmp L.Icmp.Eq|
| A. Neq  -> L. build_icmp L.Icmp.Ne|
| A. Less -> L. build_icmp L.Icmp.Slt|
| A. Leq  -> L. build_icmp L.Icmp.Sle|
| A. Greater -> L. build_icmp L.Icmp.Sgt|
| A. Geq  -> L. build_icmp L.Icmp.Sge) e1, e2, "tmp" builder)

| S. SUnop (op, e, _) -> |
| let e' = expr vars builder e in |
| (match op with |
| A. Neg     -> if ((get_sexp_type e)) = A.Float) then L.build_fneg else |
| L.build_neg |
| A. Not     -> L.build_not) e' "tmp" builder |
| S.SAssign (id, e, _) -> let e' = expr vars builder e in |
| ignore (L.build_store e' (lookup vars id) builder); e' |
| S.SGraph_Lit (nodes, edges, nodes_init, graph_subtyp, _) -> (* TODO: use |
| the last field for generics *) |
| (* create new graph struct, return pointer *) |
| let g = L.build_call new_graph_func [] "tmp" builder in |
| (* map node names to vertex_list_node pointers created by calling |
| add_vertex *) |
| let get_data_ptr node = L.build_load (lookup vars node) node builder in |
| let call_add_vertex node = L.build_call add_vertex_func [g ; ( |
| get_data_ptr node)] "vertex_struct" node builder in |
| let nodes_map = List.fold_left (fun map node -> StringMap.add node ( |
| call_add_vertex node) map) StringMap.empty nodes in |
| (* add edge *) |
| let add_edge n1 n2 = |
| L.build_call add_edge_func [StringMap.find n1 nodes_map] ; ( |
| StringMap.find n2 nodes_map) | "" builder |
| and add_wedge n1 n2 w = |
| L.build_call add_wedge_func [StringMap.find n1 nodes_map] ; ( |
| StringMap.find n2 nodes_map) ; (expr vars builder w)] | "" builder |
| in ignore (match graph_subtyp with |
| A.Wegraph (_) | A.Wedigraph (_) -> List.map (fun (n1, n2, w) -> |
| add_wedge n1 n2 w) edges |
| (* unweighted graphs *) -> List.map (fun (n1, n2, _) -> add_edge |
| n1 n2 w) edges) |
| (* initialize nodes with data *) |
| let data_type = (match graph_subtyp with A.Graph(t) | A.Digraph(t) | A. |
| Wedigraph(t) -> t |
| _ -> A.Void) in |
| let set_data type_func (node, data) = |
| L.build_call type_func [get_data_ptr node) ; (expr vars builder data |
| ) | "" builder in |
| let set_data_bool (node, data) = |
| let data_bool = L.build_intcast (expr vars builder data) i32_t " |
| tmp_intcast" builder in |
| L.build_call set_data_int_func [get_data_ptr node); data_bool | | " |
| builder |
| in |
let set_all_data = (match data_type with
  | A. Int -> List.map (set_data set_data_int_func) nodes_init
  | A. Bool -> List.map set_data_bool nodes_init
  | A. Float -> List.map (set_data set_data_float_func) nodes_init
  | A. String -> List.map (set_data set_data_char_ptr_func) nodes_init
  | _ -> List.map (set_data set_data_void_ptr_func) nodes_init)
in
if (data_type <> A.Void) then ignore(set_all_data);
(* return pointer to graph struct *)
g
| S. SCall ("print", [e], _) | S. SCall ("printb", [e], _) ->
  L.build_call printf_func [| int_format_str ; (expr vars builder e) |]
  "printf" builder
| S. SCall ("prints", [e], _) ->
  L.build_call printf_func [| string_format_str ; (expr vars builder e) |]
  "printfs" builder
| S. SCall (f, act, _) ->
  let (fdef, fdecl) = StringMap.find f function_decls in
  let actuals = List.rev (List.map (expr vars builder) (List.rev act)) in
  let result = (match fdecl.S. sf_typ with A. Void -> ""
    | _ -> f " _result") in
  L.build_call fdef (Array.of_list actuals) result builder

(* node methods *)
| S. SMethod (node_expr, "data", [], _) ->
  let data_ptr = expr vars builder node_expr in
  let data_type = (match (get_sexpr_type node_expr) with A.Node(t) -> t) in
  let get_data_func = (match data_type with
    | A. Int | A. Bool -> get_data_int_func
    | A. Float -> get_data_float_func
    | A. String -> get_data_char_ptr_func
    | _ -> get_data_void_ptr_func) in
  let ret = L.build_call get_data_func [| data_ptr |] "tmp_data" builder in
  if (data_type = A. Bool) then
    (L.build_icmp L. Icmp .Ne ret (L.const_int i32_t 0) "tmp_booldata" builder)
  else
    ret
| S. SMethod (node_expr, "set_data", [data_expr], _) ->
  let data_ptr = expr vars builder node_expr in
  let data_type = (match (get_sexpr_type node_expr) with A.Node(t) -> t) in
  let new_data = expr vars builder data_expr in
  let new_data = if (data_type = A. Bool) then
    L.build_intcast new_data i32_t "tmp_intcast" builder
  else
    new_data in
  let set_data_func = (match data_type with
    | A. Int | A. Bool -> set_data_int_func
    | A. Float -> set_data_float_func
    | A. String -> set_data_char_ptr_func
    | _ -> set_data_void_ptr_func) in
  L.build_call set_data_func [| data_ptr ; new_data |] "" builder

(* edge methods *)
| S. SMethod (edge_expr, "from", [], _) ->
  let data_ptr = expr vars builder edge_expr in
L.build_call edge_from_func [| data_ptr |] "tmp_edge_from" builder
| S.SMethod (edge_expr, "to", [], _) ->
  let data_ptr = expr vars builder edge_expr in
  L.build_call edge_to_func [| data_ptr |] "tmp_edge_to" builder
| S.SMethod (edge_expr, "weight", [], _) -> (* TODO: add sem. check so these can only be done on wedges *)
  let data_ptr = expr vars builder edge_expr in
  L.build_call edge_weight_func [| data_ptr |] "tmp_edge_weight" builder
| S.SMethod (edge_expr, "set_weight", [data], _) ->
  let data_ptr = expr vars builder edge_expr in
  let edge_type = get_sexpr_type edge_expr in
  let which_func = match edge_type with
  Diwedge (_) -> edge_set_weight_func
  | _ ->
  undirected_edge_set_weight_func in
  L.build_call which_func [| data_ptr ; (expr vars builder data) |] "" builder

(* graph methods *)
| S.SMethod (graph_expr, "print", [], _) ->
  let graph_ptr = expr vars builder graph_expr in
  let graph_type = get_sexpr_type graph_expr in
  let print_func = (match graph_type with
  A. Graph (A. Int) | A. Digraph (A. Int) -> print_unweighted_int_func
  | A. Wegraph (A. Int) | A. Wedigraph (A. Int) -> print_int_func
  | A. Graph (A. Float) | A. Digraph (A. Float) ->
  print_unweighted_float_func
  | A. Wegraph (A. Float) | A. Wedigraph (A. Float) -> print_float_func
  | A. Graph (A. String) | A. Digraph (A. String) ->
  print_unweighted_char_ptr_func
  | A. Wegraph (A. String) | A. Wedigraph (A. String) -> print_char_ptr_func
  | A. Graph (A. Bool) | A. Digraph (A. Bool) ->
  print_unweighted_bool_func
  | A. Wegraph (A. Bool) | A. Wedigraph (A. Bool) -> print_bool_func
  ) in
  L.build_call print_func [| graph_ptr |] "" builder
| S.SMethod (graph_expr, "add_node", [node_expr], _) ->
  let graph_ptr = expr vars builder graph_expr
  and data_ptr = expr vars builder node_expr in
  L.build_call add_vertex_if_not_func [| graph_ptr ; data_ptr |] "" builder
| S.SMethod (graph_expr, "remove_node", [node_expr], _) ->
  let graph_ptr = expr vars builder graph_expr
  and data_ptr = expr vars builder node_expr in
  L.build_call remove_vertex_func [| graph_ptr ; data_ptr |] "" builder
| S.SMethod (graph_expr, "has_node", [node_expr], _) ->
  let graph_ptr = expr vars builder graph_expr
  and data_ptr = expr vars builder node_expr in
  let ret = L.build_call has_vertex_func [| graph_ptr ; data_ptr |] "" builder in
  L.build_icmp L.Icmp.Eq ret (L.const_int i32_t 1) "has_node" builder
| S.SMethod (graph_expr, "add_edge", [from_node_expr ; to_node_expr], _) ->
  let graph_ptr = expr vars builder graph_expr
  and from_data_ptr = expr vars builder from_node_expr
  and to_data_ptr = expr vars builder to_node_expr in
  (* if is an undirected graph, add reverse edge as well *)
  let graph_type = get_sexpr_type graph_expr in
  (match graph_type with
  A. Graph (_) ->

```ocaml
ignore (L. build_call add_edge_method_func [| graph_ptr ; to_data_ptr ; from_data_ptr |] "" builder)
| _ -> ();
L. build_call add_edge_method_func [| graph_ptr ; from_data_ptr ; to_data_ptr |] "" builder
| S. SMethod (graph_expr, "add_edge", [from_node_expr ; to_node_expr ; weight_expr], _) ->
  let graph_ptr = expr vars builder graph_expr
  and from_data_ptr = expr vars builder from_node_expr
  and to_data_ptr = expr vars builder to_node_expr
  and weight = expr vars builder weight_expr in
  (* if is an undirected graph, add reverse edge as well *)
  let graph_type = get_sexpr_type graph_expr in
  (match graph_type with
   A. Wegraph (_) ->
     ignore (L. build_call add_wedge_method_func [| graph_ptr ; to_data_ptr ; from_data_ptr ; weight |] "" builder)
   | _ -> ());
L. build_call add_wedge_method_func [| graph_ptr ; from_data_ptr ; to_data_ptr ; weight |] "" builder
| S. SMethod (graph_expr, "remove_edge", [from_node_expr ; to_node_expr], _) ->
  let graph_ptr = expr vars builder graph_expr
  and from_data_ptr = expr vars builder from_node_expr
  and to_data_ptr = expr vars builder to_node_expr in
  (* if this is an undirected graph, remove reverse edge as well *)
  let graph_type = get_sexpr_type graph_expr in
  (match graph_type with
   A. Graph (_) | A. Wegraph (_) ->
     ignore (L. build_call remove_edge_func [| graph_ptr ; to_data_ptr ; from_data_ptr |] "" builder)
   | _ -> ());
L. build_call remove_edge_func [| graph_ptr ; from_data_ptr ; to_data_ptr ; from_data_ptr |] "" builder
| S. SMethod (graph_expr, "has_edge", [from_node_expr ; to_node_expr], _) ->
  let graph_ptr = expr vars builder graph_expr
  and from_data_ptr = expr vars builder from_node_expr
  and to_data_ptr = expr vars builder to_node_expr in
  let ret = L. build_call has_edge_func [| graph_ptr ; from_data_ptr ; to_data_ptr |] "" builder
  L. build_icall L. Icmp.Eq ret (L. const_int i32_t 1) "has_edge" builder
| S. SMethod (graph_expr, "neighbors", [hub_node], _) ->
  let graph_ptr = expr vars builder graph_expr
  and hub_data_ptr = expr vars builder hub_node in
  L. build_call graph_neighbors_func [| graph_ptr ; hub_data_ptr |] "" builder
| S. SMethod (graph_expr, "get_edge_weight", [from_node_expr ; to_node_expr], _) ->
  let graph_ptr = expr vars builder graph_expr
  and from_data_ptr = expr vars builder from_node_expr
  and to_data_ptr = expr vars builder to_node_expr in
  L. build_call get_edge_weight_func [| graph_ptr ; from_data_ptr ; to_data_ptr |] "" builder
| S. SMethod (graph_expr, "set_edge_weight", [from_node_expr ; to_node_expr ; weight_expr], _) ->
  let graph_ptr = expr vars builder graph_expr
  and from_data_ptr = expr vars builder from_node_expr
  and to_data_ptr = expr vars builder to_node_expr in
  L. build_call set_edge_weight_func [| graph_ptr ; from_data_ptr ; to_data_ptr ; weight_expr |] "" builder
```
and to_data_ptr = expr vars builder to_node_expr
and weight = expr vars builder weight_expr in
let graph_type = get SEXP type graph_expr in
let which_func = (match graph_type with
  A. Wegraph (_) -> set undirected edge weight_func
  | _ -> set edge weight_func) in
L. build call which_func [| graph_ptr ; from data_ptr ; to data_ptr ;
  weight |] "" builder

(* map methods*)
| S. SMethod (map_expr, "put", [node_expr ; value_expr], _) ->
  let map_ptr = expr vars builder map_expr
  and node_ptr = expr vars builder node_expr
  and value = expr vars builder value_expr in
  let map_type = get SEXP type map_expr in
  let value_type = (match map_type with Map (t) -> t | _ -> A. Int (* never
  happens *)) in
  let which_func = (match value_type with
    A. Int -> map put int_func
    | A. String -> map put char_ptr_func
    | A. Bool -> map put int_func
    | A. Float -> map put float_func
    | _ -> map put void_ptr_func) in
  let value = if (value_type = A. Bool) then
    L. build intcast value i32_t " tmp_intcast " builder
  else
    value
  in
  L. build call which_func [| map_ptr ; node_ptr ; value |] "" builder
| S. SMethod (map_expr, "get", [node_expr], _) ->
  let map_ptr = expr vars builder map_expr
  and node_ptr = expr vars builder node_expr in
  let map_type = get SEXP type map_expr in
  let value_type = (match map_type with Map (t) -> t | _ -> A. Int) in
  let which_func = (match value_type with
    A. Int -> map get int_func
    | A. String -> map get char_ptr_func
    | A. Bool -> map get int_func
    | A. Float -> map get float_func
    | _ -> map get void_ptr_func) in
  let ret = L. build call which_func [| map_ptr ; node_ptr |] "tmp get"
  builder in
  if (value_type = A. Bool) then
    (L. build icmp L. Icmp . Ne ret (L. const_int i32_t 0) "tmp get bool"
      builder)
  else
    ret
| S. SMethod (map_expr, "contains", [node_expr], _) ->
  let map_ptr = expr vars builder map_expr
  and node_ptr = expr vars builder node_expr in
  let ret =
    L. build call map contains_func [| map_ptr ; node_ptr |] "tmp contains"
  builder in
    L. build icmp L. Icmp . Eq ret (L. const_int i32_t 1) "contains" builder

in
(* Invoke "f builder" if the current block doesn’t already have a terminal (e.g., a branch). *)
let add_terminal builder f =
  match L.block_terminator (L.insertion_block builder) with
  Some _ -> ()
| None -> ignore (f builder) in

(* Build the code for the given statement; return the builder for the statement’s successor *)
let rec stmt vars builder = function
  S.SBlock sl -> let vars = add_local_vars vars builder sl in
  List.fold_left (stmt vars) builder sl
  | S.SExpr (e, _) -> ignore (expr vars builder e); builder
  | S.SVdecl (t, n, e) -> ignore (expr vars builder e); builder
  | S.SReturn e -> ignore (match fdecl.S.sf_typ with
    | A.Void -> L.build_ret_void builder
    | _ -> L.build_ret (expr vars builder e) builder); builder
  | S.SIf (predicate, then_stmt, else_stmt) ->
    let bool_val = expr vars builder predicate in
    let merge_bb = L.append_block context "merge" the_function in
    let then_bb = L.append_block context "then" the_function in
    add_terminal (stmt vars (L.builder_at_end context then_bb) then_stmt)
    (L.build_br merge_bb);

    let else_bb = L.append_block context "else" the_function in
    add_terminal (stmt vars (L.builder_at_end context else_bb) else_stmt)
    (L.build_br merge_bb);

    ignore (L.build_cond_br bool_val then_bb else_bb builder);
    L.builder_at_end context merge_bb
  | S.SWhile (predicate, body) ->
    let pred_bb = L.append_block context "while" the_function in
    ignore (L.build_br pred_bb builder);

    let body_bb = L.append_block context "while_body" the_function in
    add_terminal (stmt vars (L.builder_at_end context body_bb) body)
    (L.build_br pred_bb);

    let pred_builder = L.builder_at_end context pred_bb in
    let bool_val = expr vars pred_builder predicate in
    let merge_bb = L.append_block context "merge" the_function in
    ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
    L.builder_at_end context merge_bb
  | S.SBreak -> builder (*not implemented *)
  | S.SContinue -> builder (*not implemented *)
  | S.SFor (e1, e2, e3, body) ->
    stmt vars builder(S.SBlock [S.SExpr (e1, get_sexpr_type e1); S.SExpr (e2, get_sexpr_type e3)])];
  | S.SFor_Node (n, g, body) ->
    let graph_ptr = (expr vars builder g) in

    (* allocate counter variable - counts number of nodes seen so far *)
    let counter = L.build_alloca i32_t "counter" builder in
  | S.SFor (e1, e2, e3, body) ->
    S.SWhile (e2, S.SBlock [body; S.SExpr (e3, get_sexpr_type e3)])];
  | S.SFor (n, g, body) ->
    let graph_ptr = (expr vars builder g) in

    (* allocate counter variable - counts number of nodes seen so far *)
    let counter = L.build_alloca i32_t "counter" builder in

ignore(L.build_store (L.const_int i32_t 0) counter builder);
            (* get number of nodes in graph *)
            let size = L.build_call num_vertices_func [| graph_ptr |] "size" builder in
            (* allocate register for n; then, add to symbol table, so the body can access it *)
            let node_var = L.build_alloca (ltype_of_typ (A.Node(A.Int))) n builder in
            let vars = StringMap.add n node_var vars in
            (* allocate pointer to current vertex struct *)
            let current_vertex_ptr = L.build_alloca void_ptr_t "current" builder in
            (* get head of vertex list *)
            let head_vertex = L.build_call get_head_vertex_func [| graph_ptr |] "head" builder in
            ignore(L.build_store head_vertex current_vertex_ptr builder);
            let pred_bb = L.append_block context "while" the_function in
            ignore(L.build_br pred_bb builder);
            let body_bb = L.append_block context "while_body" the_function in
            let body_builder = L.builder_at_end context body_bb in
            (* load value of current vertex *)
            let current_vertex = L.build_load current_vertex_ptr "current_tmp" body_builder in
            (* get node data pointer from current vertex struct *)
            let data_ptr = L.build_call get_data_from_vertex_func [| current_vertex |] (n ^ "_tmp") body_builder in
            ignore(L.build_store data_ptr node_var body_builder);
            (* change current_vertex to be pointer to next_vertex *)
            let next_vertex = L.build_call get_next_vertex_func [| current_vertex |] "next" body_builder in
            ignore(L.build_store next_vertex current_vertex_ptr body_builder);
            (* increment counter *)
            let counter_val = L.build_load counter "counter_tmp" body_builder in
            let counter_incr = L.build_add (L.const_int i32_t 1) counter_val "counter_incr" body_builder in
            ignore(L.build_store counter_incr counter body_builder);
            (* build body of loop *)
            add_terminal (stmt vars body_builder body) (L.build_br pred_bb);
            (* branch to while_body iff counter < size *)
            let pred_builder = L.builder_at_end context pred_bb in
            let counter_val = L.build_load counter "counter_tmp" pred_builder in
            let done_bool_val = L.build_icmp L.Icmp.Slt counter_val size "done" pred_builder in
            let merge_bb = L.append_block context "merge" the_function in
            ignore (L.build_cond_br done_bool_val body_bb merge_bb pred_builder);
            L.builder_at_end context merge_bb
        | S.SFor_Edge (e, g, body) ->
            let graph_ptr = (expr vars builder g) in
            (* allocate counter variable - counts number of edges seen so far *)
            let counter = L.build_alloca i32_t_t "counter" builder in
            ignore(L.build_store (L.const_int i32_t 0) counter builder);
let edge_var = L.buildalloca (ltype_of_typ (A.Edge(A.Int))) e builder in
let vars = StringMap.add e edge_var vars in

let current_edge_ptr = L.buildalloca void_ptr_t "current" builder in

let graph_type = get_sexpr_type g in
let construct_func = (match graph_type with
  A.Digraph(_) | A.Wedigraph(_) -> construct_edge_list_func
  | _ -> constructUndirectedEdgeList_func) in
let head_edge = L.buildcall construct_func [| graph_ptr |] "head" builder in
ignore(L.buildstore head_edge current_edge_ptr builder);

let size = L.buildcall num_edges_func [| head_edge |] "size" builder in

let pred_bb = L.appendblock context "while" the_function in
ignore(L.buildbr pred_bb builder);

let body_bb = L.appendblock context "while_body" the_function in
let body_builder = L.builderatend context body_bb in

let current_edge = L.buildload current_edge_ptr "current_tmp" body_builder in

let next_edge = L.buildcall get_next_edge_func [| current_edge |] "next" body_builder in
ignore(L.buildstore next_edge current_edge_ptr body_builder);

let counter_incr = L.buildadd (L.const_int i32_t 1) counter_val "counter_incr" body_builder in
ignore(L.buildstore counter_incr counter body_builder);

let merge_bb = L.appendblock context "merge" the_function in
ignore (L.buildcond_br done_bool_val body_bb merge_bb pred_builder);

S.SBfs (n, g, r, body) ->
let graph_ptr = (expr vars builder g) in
let root_ptr = (expr vars builder r) in
(* allocate register for n; then, add to symbol table, so the body can
access it *)

let node_var = L.buildalloca (ltype_of_typ (A.Node(A.Int))) n builder in

(* add the node data pointer to symbol table, so the body can access it *)

let vars = StringMap.add n node_var vars in

(* allocate pointer to current vertex struct *)

let current_vertex_ptr = L.buildalloca void_ptr_t "current" builder in

(* get root vertex list_node for bfs search *)

let root_vertex = L.buildcall find_vertex_func [| graph_ptr ; root_ptr |
"root" builder in
ignore(L.buildstore root_vertex current_vertex_ptr builder);

let visited = L.buildcall get_visited_array_func [| graph_ptr |] "visited" builder in

let queue = L.buildcall get_bfs_queue_func [| root_vertex ; visited |] "queue" builder in

(* populate queue and visited on the root node, but do not need to save
returned vertex because current_vertex_ptr is already root_vertex during first
iteration of the loop *)
ignore(L.buildcall get_next_bfs_vertex_func [| visited ; queue |] "get_next" builder);

let pred_bb = L.appendblock context "while" the_function in
ignore(L.buildbr pred_bb builder);

let body_bb = L.appendblock context "while_body" the_function in
let body_builder = L.builderatend context body_bb in

(* load value of current vertex *)

let current_vertex = L.buildload current_vertex_ptr "current_tmp" body_builder in

(* change current_vertex to be pointer to next_vertex *)

let next_vertex = L.buildcall get_next_bfs_vertex_func [| visited ; queue |] "get_next" builder in

(* build body of loop *)
addterminal (stmt vars body_builder body) (L.buildbr pred_bb);

(* determine whether current_vertex_ptr is NULL using c bfs_done function *)

let pred_vertex = L.buildload current_vertex_ptr "pred_tmp" pred_builder in

let done_flag = L.buildcall bfs_done_func [| pred_vertex |] "done" pred_builder in

(* branch to while_body iff done_flag is 0 (i.e. if current_vertex_ptr is
not NULL *)

let done_bool_val = L.buildicmp L.Icmp.Eq done_flag(L.const_int i32_t 0) "done_pred" pred_builder in
let merge_bb = L.append_block context "merge" the_function in
ignore (L.build_cond_br done_bool_val body_bb merge_bb pred_builder);
L.builder_at_end context merge_bb
| S.SDfs (n, g, r, body) ->
let graph_ptr = (expr vars builder g) in
let root_ptr = (expr vars builder r) in

(* allocate register for n; then, add to symbol table, so the body can
access it *)
let node_var = L.buildalloca (ltype_of_typ (A.Node(A.Int))) n builder in
(* add the node data pointer to symbol table, so the body can access it *)
let vars = StringMap.add n node_var vars in
(* allocate pointer to current vertex struct *)
let current_vertex_ptr = L.buildalloca void_ptr_t "current" builder in
(* get root vertex_list_node for bfs search *)
let root_vertex = L.buildcall find_vertex_func [| graph_ptr ; root_ptr |
"root" builder in
ignore (L.buildstore root_vertex current_vertex_ptr builder);

let visited = L.buildcall get_visited_array_func [| graph_ptr |] "visited" builder in

let stack = L.buildcall get_dfs_stack_func [| root_vertex ; visited |] "stack" builder in
(* populate stack and visited on the root node, but do not need to save
returned vertex
because current_vertex_ptr is already root_vertex during first
iteration of the loop *)
ignore (L.buildcall get_next_dfs_vertex_func [| visited ; stack |] "get_next" builder);

let pred_bb = L.append_block context "while" the_function in
ignore (L.build_br pred_bb builder);

let body_bb = L.append_block context "while_body" the_function in
let body_builder = L.builder_at_end context body_bb in
(* load value of current vertex *)
let current_vertex = L.buildload current_vertex_ptr "current_tmp"
body_builder in
(* get node data pointer from current vertex struct *)
let data_ptr = L.buildcall get_data_from_vertex_func [| current_vertex |
(n "$tmp") body_builder in
ignore (L.build_store data_ptr node_var body_builder);

(* change current_vertex to be pointer to next_vertex *)
let next_vertex = L.buildcall get_next_dfs_vertex_func [| visited ;
stack |] "get_next" body_builder in
ignore (L.build_store next_vertex current_vertex_ptr body_builder);

(* build body of loop *)
add_terminal (stmt vars body_builder body) (L.build_br pred_bb);
let pred_builder = L.builder_at_end context pred_bb in
(* determine whether current_vertex_ptr is NULL using c dfs_done function *)
let pred_vertex = L. build_load current_vertex_ptr "pred_tmp" pred_builder
in
let done_flag = L. build_call dfs_done_func [| pred_vertex |] "done"
pred_builder in
(* branch to while_body iff done_flag is 0 (i.e. if current_vertex_ptr is
not NULL *)
let done_bool_val = L. build_icmp L. Icmp .Eq done_flag (L. const_int i32_t
0) "done_pred" pred_builder in

let merge_bb = L. append_block context "merge" the_function in
ignore (L. build_cond_br done_bool_val body_bb merge_bb pred_builder);
L. builder_at_end context merge_bb

(* Build the code for each statement in the function *)
let builder = stmt globals_and_formals builder (S. SBlock fdecl .S. sf_body ) in

(* Add a return if the last block falls off the end *)
add_terminal builder ( match fdecl .S. sf_typ with
| A. Void -> L. build_ret_void
| t -> L. build_ret (L. const_int ( ltype_of_typ t) 0))
in
List.iter build_function_body functions ;
the_module

Code generation: codegen.ml

(* Authors:
Daniel Benett deb2174
Seth Benjamin sjb2190
Jennifer Bi jb3495
Jessie Liu jll2219
*)

(* Top-level of the Giraph compiler: scan & parse the input,
check the resulting AST and generate SAST, generate LLVM IR,
and dump the module *)
module StringMap = Map.Make(String)
type action = Ast | LLVM_IR | Compile
let _ =
let action = ref Compile in
let set_action a () = action := a in
let speclist = [
  ("-a", Arg.Unit (set_action Ast), "Print the SAST");
  ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
  ("-c", Arg.Unit (set_action Compile),
   "Check and print the generated LLVM IR (default)");
] in
let usage_msg = "usage: ./giraph.native [-a|-l|-c] [file.gir]" in
let channel = ref stdin in
Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;
let lexbuf = Lexing.from_channel !channel in
let ast = Parser.program Scanner.token lexbuf in
let sast = Semant.check ast in
match !action with
   Ast -> (** print_string (Ast.string_of_program ast) *) print_string (Sast.
   string_of_sprogram sast)
   | LLVM_IR -> ** print_string (Llvm.string_of_llmodule (Codegen.translate sast)
   )
   | Compile -> let m = Codegen.translate sast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)

Code generation: giraph.ml

Code generation: giraph.ml

Makefile

8.5 C Libraries

/* Authors:
 Daniel Benett deb2174
 Seth Benjamin sjb2190
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* for use with generic graph/node types */
union data_type {
  int i;
  float f;
  char *s;
  void *v;
};

/* so we can cast void *'s to floats/ints in maps without triggering undefined
 behavior
 shoulda done the whole map thing with a union but hindsight is 20/20 */
union extract_float {
  float vf;
  void *vp;
};

union extract_int {
  int vi;
  void *vp;
/* Terminology-wise, we’ve painted ourselves into a corner here.

Elsewhere in this project, "node" refers to a single node in a graph.
That is NOT true in this file. In this file, "vertex" refers to a node in
a graph, and "node" refers to a single node of a linked list. */

///////////////////// STRUCTS //////////////////////

/* A single node of the adjacency list for a single vertex. */
struct adj_list_node {
    struct vertex_list_node *vertex;
    struct adj_list_node *next;
    int weight;
};

/* a single node in the edge list containing relevant edge information */
struct edge_list_node {
    struct vertex_list_node *from;
    struct vertex_list_node *to;
    struct edge_list_node *next;
    int weight;
};

/* A single vertex in a graph. */
struct vertex_list_node {
    void *data;
    struct adj_list_node *adjacencies;
    struct vertex_list_node *next;
};

/* A graph. */
struct graph {
    struct vertex_list_node *head;
};

/* node for queue */
struct queue_list_node {
    struct vertex_list_node *v;
    struct queue_list_node *next;
};

/* node for stack */
struct stack_list_node {
    struct vertex_list_node *v;
    struct stack_list_node *next;
};

/* node for map */
struct map_node {
    struct map_node *next;
    unsigned int key;
    void *value;
};

///////////////////// END STRUCTS //////////////////////
A single node of the adjacency list for a single vertex.

```c
struct map_node *get_node(unsigned int key, void *value) {
    struct map_node *out = (struct map_node *) malloc(sizeof(struct map_node));
    out->key = key;
    out->value = value;
    out->next = NULL;
    return out;
}
```

Returns pointer to map.

```c
void *make_map() {
    int default_size = 432;
    struct map_node **map = (struct map_node **) malloc(sizeof(struct map_node *) * default_size);
    memset(map, 0, sizeof(struct map_node *) * default_size);
    int *size = (int *) malloc(sizeof(int));
    *size = default_size;
    map[0] = get_node(0, size);
    return map;
}
```

Free map.

```c
void free_map(void *map_in) {
    struct map_node **map = (struct map_node **) map_in;
    int size = *((int *) map[0]->value);
    for (int i = 0; i < size; i++) {
        struct map_node *bucket = map[i];
        if (bucket) {
            struct map_node *next = bucket->next;
            free(bucket);
            while (next) {
                bucket = next;
                next = bucket->next;
                free(bucket);
            }
        }
    }
    free(map);
}
```

Hash function.

```c
int hash(unsigned int in, int size) {
    return 1 + ((in * 997) % (size - 1));
}
```

Put function.

```c
void put(void *map_in, void *key, void *value) {
    struct map_node **map = (struct map_node **) map_in;
    int size = *((int *) map[0]->value);
    unsigned int for_hash = (unsigned int) key;
    int hash_val = hash(for_hash, size);
    struct map_node *bucket = map[hash_val];
    if (!bucket) {
        map[hash_val] = get_node((unsigned int) key, value);
    }
```
return;
}
if (bucket->key == (unsigned int) key) {
  bucket->value = value;
  return;
}
while (bucket->next) {
  if (bucket->key == (unsigned int) key) {
    bucket->value = value;
    return;
  }
  bucket = bucket->next;
}
bucket->next = get_node((unsigned int) key, value);

/* returns NULL if not found */
void *get(void *map_in, void *key) {
  struct map_node **map = (struct map_node **) map_in;
  int size = *((int *) map[0]->value);
  unsigned int for_hash = (unsigned int) key;
  int hash_val = hash(for_hash, size);
  struct map_node *bucket = map[hash_val];
  if (!bucket) {
    return NULL;
  }
  while (bucket) {
    if (bucket->key == (unsigned int) key) {
      return bucket->value;
    }
    bucket = bucket->next;
  }
  return NULL;
}

int contains_key(void *map_in, void *key) {
  struct map_node **map = (struct map_node **) map_in;
  int size = *((int *) map[0]->value);
  unsigned int for_hash = (unsigned int) key;
  int hash_val = hash(for_hash, size);
  struct map_node *bucket = map[hash_val];
  if (!bucket) {
    return 0;
  }
  while (bucket) {
    if (bucket->key == (unsigned int) key) {
      return 1;
    }
    bucket = bucket->next;
  }
  return 0;
}

/* The following functions implement put() for the built-in types in giraph. */
void put_int(void *map_in, void *key, int value) {
  union extract_int ei;
void put_int_ptr(void *map_in, void *key, void *value) {
    put(map_in, key, (void *) value);
}

void put_char_ptr(void *map_in, void *key, char *value) {
    put(map_in, key, (void *) value);
}

void put_float(void *map_in, void *key, float value) {
    union extract_float ef;
    ef.vf = value;
    put(map_in, key, ef.vp);
}

/* The following functions implement get() for the built-in types in giraph. */

int get_int(void *map_in, void *key) {
    union extract_int ei;
    ei.vp = get(map_in, key);
    return ei.vi;
}

int *get_int_ptr(void *map_in, void *key) {
    return (int *) get(map_in, key);
}

char *get_char_ptr(void *map_in, void *key) {
    return (char *) get(map_in, key);
}

float get_float(void *map_in, void *key) {
    union extract_float ef;
    ef.vp = get(map_in, key);
    return ef.vf;
}

void *edge_from(void *e) {
    return ((struct edge_list_node *) e)->from->data;
}

void *edge_to(void *e) {
    return ((struct edge_list_node *) e)->to->data;
}

int edge_weight(void *e) {
255 return ((struct edge_list_node *) e)->weight;
256 }
257 }
258
259 void undirected_edge_set_weight(void *e_in, int new_weight) {
260 struct edge_list_node *e = (struct edge_list_node *) e_in;
261 struct adj_list_node *from_adj = e->from->adjacencies;
262 struct adj_list_node *to_adj = e->to->adjacencies;
263 while (from_adj) {
264 if (from_adj->vertex == e->to) {
265 from_adj->weight = new_weight;
266 }
267 from_adj = from_adj->next;
268 }
269 while (to_adj) {
270 if (to_adj->vertex == e->from) {
271 to_adj->weight = new_weight;
272 }
273 to_adj = to_adj->next;
274 }
275 ((struct edge_list_node *) e)->weight = new_weight;
276 }
277
278 void edge_set_weight(void *e_in, int new_weight) {
279 struct edge_list_node *e = (struct edge_list_node *) e_in;
280 struct adj_list_node *from_adj = e->from->adjacencies;
281 while (from_adj) {
282 if (from_adj->vertex == e->to) {
283 from_adj->weight = new_weight;
284 }
285 from_adj = from_adj->next;
286 }
287 ((struct edge_list_node *) e)->weight = new_weight;
288 }
289
290
291
292
293
294 /* Change the data stored in a node <int> data pointer. */
295 void set_data_int(void *data_ptr, int data_val) {
296 ((union data_type *)) data_ptr)->i = data_val;
297 }
298
299 /* Change the data stored in a node <float> data pointer. */
300 void set_data_float(void *data_ptr, float data_val) {
301 ((union data_type *) data_ptr)->f = data_val;
302 }
303
304 /* Change the data stored in a node <string> data pointer. */
305 void set_data_char_ptr(void *data_ptr, char *data_val) {
306 ((union data_type *) data_ptr)->s = data_val;
307 }
308
309 /* Change the data stored in a data pointer for all other node types. */
310 void set_data_void_ptr(void *data_ptr, void *data_val) {
311 /* *((void **) data_ptr) = data_val; */
((union data_type *) data_ptr)->v = data_val;
}

/* Get data stored in a node<int> data pointer. */
int get_data_int(void *data_ptr) {
    return ((union data_type *) data_ptr)->i;
}

/* Get data stored in a node<float> data pointer. */
float get_data_float(void *data_ptr) {
    return ((union data_type *) data_ptr)->f;
}

/* Get data stored in a node<string> data pointer. */
char * get_data_char_ptr(void *data_ptr) {
    return ((union data_type *) data_ptr)->s;
}

/* Get data stored in a data pointer for all other node types. */
void * get_data_void_ptr(void *data_ptr) {
    return ((union data_type *) data_ptr)->v;
}

/////////////////////////// END NODE METHODS /////////////////

/////////////////////////// GRAPH HELPER METHODS /////////////////
/* Find and return the vertex_list_node associated with a data pointer. Returns null if there is none. */
void *find_vertex(void *g_in, void *data_ptr) {
    struct graph *g = (struct graph *) g_in;
    struct vertex_list_node * vertex = g->head;
    while (vertex)
    {
        if (vertex->data == data_ptr)
        {
            return vertex;
        }
        vertex = vertex->next;
    }
    return NULL;
}

/* Returns a pointer to a new graph. */
void *new_graph() {
    struct graph *g = malloc(sizeof(struct graph));
    g->head = NULL;
    return (void *) g;
}

/* Allocate a new unique data pointer. */
void *new_data() {
    return (void *) malloc(sizeof(union data_type));
}

/* Add a new vertex to the end of the vertex list in a graph, and return a pointer to the new vertex. */
369 void *add_vertex(void *graph_ptr, void *data_ptr) {
370 struct vertex_list_node *vertex = malloc(sizeof(struct vertex_list_node));
371 vertex->data = data_ptr;
372 vertex->adjacencies = NULL;
373 vertex->next = NULL;
374
375 struct graph *g = (struct graph *)graph_ptr;
376 if (g->head == NULL) {
377     g->head = vertex;
378 } else {
379     struct vertex_list_node *last_node = g->head;
380     while (last_node->next) {
381         last_node = last_node->next;
382     }
383     last_node->next = vertex;
384 }
385
386 return (void *) vertex;
387 }
388
389 /* Add a directed, weighted edge between two vertices */
390 void add_wedge(void *from_ptr, void *to_ptr, int w) {
391 struct vertex_list_node *from = (struct vertex_list_node *) from_ptr;
392 struct vertex_list_node *to = (struct vertex_list_node *) to_ptr;
393 if (from->adjacencies == NULL) {
394     from->adjacencies = malloc(sizeof(struct adj_list_node));
395     from->adjacencies->vertex = to;
396     from->adjacencies->weight = w;
397     from->adjacencies->next = NULL;
398 } else {
399     struct adj_list_node *last_node = from->adjacencies;
400     while (last_node->next) {
401         last_node = last_node->next;
402     }
403     last_node->next = malloc(sizeof(struct adj_list_node));
404     last_node->next->vertex = to;
405     last_node->next->weight = w;
406     last_node->next->next = NULL;
407 }
408 }
409
410 /* Add a (directed) edge between two vertices. Give default weight 0. */
411 void add_edge(void *from_ptr, void *to_ptr) {
412     add_wedge(from_ptr, to_ptr, 0);
413 }
414} // END GRAPH HELPER METHODS

417} // GRAPH METHODS

419} /* Given a graph and a data pointer, checks if the graph has a vertex associated
with the data pointer, and if not, creates one and adds it to the graph.
Corresponds to add_node method in giraph. */
420 void add_vertex_if_not_present(void *g_in, void *data_ptr) {
421     struct graph *g = (struct graph *) g_in;
422     if (find_vertex(g_in, data_ptr) == NULL) {
423         add_vertex(g_in, data_ptr);
424     }
425 }
/* Given a graph and a data pointer, finds the vertex in the graph associated
with the data pointer and removes it from the vertex list and all adjacency
lists. If no such vertex exists, does nothing.
Corresponds to remove_node method in giraph. */
void remove_vertex(void *g_in, void *data_ptr) {
    struct graph *g = (struct graph *)g_in;
    struct vertex_list_node *remove = (struct vertex_list_node *)find_vertex(g_in, data_ptr);
    if (remove == NULL) {
        return;
    }

    /* Iterate through all vertices and remove this vertex from any adjacency
    lists. */
    struct vertex_list_node *vertex = g->head;
    while (vertex) {
        if (vertex->adjacencies) {
            /* if we need to remove the first adjacency, set vertex’s
            "adjacencies" pointer to be the next adjacency */
            struct adj_list_node *curr_e = vertex->adjacencies;
            if (curr_e->vertex == remove) {
                vertex->adjacencies = curr_e->next;
                free(curr_e); /* wooaaaahh */
            } else {
                /* else, just remove appropriate adj_list_node from list
                by reconnecting surrounding nodes */
                struct adj_list_node *prev_e = vertex->adjacencies;
                curr_e = prev_e->next;
                while (curr_e) {
                    if (curr_e->vertex == remove) {
                        prev_e->next = curr_e->next;
                        free(curr_e);
                        break;
                    }
                    prev_e = curr_e;
                    curr_e = curr_e->next;
                }
            }
        }
        vertex = vertex->next;
    }

    /* Remove vertex from vertex list. */
    struct vertex_list_node *curr_v = g->head;
    /* If it’s the first vertex, connect g->head to next vertex. */
    if (curr_v == remove) {
        g->head = curr_v->next;
        free(curr_v);
        return;
    }

    /* Else, remove from vertex list by reconnecting surrounding nodes. */
    struct vertex_list_node *prev_v = g->head;
curr_v = prev_v->next;
while (curr_v) {
    if (curr_v == remove) {
        prev_v->next = curr_v->next;
        free(curr_v);
        return;
    }
    prev_v = curr_v;
    curr_v = curr_v->next;
}

/* Given a graph and two data pointers, adds a directed, weighted edge between
the
vertices corresponding to each data pointer. If either of such vertices
does not exist, they are created. If the edge already exists, does nothing.
Corresponds to add_edge method in giraph. */
void add_wedge_method(void *g_in, void *from_data_ptr, void *to_data_ptr, int w)
{
    struct graph *g = (struct graph *) g_in;
    void *from = find_vertex(g_in, from_data_ptr);
    if (from == NULL) {
        from = add_vertex(g_in, from_data_ptr);
    }
    void *to = find_vertex(g_in, to_data_ptr);
    if (to == NULL) {
        to = add_vertex(g_in, to_data_ptr);
    }
    /* Check if from->to edge already exists - if so, return. */
    struct adj_list_node *curr_adj = ((struct vertex_list_node *) from)->adjacencies;
    while (curr_adj) {
        if (curr_adj->vertex == to) {
            return;
        }
        curr_adj = curr_adj->next;
    }
    add_wedge(from, to, w);
}

/* calls add_wedge_method with default weight of 0 */
void add_edge_method(void *g_in, void *from_data_ptr, void *to_data_ptr)
{
    add_wedge_method(g_in, from_data_ptr, to_data_ptr, 0);
}

/* Given a graph and two data pointers, removes the directed edge between the
vertices corresponding to each data pointer. If either of such vertices
does not exist, or if the edge does not exist, does nothing.
Corresponds to remove_edge method in giraph. */
void remove_edge(void *g_in, void *from_data_ptr, void *to_data_ptr)
{
    struct graph *g = (struct graph *) g_in;
    struct vertex_list_node *from = (struct vertex_list_node *) find_vertex(g_in, from_data_ptr);
    struct vertex_list_node *to = (struct vertex_list_node *) find_vertex(g_in, to_data_ptr);
    if (from == NULL || to == NULL) {

return;
}

/* Remove adj_list_node for "to" from adjacency list of "from" */
if (from->adjacencies) {
    /* if we need to remove the first adjacency, set from's
    "adjacencies" pointer to be the next adjacency */
    struct adj_list_node *curr = from->adjacencies;
    if (curr->vertex == to) {
        from->adjacencies = curr->next;
        free(curr);
    } else {
        /* else, just remove appropriate adj_list_node from list
           by reconnecting surrounding nodes */
        struct adj_list_node *prev = from->adjacencies;
        curr = prev->next;
        while (curr) {
            if (curr->vertex == to) {
                prev->next = curr->next;
                free(curr);
                break;
            }
            prev = curr;
            curr = curr->next;
        }
    }
}

int has_vertex(void *g_in, void * data_ptr) {
    return (find_vertex(g_in, data_ptr) != NULL);
}

/* Checks if a graph contains an edge between two vertices.
   Returns 1 if so, 0 otherwise.
   Corresponds to graph.has_edge() method in giraph. */
int has_edge(void *g_in, void *from_data_ptr, void *to_data_ptr) {
    struct graph *g = (struct graph *) g_in;
    struct vertex_list_node *from = (struct vertex_list_node *) find_vertex(g_in, from_data_ptr);
    struct vertex_list_node *to = (struct vertex_list_node *) find_vertex(g_in, to_data_ptr);
    /* If either of the vertices is not in the graph, neither is the edge. */
    if (from == NULL || to == NULL) {
        return 0;
    }
    struct adj_list_node *curr_adj = ((struct vertex_list_node *) from)->adjacencies;
    while (curr_adj) {
        if (curr_adj->vertex == to) {
            return 1;
        }
    }
curr_adj = curr_adj->next;
}
return 0;
}

/* return a graph pointer to a graph containing every neighboring vertex */
void *graph_neighbors(void *g_in, void *data_ptr) {
    struct graph *g = (struct graph *) g_in;
    struct vertex_list_node *v = find_vertex(g_in, data_ptr);
    struct graph *g_out = (struct graph *) malloc(sizeof(struct graph));
    /* if there are no adjs, or if data_ptr is not in g, return graph * with NULL
     * head */
    g_out->head = NULL;
    if (v == NULL) {
        return (void*) g_out;
    }
    struct adj_list_node *curr_adj = (struct adj_list_node *) v->adjacencies;
    if (curr_adj) {
        /* add first vertex in new graph with data in first adjacency */
        g_out->head = (struct vertex_list_node *) malloc(sizeof(struct vertex_list_node));
        struct vertex_list_node *curr_g_out = g_out->head;
        curr_g_out->next = NULL;
        curr_g_out->adjacencies = NULL;
        curr_g_out->data = curr_adj->vertex->data;
        curr_adj = curr_adj->next;
        while (curr_adj) {
            /* add all vertices in new graph with data in subsequent adjacencies */
            curr_g_out->next = (struct vertex_list_node *) malloc(sizeof(struct vertex_list_node));
            curr_g_out->next->data = curr_adj->vertex->data;
            curr_g_out->next->next = NULL;
            curr_g_out->next->adjacencies = NULL;
            curr_adj = curr_adj->next;
            curr_g_out = curr_g_out->next;
        }
        return (void*) g_out;
    }
}

int graph_get_edge_weight(void *g_in, void *from_data_ptr, void *to_data_ptr) {
    struct vertex_list_node *from = (struct vertex_list_node *) find_vertex(g_in, from_data_ptr);
    if (from == NULL) {
        return 0;
    }
    struct vertex_list_node *to = (struct vertex_list_node *) find_vertex(g_in, to_data_ptr);
    if (to == NULL) {
        return 0;
    }
    struct adj_list_node *curr_adj = from->adjacencies;
while (curr_adj) {
    if (curr_adj->vertex == to) {
        return curr_adj->weight;
    }
    curr_adj = curr_adj->next;
}
return 0;

void graph_set_undirected_edge_weight (void *g_in, void *from_data_ptr, void *to_data_ptr, int new_weight) {
    struct vertex_list_node *from = (struct vertex_list_node *) find_vertex(g_in, from_data_ptr);
    if (from == NULL) {
        return;
    }
    struct vertex_list_node *to = (struct vertex_list_node *) find_vertex(g_in, to_data_ptr);
    if (to == NULL) {
        return;
    }
    struct adj_list_node *curr_adj = from->adjacencies;
    while (curr_adj) {
        if (curr_adj->vertex == to) {
            curr_adj->weight = new_weight;
        }
        curr_adj = curr_adj->next;
    }
    curr_adj = to->adjacencies;
    while (curr_adj) {
        if (curr_adj->vertex == from) {
            curr_adj->weight = new_weight;
        }
        curr_adj = curr_adj->next;
    }
}

void graph_set_edge_weight (void *g_in, void *from_data_ptr, void *to_data_ptr, int new_weight) {
    struct vertex_list_node *from = (struct vertex_list_node *) find_vertex(g_in, from_data_ptr);
    if (from == NULL) {
        return;
    }
    struct vertex_list_node *to = (struct vertex_list_node *) find_vertex(g_in, to_data_ptr);
    if (to == NULL) {
        return;
    }
    struct adj_list_node *curr_adj = from->adjacencies;
    while (curr_adj) {
        if (curr_adj->vertex == to) {
            curr_adj->weight = new_weight;
        }
        curr_adj = curr_adj->next;
    }
}

```c
/* iterate through graph to get num vertices */
int num_vertices ( void * g_in ) {
    struct graph *g = (struct graph *) g_in ;
    struct vertex_list_node *vertex = g->head ;
    int counter = 0;
    while (vertex) {
        counter ++;
        vertex = vertex->next ;
    }
    return counter ;
}

/* return head of graph */
void * get_head_vertex ( void * g_in ) {
    struct graph *g = (struct graph *) g_in ;
    struct vertex_list_node * head = g->head ;
    return (void *) head ;
}

/* given a vertex , returns next vertex from graph 's list */
void * get_next_vertex ( void * v_in ) {
    struct vertex_list_node *v = (struct vertex_list_node *) v_in ;
    return (void *) v->next ;
}

/* return the data pointer stored in a vertex */
void * get_data_from_vertex ( void * v_in ) {
    struct vertex_list_node *v = (struct vertex_list_node *) v_in ;
    return v->data ;
}

/* construct a list of edge_list_node 's and return head */
void * construct_edge_list ( void * g_in ) {
    struct graph *g = (struct graph *) g_in ;
    struct vertex_list_node *v = g->head ;
    struct edge_list_node *head = NULL ;
    int first = 1;
    struct edge_list_node *prev ;
    while (v) {
        struct adj_list_node *adjacency = v->adjacencies ;
        while (adjacency) {
            struct edge_list_node *new_edge = (struct edge_list_node *) malloc (sizeof(struct edge_list_node));
            new_edge->source = v;
            new_edge->destination = adjacency->vertex;
            new_edge->weight = adjacency->weight;
            if (first) {
                head = new_edge;
                prev = head;
                first = 0;
            } else {
                prev->next = new_edge;
                prev = new_edge;
            }
            adjacency = adjacency->next;
        }
        v = v->next;
    }
    return head ;
}
```

struct edge_list_node *e = (struct edge_list_node *) malloc(sizeof(struct edge_list_node));
e->from = v;
e->to = adjacency->vertex;
e->weight = adjacency->weight;
e->next = NULL;
if (first) {
    head = e;
    first = 0;
} else {
    prev->next = e;
    prev = e;
    adjacency = adjacency->next;
}
v = v->next;
}
if (!head) {
    return NULL;
}
return head;

/* construct a list of edge_list_node's and return head */
void *construct_undirected_edge_list(void *g_in) {
    struct graph *g = (struct graph *) g_in;
    struct vertex_list_node *v = g->head;
    struct edge_list_node *head = NULL;
    int first = 1;
    struct edge_list_node *prev;
    void *map = make_map();
    while (v) {
        struct adj_list_node *adjacency = v->adjacencies;
        while (adjacency) {
            struct adj_list_node *to_adj_list = get(map, (void *) adjacency->vertex);
            int opposite_edge_exists = 0;
            while (to_adj_list) {
                /* if this v is not in an already edge-ified adj list */
                if (to_adj_list->vertex == v) {
                    opposite_edge_exists = 1;
                }
                to_adj_list = to_adj_list->next;
            }
            if (!opposite_edge_exists) {
                struct edge_list_node *e = (struct edge_list_node *) malloc(sizeof(struct edge_list_node));
                e->from = v;
                e->to = adjacency->vertex;
                e->weight = adjacency->weight;
                e->next = NULL;
                if (first) {
                    head = e;
                    first = 0;
                } else {
                    prev->next = e;
                }
802 }  
803 prev = e;  
804 }  
805 adjacency = adjacency->next;  
806 }  
807 put(map, (void *) v, (void *) v->adjacencies);  
808 v = v->next;  
809 }  
810 free_map(map);  
811 if (!head) {  
812 return NULL;  
813 }  
814 return head;  
815 }  
816  
817 /* return size of list of edge_list_node’s */  
818 int num_edges(void *e_head) {  
819 struct edge_list_node *e = (struct edge_list_node *) e_head;  
820 int count = 0;  
821 while (e) {  
822 count++;  
823 e = e->next;  
824 }  
825 return count;  
826 }  
827  
828 /* get next edge_list_node in list */  
829 void *get_next_edge(void *e_in) {  
830 struct edge_list_node *e = (struct edge_list_node *) e_in;  
831 return e->next;  
832 }  
833  
834 ////////////////////////////////////////////////////////////// END FOR EDGE //////////////////////////////////////////////////////////////  
835  
836 ////////////////////////////////////////////////////////////// BFS and DFS //////////////////////////////////////////////////////////////  
837  
838 /* allocate an array of vertex pointers of size (num_vertices + 1), and  
store num_vertices in a dummy vertex at the first index */  
839 void *get_visited_array(void *g_in) {  
840 int *size = malloc(sizeof(int));  
841 *size = num_vertices(g_in);  
842 struct vertex_list_node **visited =  
843 (struct vertex_list_node **) malloc(sizeof(struct vertex_list_node *) * (*size + 1));  
844 memset(visited, 0, sizeof(struct vertex_list_node *) * (*size + 1));  
845 /* store num nodes in the graph in the first entry in the array */  
846 struct vertex_list_node *dummy_size_node =  
847 (struct vertex_list_node *) malloc(sizeof(struct vertex_list_node));  
848 memset(dummy_size_node, 0, sizeof(struct vertex_list_node));  
849 dummy_size_node->data = size;  
850 visited[0] = dummy_size_node;  
851 return visited;  
852 }  
853  
854 /* check if a vertex pointer is already in the visited array */  
855 int unvisited(struct vertex_list_node *v, struct vertex_list_node **visited) {  

int size = *(int *) visited[0]->data;
for (int i = 1; i <= size; i++) {
    if (visited[i] == v) {
        return 0;
    }
    if (!visited[i]) {
        break;
    }
}
return 1;

/* add a vertex pointer to the visited array */
void add_visited(struct vertex_list_node **visited, struct vertex_list_node *v) {
    int size = *(int *) visited[0]->data;
    for (int i = 1; i <= size; i++) {
        if (!visited[i]) {
            visited[i] = v;
            return;
        }
    }
}

///////////////////// BFS ///////////////////

/* create a bfs_queue, and push the first vertex pointer onto it */
void *get_bfs_queue(void *first_v, void *visited) {
    struct queue_list_node *q = malloc(sizeof(struct queue_list_node));
    memset(q, 0, sizeof(struct queue_list_node));
    q->v = (struct vertex_list_node *) first_v;
    q->next = NULL;
    add_visited(visited, q->v);
    return q;
}

/* create a queue_list_node with given vertex pointer and add it to the back of
the queue */
void push_queue(struct vertex_list_node *vertex, struct queue_list_node *queue) {
    /* if empty */
    if (!queue->v) {
        queue->v = vertex;
        queue->next = NULL;
        return;
    }
    /* else add to end */
    while (queue->next) {
        queue = queue->next;
    }
    struct queue_list_node *new_q = (struct queue_list_node *) malloc(sizeof(struct
queue_list_node));
    memset(new_q, 0, sizeof(struct queue_list_node));
    new_q->v = vertex;
    new_q->next = NULL;
    queue->next = new_q;
}

/* pop a vertex pointer from the queue */
`void *pop_queue(struct queue_list_node *queue) {
    struct vertex_list_node *out = queue->v;
    struct queue_list_node *tofree = queue->next;
    if (queue->next) {
        queue->v = queue->next->v;
        queue->next = queue->next->next;
        free(tofree);
    } else {
        queue->v = NULL;
    }
    return out;
}

void cleanup_bfs(void *visited_in, void *queue_in) {
    struct vertex_list_node **visited = (struct vertex_list_node **) visited_in;
    struct queue_list_node *queue = (struct queue_list_node *) queue_in;
    free(visited[0]->data);
    free(visited[0]);
    free(visited_in);
    /* if empty */
    if (!queue->v) {
        free(queue);
    } /*else add to end */
    while (queue->next) {
        void *temp = queue;
        queue = queue->next;
        free(temp);
    }
}

/* get the next graph vertex in bfs order, updating visited array and bfs queue */
void *get_next_bfs_vertex(void *visited_in, void *queue) {
    struct vertex_list_node **visited = (struct vertex_list_node **) visited_in;
    struct vertex_list_node *v = pop_queue(queue);
    /* if queue empty we are done */
    if (!v) {
        cleanup_bfs(visited_in, queue);
        return NULL;
    }
    struct adj_list_node *adjacency = v->adjacencies;
    while (adjacency) {
        if (unvisited(adjacency->vertex, visited)) {
            push_queue(adjacency->vertex, queue);
            add_visited(visited, adjacency->vertex);
        }
        adjacency = adjacency->next;
    }
    return v;
}

/* test if the vertex is null to determine if bfs has finished */
int bfs_done(void *curr_v) {
    if (curr_v == NULL) {
        return 1;
    }
}`
969 }  
970 return 0;  
971 }  
972 }  
973  
974 ////////////////////////////////////////////////////////////////// DFS //////////////////////////////////////////////////////////////////  
975  
976 /* create a stack_list_node with given vertex, connect to top of stack */  
977 void push_stack(struct vertex_list_node *vertex, struct stack_list_node *s)  
978 {  
979 struct stack_list_node *new_s = (struct stack_list_node *) malloc(sizeof(struct  
980 stack_list_node));  
981 memset(new_s, 0, sizeof(struct stack_list_node));  
982 new_s->v = vertex;  
983 new_s->next = s->next;  
984 s->next = new_s;  
985 }  
986  
987 /* pop a vertex pointer from stack */  
988 void *pop_stack(struct stack_list_node *s)  
989 {  
990 if (s->next) {  
991 struct stack_list_node *t = s->next;  
992 s->next = t->next;  
993 struct vertex_list_node *out = t->v;  
994 free(t);  
995 return out;  
996 }  
997 return NULL;  
998 }  
999  
1000 /* create a dfs_stack, and push the first vertex pointer onto it */  
1001 void *get_dfs_stack(void *first_v, void *visited)  
1002 {  
1003 struct vertex_list_node *vertex = (struct vertex_list_node *) first_v;  
1004 struct stack_list_node *s_static_top = malloc(sizeof(struct stack_list_node));  
1005 memset(s_static_top, 0, sizeof(struct stack_list_node));  
1006 s_static_top->v = NULL;  
1007 push_stack((struct vertex_list_node *) vertex, s_static_top);  
1008 return s_static_top;  
1009 }  
1010  
1011 void cleanup_dfs(void *visited_in, void *stack_in)  
1012 {  
1013 struct vertex_list_node **visited = (struct vertex_list_node **) visited_in;  
1014 struct stack_list_node *stack = (struct stack_list_node *) stack_in;  
1015 free(visited[0]->data);  
1016 free(visited[0]);  
1017 free(visited_in);  
1018 while (stack->next) {  
1019 void *temp = stack;  
1020 stack = stack->next;  
1021 free(temp);  
1022 }  
1023 }  
1024  
1025 /* get the next graph vertex in bfs order, updating visited array and bfs queue */  
1026 void *get_next_dfs_vertex(void *visited_in, void *stack)  
1027 {  
1028 struct vertex_list_node **visited = (struct vertex_list_node **) visited_in;  
1029 struct vertex_list_node *v = pop_stack(stack);  
1030 while (v && unvisited(v, visited) == 0) {

v = pop_stack(stack);

} /* if stack empty we are done */
if (!v) {
    cleanup_dfs(visited_in, stack);
    return NULL;
}
add_visited(visited, v);
struct adj_list_node *adjacency = v->adjacencies;
while (adjacency) {
push_stack(adjacency->vertex, (struct stack_list_node *) stack);
    adjacency = adjacency->next;
}
return v;

} /* test if the vertex is null to determine if bfs has finished */
int dfs_done(void * curr_v) {
    if (curr_v == NULL) {
        return 1;
    }
    return 0;
}

END BFS and DFS /*------------------------*/

PRINT /*------------------------*/

void print_int(void * graph_ptr) {
    struct graph *g = (struct graph *) graph_ptr;
    struct vertex_list_node * vertex = g->head;
    while (vertex) {
        printf("vertex: %d\n", ((union data_type *) vertex->data)->i);
        printf("adjacencies: ");
        struct adj_list_node * adjacency = vertex->adjacencies;
        while (adjacency) {
            printf("(%d, weight : %d) ",
               ((union data_type *) adjacency->vertex->data)->i,
               adjacency->weight);
            adjacency = adjacency->next;
        }
        printf("\n");
        vertex = vertex->next;
    }
    printf("\n");
}

void print_float(void * graph_ptr) {
    struct graph *g = (struct graph *) graph_ptr;
    struct vertex_list_node * vertex = g->head;
    while (vertex) {
        printf("vertex: %f\n", ((union data_type *) vertex->data)->f);
        printf("adjacencies: ");
        struct adj_list_node * adjacency = vertex->adjacencies;
        while (adjacency) {
            printf("(%f, weight : %d) ",
               ((union data_type *) adjacency->vertex->data)->f,
               adjacency->weight);
            adjacency = adjacency->next;
        }
        printf("\n");
        vertex = vertex->next;
    }
}
void print_char_ptr(void *graph_ptr) {
  struct graph *g = (struct graph *) graph_ptr;
  struct vertex_list_node *vertex = g->head;
  while (vertex) {
    printf("vertex: %s
", ((union data_type *) vertex->data)->s);
    printf("adjacencies: ");
    struct adj_list_node *adjacency = vertex->adjacencies;
    while (adjacency) {
      printf("(%s, weight: %d) ",
        ((union data_type *) adjacency->vertex->data)->s,
        adjacency->weight);
      adjacency = adjacency->next;
    }
    printf("\n");
    vertex = vertex->next;
  }
}

void print_unweighted_int(void *graph_ptr) {
  struct graph *g = (struct graph *) graph_ptr;
  struct vertex_list_node *vertex = g->head;
  while (vertex) {
    printf("vertex: %d
", ((union data_type *) vertex->data)->i);
    printf("adjacencies: ");
    struct adj_list_node *adjacency = vertex->adjacencies;
    while (adjacency) {
      printf("%d ", ((union data_type *) adjacency->vertex->data)->i);
      adjacency = adjacency->next;
    }
    printf("\n");
    vertex = vertex->next;
  }
}

void print_unweighted_float(void *graph_ptr) {
  struct graph *g = (struct graph *) graph_ptr;
  struct vertex_list_node *vertex = g->head;
  while (vertex) {
    printf("vertex: %f
", ((union data_type *) vertex->data)->f);
    printf("adjacencies: ");
    struct adj_list_node *adjacency = vertex->adjacencies;
    while (adjacency) {
      printf("%f ", ((union data_type *) adjacency->vertex->data)->f);
      adjacency = adjacency->next;
    }
    printf("\n");
  }
void print_unweighted_char_ptr ( void * graph_ptr ) {
    struct graph *g = (struct graph *) graph_ptr ;
    struct vertex_list_node * vertex = g->head ;
    while ( vertex ) {
        printf ( " vertex : %s
", (( union data_type *) vertex -> data ) ->s);
        printf ( " adjacencies : 
" );
        struct adj_list_node * adjacency = vertex -> adjacencies ;
        while ( adjacency ) {
            printf ( "%s ", (( union data_type *) adjacency -> vertex -> data ) ->s);
            adjacency = adjacency -> next ;
        }
        printf ( "
" );
        vertex = vertex -> next ;
    }
    printf ( "
" );
}

char * get_bool_str ( int val ) {
    if ( val ) {
        return "true" ;
    } else {
        return "false" ;
    }
}

void print_bool ( void * graph_ptr ) {
    struct graph *g = (struct graph *) graph_ptr ;
    struct vertex_list_node * vertex = g->head ;
    while ( vertex ) {
        printf ( " vertex : %s
", get_bool_str ((( union data_type *) vertex -> data ) ->i));
        printf ( " adjacencies: 
" );
        struct adj_list_node * adjacency = vertex -> adjacencies ;
        while ( adjacency ) {
            printf ( " (%s, weight : %d) ",
                get_bool_str ((( union data_type *) adjacency -> vertex -> data ) ->i),
                adjacency -> weight );
            adjacency = adjacency -> next ;
        }
        printf ( "
" );
        vertex = vertex -> next ;
    }
    printf ( "
" );
}

void print_unweighted_bool ( void * graph_ptr ) {
    struct graph *g = (struct graph *) graph_ptr ;
    struct vertex_list_node * vertex = g->head ;
    while ( vertex ) {
        printf ( " vertex : %s
", get_bool_str ((( union data_type *) vertex -> data ) ->i));
        printf ( " adjacencies: 
" );
        struct adj_list_node * adjacency = vertex -> adjacencies ;
        while ( adjacency ) {
            printf ( " %s ", get_bool_str ((( union data_type *) adjacency -> vertex -> data ) ->i),
                adjacency -> weight );
            adjacency = adjacency -> next ;
        }
        printf ( "
" );
        vertex = vertex -> next ;
    }
    printf ( "
" );
}

```c
1195    printf("%s ",
1196           get_bool_str(((union data_type *) adjacency->vertex->data)->i));
1197    adjacency = adjacency->next;
1198 }
1199    printf("\n");
1200    vertex = vertex->next;
1201 }
1202    printf("\n");
1203 }

//////////////////////////////////////////////////////////////// TESTNG  //////////////////////////////////////////////////////////////////////

1209    //void print_graph(void * graph_ptr) {
1210    struct graph *g = (struct graph *) graph_ptr ;
1211    struct vertex_list_node *vertex = g->head ;
1212    while (vertex ) {
1213        printf(" vertex: %p\n", vertex);
1214        printf("data: %p\n", vertex->data);
1215        printf("adjacencies: ");
1216        struct adj_list_node *adjacency = vertex->adjacencies ;
1217        while (adjacency) {
1218            printf(" %p", adjacency->vertex);
1219            adjacency = adjacency->next ;
1220        }
1221        printf("\n\n");
1222        vertex = vertex->next ;
1223 }
1224    printf("\n");
1225 }

1228    void print_queue (struct queue_list_node * queue ) {
1229        fprintf(stderr, "printing queue: ");
1230        while (queue && queue->v) {
1231            fprintf(stderr, "%d ", *(int *) queue->v->data);
1232            queue = queue->next ;
1233        }
1234        printf("\n");
1235    }

1236    void print_visited (struct vertex_list_node ** visited ) {
1237        int size = *(int *) visited[0]->data ;
1238        printf("printing visited (excluding dummy size node): [ ");
1239        for (int i = 1; i <= size ; i++) {
1240            if (visited[i]) {
1241                printf("%d ", *(int *) visited[i]->data);
1242            } else {
1243                printf("0x0 ",
1244            }
1245        }
1246        printf("] \n");
1247    }

1250    void print_edges (struct edge_list_node *e) {
1251        while (e) {
```
printf("from: %d to: %d weight: %d\n", *(int *) e->from->data, *(int *) e->to->data, e->weight);
e = e->next;
}

void add_bidirectional_edge(void *a, void *b) {
    add_edge(a, b);
    add_edge(b, a);
}

int main() {
    struct graph *g = (struct graph *) new_graph();
    struct graph *g2 = (struct graph *) new_graph();
    int *new_data = malloc(sizeof(int));
    add_vertex(g, new_data);
    struct vertex_list_node *head = (struct vertex_list_node *) get_head_vertex(g);
    *(int *) head->data = 0;

    int vertices = 6;
    int save_vertex_num = 2;
    struct vertex_list_node *save;
    struct vertex_list_node *savedarray[vertices];
    savedarray[0] = head;

    for (int i = 1; i < vertices; i++) {
        int *new_data = malloc(sizeof(int));
        struct vertex_list_node *vertex = (struct vertex_list_node *) add_vertex(g, new_data);
        *(int *) vertex->data = i;
        if (i == save_vertex_num) {
            save = vertex;
        }
        savedarray[i] = vertex;
    }

    add_bidirectional_edge(savedarray[0], savedarray[1]);
    add_bidirectional_edge(savedarray[1], savedarray[2]);
    add_bidirectional_edge(savedarray[2], savedarray[3]);
    add_bidirectional_edge(savedarray[3], savedarray[4]);
    add_bidirectional_edge(savedarray[0], savedarray[5]);
    add_bidirectional_edge(savedarray[1], savedarray[5]);
    //printf("num vertices: %d\n", num_vertices(g));
    //printf("before entering bfs land *save->data: %d \n", *(int *) save->data);
    struct vertex_list_node **visited = get_bfs_visited_array(g2);
    void *queue = get_dfs_stack(savedarray[0], visited);
    struct vertex_list_node *curr = get_next_dfs_vertex(visited, queue);
    printf("asdfsdf %d\n", *curr->data);
    curr = get_next_dfs_vertex(visited, queue);
}
struct graph *g_nei = (struct graph *) graph_neighbors (g, savedarray[0] -> data);

graph_set_undirected_edge_weight (g, savedarray[3] -> data, savedarray[4] -> data, 50);

graph_set_edge_weight (g, savedarray[4] -> data, savedarray[0] -> data, 80);

graph_set_edge_weight (g, savedarray[1] -> data, savedarray[2] -> data, 70);

graph_set_undirected_edge_weight (g, savedarray[0] -> data, savedarray[3] -> data, 70);

fprintf (stderr, "%d \n", graph_get_edge_weight (g, savedarray[0] -> data, savedarray[1] -> data));


//print_edges (edge_list);

print_data ((void *) g);

//printf ("\n%d\n", num_edges (edge_list));

}*

C library: graph.c

8.6 Testing

#!/bin/sh

# Regression testing script for giraph, adapted from microC
# Step through a list of files
# Compile, run, and check the output of each expected-to-work test
# Compile and check the error of each expected-to-fail test
# Path to the LLVM interpreter
LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="/usr/local/opt/llvm/bin/llc"

# Path to the C compiler
CC="cc"

# Path to the microc compiler. Usually "./microc.native"
# Try ".build/microc.native" if ocamllbuild was unable to create a symbolic link.
GIRAPH="/giraph.native"

# Set time limit for all operations
ulimit -t 30
globallog=testall.log
rm -f $globallog
testall.sh:85:
error=0
globalerror=0
keep=0

Usage () {
  echo "Usage: testall.sh [options] [.mc files]"
  echo "-k Keep intermediate files"
  echo "-h Print this help"
  exit 1
}

SignalError () {
  if [ $error -eq 0 ]; then
    echo "FAILED"
    error=1
  fi
  echo "$1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile. Differences, if any, written to difffile
Compare () {
  generatedfiles="$generatedfiles $3"
  echo diff -b "$1" "$2" "$3" 1>&2
  diff -b "$1" "$2" "$3" 2>&1 || {
    SignalError "$1 differs"
    echo "FAILED $1 differs from $2" 1>&2
  }
}

# Run <args>
# Report the command, run it, and report any errors
Run () {
  echo $* 1>&2
  eval $* || {
    SignalError "$1 failed on $*"
    return 1
  }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail () {
  echo $* 1>&2
  eval $* && {
    SignalError "failed: $* did not report an error"
    return 1
  }
  return 0
}

Check () {
  error=0
  basename='echo $1 | sed 's/.*//g'

  # remaining checks...
}
```bash
s/. gir//'
refile='echo $1 | sed 's/. gir//'
basedir="'echo $1 | sed 's/\([-/\)*//'

echo -n "$basename..."
echo 1>&2
echo "##### Testing $ basename" 1>&2

generatedfiles="
generatedfiles="$generatedfiles ${ basename}.ll ${ basename}.s ${ basename}.exe
${ basename}.out" &&
Run "$GIRAPH" "$1" "${ basename}.ll" &&
Run "$LLC" "${ basename}.ll" "${ basename}.s" &&
Run "$CC" "-o" "${ basename}.exe" "${ basename}.s" "graph.o" &&
Run "/${ basename}.exe" > "${ basename}.out" &&
Compare ${ basename}.out $reffile.out ${ basename}.diff

# Report the status and clean up the generated files
if [ $error -eq 0 ]; then
  if [ $keep -eq 0 ]; then
    rm -f $generatedfiles
  fi
  echo "OK"
echo "##### SUCCESS" 1>&2
else
echo "##### FAILED" 1>&2
globalerror=$error
fi

CheckFail() {
  error=0
  basename='echo $1 | sed 's/.*\///'
  s/. gir//'
  reffile='echo $1 | sed 's/. gir//'
basedir="'echo $1 | sed 's/\([-/\)*//'

  echo -n "$basename..."
echo 1>&2
echo "##### Testing $ basename" 1>&2

generatedfiles="
generatedfiles="$generatedfiles ${ basename}.err ${ basename}.diff" &&
RunFail "$GIRAPH" "<" $1 "2>" "$ {basename}.err ">>" $globallog &&
Compare ${ basename}.err $reffile.err ${ basename}.diff

# Report the status and clean up the generated files
if [ $error -eq 0 ]; then
  if [ $keep -eq 0 ]; then
    rm -f $generatedfiles
  fi
```
137   echo "OK"
138   echo "#### SUCCESS" 1>&2
139 else
140   echo "#### FAILED" 1>&2
141   globalerror=$error
142   fi
143 }
144
145 while getopts kdpsh c; do
146   case $c in
147     k) # Keep intermediate files
148        keep=1
149        ;;
150     h) # Help
151        Usage
152        ;;
153     esac
154   done
155
156 shift 'expr $OPTIND - 1'
157
158 LLIFail() {
159   echo "Could not find the LLVM interpreter "$LLI"."
160   echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
161   exit 1
162 }
163
164 which "$LLI" >> $globallog || LLIFail
165
166 if [ ! -f graph.o ]
167 then
168   echo "Could not find graph.o"
169   echo "Try \"make graph.o\""
170   exit 1
171 fi
172
173 if [ $# -ge 1 ]
174 then
175   files=$@
176 else
177   files="tests/test-*
178   fi
179
180 for file in $files
181 do
182   case $file in
183     *test-*)
184       Check $file 2>> $globallog
185       ;;
186     *fail-*)
187       CheckFail $file 2>> $globallog
188       ;;
189     *)
190       echo "unknown file type $file"
191       globalerror=1
192       ;;
8.7 Example code

```c
/* Author: Seth Benjamin sjb2190 */
void augment(wedigraph<string> flow, wedigraph<string> path)
{
    /* Get bottleneck capacity of path. */
    int min = -1;
    for_edge (e : path)
    {
        if (min > e.weight() || min == -1)
        {
            min = e.weight();
        }
    }
    /* Augment flow. */
    for_edge (e : path)
    {
        if (flow.has_edge(e.from(), e.to()))
        {
            int current_flow = flow.get_edge_weight(e.from(), e.to());
            /* Add bottleneck capacity to current flow. */
            flow.set_edge_weight(e.from(), e.to(), current_flow + min);
        } else {
            int current_flow = flow.get_edge_weight(e.to(), e.from());
            /* Subtract bottleneck capacity from current flow. */
            flow.set_edge_weight(e.to(), e.from(), current_flow - min);
        }
    }
    return;
}
wedigraph<string> make_residual_graph(wedigraph<string> flow, wedigraph<string> network)
{
    wedigraph<string> residual_graph = [];
    for_edge (e : flow)
    {
        int forward = network.get_edge_weight(e.from(), e.to()) - e.weight();
        int backward = e.weight();
        if (forward > 0) {
            residual_graph.add_edge(e.from(), e.to(), forward);
        } if (backward > 0) {
            residual_graph.add_edge(e.to(), e.from(), backward);
        }
    }
    return residual_graph;
}
wedigraph<string> edmonds_karp(wedigraph<string> network, node<string> source, node<string> sink)
{
    /* The argument network contains the capacities as weights on edges.
    Flow is represented with a graph exactly equivalent to network, but
    with the flow on each edge as the weight instead of the capacity.
    First, set up initial flow of 0 on every edge. */
    /* The argument network contains the capacities as weights on edges.
    Flow is represented with a graph exactly equivalent to network, but
    with the flow on each edge as the weight instead of the capacity.
    First, set up initial flow of 0 on every edge. */	
```
wedigraph<
string> flow = [];  
for_edge (e : network) {  
    flow.add_edge(e.from(), e.to(), 0);  
}  
bool has_path = true;  
while (has_path) {  
    wedigraph<
string> residual = make_residual_graph(flow, network);  
    map<node<
string>> parents;  
    !~ Find shortest s-t path with BFS. ~!  
    bfs (n : residual ; source) {  
        for_node (neighbor : residual.neighbors(n)) {  
            if (!parents.contains(neighbor)) {  
                parents.put(neighbor, n);  
            }  
        }  
    }  
    !~ If we didn’t reach the sink, there is no s-t path in residual ~!  
    if (!parents.contains(sink)) {  
        has_path = false;  
    } else {  
        wedigraph<
string> path = [];  
        node<
string> i = sink;  
        while (i != source) {  
            path.add_edge(parents.get(i), i, residual.get_edge_weight(parents.get(i ), i));  
            i = parents.get(i);  
        }  
        augment(flow, path);  
    }  
    return flow;  
}  

int main() {  
    wedigraph<
string> network = [s:"s" -{20} - u:"u" -{10} - t:"t" -{20} - v:"v"  
                        <-{10} - s ;  
                        u -{30} -> v];  
    prints("network");  
    network.print();  
    wedigraph<
string> max_flow = edmonds_karp(network, s, t);  
    prints("max flow");  
    max_flow.print();  
    return 0;  
}

example: edmonds-karp.gir

8.8 Project Log

commit 4c7ab67b21d1ebc13488c8ba5a18f7e701022cfd5
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Wed Dec 20 19:12:22 2017 -0500

add author comments
commit 68bd47968b3480abe0677d0cac2148c7cd4820d2
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Wed Dec 20 17:14:36 2017 -0500

    semant check to prevent void graphs

commit 6293a3c3ca4a49b9db19a488aa63ef0fd89c383
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Wed Dec 20 16:12:42 2017 -0500

    implement bool graphs, print tests

commit 805802da2858066e4e4e028befa3d81650e5efb
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Wed Dec 20 15:46:25 2017 -0500

    add all print methods

commit dfdd8a8e842be964d422067b4c288af5476bd5e
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Wed Dec 20 06:24:58 2017 -0500

    edmonds-karp prints with strings

commit a9235c68a59ee5940a26e9d232c599dcd98d17f9
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Wed Dec 20 06:04:23 2017 -0500

    prepare edmonds karp for use with generics

commit 8371fe9044f4cfffab66e102b2fa9d91cbd3c5cf
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Wed Dec 20 05:59:02 2017 -0500

    fix all tests after adding generics

commit c7508c5931142d6e564182cfe8a5ae60c286b0d8
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Wed Dec 20 02:25:52 2017 -0500

    wow generics work! need to fix all tests and add more semant checks

commit c001a844a795002f31b8050f6c111b75be26ac4c
Merge: ac67c82 977600d
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Tue Dec 19 19:34:08 2017 -0500

    Merge branch 'master' of https://github.com/jessie-liu1/giraph

commit ac67c821603a47ef2fd6a401e294d70a56602108
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Tue Dec 19 19:19:41 2017 -0500

    fix float operation implementation, implement map<float>

commit 977600d9b18de223ae1c087e00c55ac13118456a
semant checks for if map type is void
commit ecea45cad731d3f4d2ce62558ac083ae4b552b47
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Tue Dec 19 18:09:33 2017 -0500

    implement map<bool>
commit 191a01132b3bef3d0cc701575fadbcb8fa7753e49
Merge: 4acaf9e 07aa887
Author: jessieliu1 <jliu997@gmail.com>
Date: Tue Dec 19 17:49:53 2017 -0500

    Merge branch 'master' of https://github.com/jessieliu1/giraph
commit 4acaf9e48c09b6039519052f37e815bef1bdde187
Author: jessieliu1 <jliu997@gmail.com>
Date: Tue Dec 19 17:49:47 2017 -0500

    semant indentation consistency
commit 7c2a5965d4f0697f1be8ac9f98a147a57f9504431
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Tue Dec 19 17:43:43 2017 -0500

    implement map<string>
commit 07aa88714c09dc49204d1b7de98289f315d8cd10
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Tue Dec 19 05:15:28 2017 -0500

    max. flow. fucking. works.
commit 6ef1625b21186a1ba8648c1303509778591f0ce9
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Tue Dec 19 01:35:31 2017 -0500

    add maps for nodes, graphs
commit 35676788f931ef70af8f68715e88a2c590ec7f7b
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Tue Dec 19 00:43:52 2017 -0500

    implement map.contains() in codegen
commit d3a654fa96989c28ea573d2045f43eee3409118f
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Tue Dec 19 00:22:47 2017 -0500

    int maps work w/ put, get
commit 7ac131c582fc0bd8b59be12df228cc5f1d4bc61
Author: jessieliu1 <jliu997@gmail.com>
Date: Mon Dec 18 23:51:37 2017 -0500
add tests for nested funcs in edge methods

commit 310b03360ebc912912cf35c6eba437d79a91e8c6
Merge: a527f26 5fb2048
Author: jessieliu1 <jliu997@gmail.com>
Date: Mon Dec 18 23:49:50 2017 -0500

Merge branch 'master' of https://github.com/jessieliu1/giraph

commit a527f26231553ff4d823fd539abc75c1c6d89526
Author: jessieliu1 <jliu997@gmail.com>
Date: Mon Dec 18 23:49:45 2017 -0500

add env propagation to edge methods

commit 5fb2048967184d7ef0d41a28dba455297e608145
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Mon Dec 18 22:40:02 2017 -0500

fixed issue with skipping nested func calls

commit 4556ff32039d0ca26f25a20e98bd3f74d5efd47
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 21:03:20 2017 -0500

kill off printbig

commit 6975e3454d660e26db2bdc631dcede26cceeabf8d
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 20:53:04 2017 -0500

remove old test file

commit 505afcce25f233cd376894e3a9e33f56109a8269
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 20:50:08 2017 -0500

add binary and, or operators to parser, scanner

commit ede59bdbc33b774c392f5eca4a5d66815f5534cf
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 19:41:54 2017 -0500

actually fix fail-func error msg, oops

commit d49bb7a1cc37affabaa25f77d9bf506df7244
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 19:38:47 2017 -0500

fix fail-func error

commit bd8d5455c3008e50077f1df3f7036f45b20902f5
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 19:35:25 2017 -0500

update test-funcallgraphlit
commit 6f22ec4087819d1b13b43e21a5614f16f3070508
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 19:30:09 2017 -0500

  recursively handle graph lits in any expr, not just assigns

commit 27945f0d9d0a5ceb3fdbb31c20fd1c2d1b5a9519
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 19:27:56 2017 -0500

  add bool_lit to semant convert_expr

commit aec6973408b41728a0e21f45fd74c82d0051c4
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 19:08:17 2017 -0500

  ret_in_loop test returns 0

commit 8827a6bba6fbd2e3c298a75da1eca94b9f4d4b8
Author: jessieliu1 <jliu997@gmail.com>
Date: Mon Dec 18 16:13:45 2017 -0500

  test return in loop

commit 482a673918fc72c74e8dca950d9b0cda99fed0e
Author: jessieliu1 <jliu997@gmail.com>
Date: Mon Dec 18 16:00:44 2017 -0500

  add check for returns in for loops

commit c77253927644c3a5b365529ea41bd7f2a74d395f
Author: jessieliu1 <jliu997@gmail.com>
Date: Mon Dec 18 15:39:21 2017 -0500

  add err files for test cases, tests for calls/returns in loops

commit bc620ce51f6d0e06e38ccdfb3d2a24b303fabe
Merge: 5a1db22 a0210f7
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Mon Dec 18 13:00:32 2017 -0500

  Merge branch 'master' of https://github.com/jessieliu1/giraph

commit a0210f780fb23469cf9beb8d66a366071135d60
Author: Daniel Benett <deb2174@columbia.edu>
Date: Mon Dec 18 12:47:54 2017 -0500

  updated undirected for edge for self loops

commit 5a1db22ec3719eeb45cfe5b6271a36b911b5be5f
Merge: 5408256 4e34883
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Mon Dec 18 11:24:37 2017 -0500

  Merge branch 'master' of https://github.com/jessieliu1/giraph
commit 540825646b52c3a1bd87a83797eef39ae0612fac
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Mon Dec 18 11:24:28 2017 -0500

err file for for_node fail test

commit a31c62a1a2815b4d8e7e3e3587b352e483d19f14
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Mon Dec 18 11:22:33 2017 -0500

added rest of concurrent modification checks+tests

commit 7fb51c20d8ba3db71d7742ad38e7398e430776c1
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Mon Dec 18 10:48:20 2017 -0500

fixed semant match error

commit 4b86f698937df832cf0dddb033eb765afe18b1119
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Mon Dec 18 10:42:47 2017 -0500

added for_node concurrent modification check

commit 4e34883e54343dcabc425cf1a1430ce09a77a655
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 07:58:58 2017 -0500

add remove edge tests

commit a11c007064296e35a5b7d92a4869a7eed025a530
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 07:52:17 2017 -0500

no reinitializing nodes or reweighting edges in graph literals

commit 62eb741682ca465180bca1ca6b2df08dc1e921d6
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 07:17:26 2017 -0500

print returns int

commit 622bf89c9e5133b37ec5eaad7833bdc502c67995
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 07:12:07 2017 -0500

edgeless graph literals can be any graph type

commit 1a7a16d12b3892f8d211c267bda55bc3cdc5dc5
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 06:47:26 2017 -0500

edge.set_weight() acts appropriately given wegraph or wedigraph

commit 86d5abdb5e8c4093ba7c95d1e28b6251b8217581
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 06:12:46 2017 -0500
test dfs in 3rd self-loop test

commit ab09a76f7bc8884db939fba4b3779239cfd9ff1f
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 06:10:19 2017 -0500

add tests for self-loops - two break

commit 71ad5a39f25b2f4e1338b24abb0ae2777e1f6419
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 05:47:27 2017 -0500

add has_edge, has_node and tests; fix semant bug to allow nodes to occur in multiple graph lits

commit 14a9f2ad7324e81be86e42fa283a766df65068bb
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 04:43:34 2017 -0500

fix neighbors() segfaulting when given vertex not in graph

commit 8a5ae5b5f1cc2a6f04e303f9c1b542ea73d9dda
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 04:32:54 2017 -0500

add_wedge_method checks if edge is present before adding

commit 1bb318c6960a4233f8baf51d96982cddf25206ce
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 04:26:21 2017 -0500

wedigraph test for get/set_edge_weight

commit c9d48ac8eb7d98282de4146db575e6c9499
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 04:25:18 2017 -0500

get/setweight in codegen, tests

commit 42832aa7b36475bc2317a4facda0424b20d92b2d
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 02:38:12 2017 -0500

add_edge method in codegen for weighted graphs; fix semant problems of non-initialized nodes and checking graph methods

commit 330e633da16d66375cd888a0a390b21e20f724b8
Merge: 3b66205 cd7e6b7
Author: Jennifer Bi <jbi3495@columbia.edu>
Date: Mon Dec 18 01:56:41 2017 -0500

Merge branch 'master' of https://github.com/jessieliu1/giraph

commit 3b6620557ed4b3cb88a481ab17159db947b925a5
Author: Jennifer Bi <jbi3495@columbia.edu>
Date: Mon Dec 18 01:56:31 2017 -0500
fixed last env nesting problems, + test

commit cd7ebf73a12aed19ed2b4e18271f7452ca8413ed
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 01:51:54 2017 -0500

fix edge/graph method checking, add tests - all pushed tests pass at this point

commit c7097ed6a639f7febb40348c584c82578657e048
Author: Daniel Benett <deb2174@columbia.edu>
Date: Mon Dec 18 01:16:20 2017 -0500

memset after malloc

commit 8d5cb9d7fba0c457a4ce0f7206c3ab441156f9a0
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 01:13:43 2017 -0500

fix return type of check_edgemtd

commit a539b578043a7dad5c962d138fbaca4ad0f1753b
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Mon Dec 18 01:10:39 2017 -0500

add wedge, diwedge types; semantic checking ensures weight() only called on those (with tests)

commit e65bc83f8c87d21b5b7109c20074d055b54f7006
Author: Daniel Benett <deb2174@columbia.edu>
Date: Mon Dec 18 01:02:33 2017 -0500

dfs with tests

commit 28c941c57e2949edba046d2bb1ec682f6fe9a038
Merge: ca0c4de 6b9e643
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 17 23:57:25 2017 -0500

Merge branch 'master' of https://github.com/jessieliu1/giraph

commit ca0c4de131329b608f090eeac31a59577fa2fcf4
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 17 23:57:06 2017 -0500

added return check and return tests

commit 6b9e6430431747f0162819448d44cd17cbd59374
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sun Dec 17 23:50:19 2017 -0500

check neighbors, get/set_edge_weight in semant
changed env propagation in checks

commits:
  95af9bac43774f55ca3cd4ae8ed290e2642fe40
  Sun Dec 17 23:21:48 2017 -0500
  Seth Benjamin <sethjbenjamin@gmail.com>

fix for loops

for_edge works correctly for undirected graphs

finally add gitignore

diff edge set_weight for undirected/directed

all the functions\! in the sast\!

organize graph.c
commit b6a77128276a182a280a223d72b2827cf1aa50ec
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sun Dec 17 19:19:25 2017 -0500
delete map.c (moved into graph.c)

commit 171e137a780488a0af230facedb0692ae04662c9
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sun Dec 17 19:18:37 2017 -0500
add graph neighbors method

commit 85efe80ae855d564e644f014edcc087f242e4dcf
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sun Dec 17 18:24:21 2017 -0500
fix graph method return types in semant

commit 576f1ada1b6a235fe48b3b4980cc5e003b037f0a
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sun Dec 17 18:04:55 2017 -0500
printing sast

commit 22380987b0316f8a515020684888e6381ab5cfd5
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sun Dec 17 17:49:22 2017 -0500
put map into graph.c, create undirected for_edge, handle map put key
collisions

commit e4e93af874e548cc996bd3f4fd721f56a7fd7d32
Merge: 19d1aaa 541d7f4
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 17 17:18:46 2017 -0500
Merge branch 'master' of https://github.com/jessieliu1/giraph
commit 19d1aaa76d8019834ab8057bd717ca937e125a7d
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 17 17:18:43 2017 -0500
change id matching to expr type check, for multiple method calls

commit 541d7f4c15a7b87a384ecb657651aa0391c597ee
Merge: 5608b02 eba5994
Author: jessieliu1 <jliu997@gmail.com>
Date: Sun Dec 17 16:14:50 2017 -0500
resolve failures with list.hd in foredge tests

commit 5608b02d83b3399314bc0a2eaa491427570d3d03c
Author: jessieliu1 <jliu997@gmail.com>
Date: Sun Dec 17 16:04:02 2017 -0500
allow two method calls like .from().data()
commit eba5994076b2950a3a2e94a8c8a1b3e96581d33b
Merge: 0bf0725 b464054
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 17 15:18:42 2017 -0500

Merge branch 'master' of https://github.com/jessieliu1/giraph

commit 0bf072543c3f1badf347730a24ed9173e3753f46287
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 17 15:18:31 2017 -0500

correct return types in SMethod

commit b46405422a3a15972e511c6399509511e9021af
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sun Dec 17 11:55:42 2017 -0500

add map contains_key method

commit 9c082d4d5e4e7a8fbb0168d88b2a0a38c76ce
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 17 11:51:52 2017 -0500

added graph, edge method checks

commit 93f25186dd5ebd37e5e0b4f76c8831a532b8f03
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sun Dec 17 08:37:30 2017 -0500

add add_node semantic check

commit c1817a9d83b27f041a11ace7185b00ce98fd7a69d
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sun Dec 17 08:07:20 2017 -0500

wegraphs, wedigraphs work end-to-end

commit e50812499032d260f8b769625f28f7fee7d9f54f
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sun Dec 17 06:10:16 2017 -0500

update edmonds karp

commit 776e82dcafdff86e41aa67b7a910a956f258e0cd8
Author: jessieliu1 <jliu997@gmail.com>
Date: Sun Dec 17 03:39:06 2017 -0500

init nodes in graph literal in semant

commit 534c14f5a459225a64423380f8583c87ab946c59
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sun Dec 17 03:25:56 2017 -0500

preparing graphs for generics

commit 35b0725fe8684751c8a778df4c6378a516d8ef71
reserve map location 0

commit b26e8080f681925904206e0cd79c9d44d8ec74b3
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sun Dec 17 03:00:16 2017 -0500

simple map

commit 16fa0d8a3018febca6040f124430d308b4ef1031
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sun Dec 17 02:29:14 2017 -0500

merge graph.c

commit fc0497bcb0c96192ed1e3c22f4e233ad6440
Merge: efcfae7 c0adb84
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sun Dec 17 02:19:43 2017 -0500

merge

commit efcfae7b64637ea4d4b2a2f82a32eb15586c7eba
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sun Dec 17 02:16:42 2017 -0500

edge weights graph.c and codegen

commit c0adb842ab3f292d5a4148bef870d75240125581
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sun Dec 17 00:40:46 2017 -0500

only allocate new node if not immediately initialized to another

commit 00494c9872e9f6f72bcf246f9724ef71627e888
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sun Dec 17 00:32:21 2017 -0500

remove unnecessary edge functions

commit c48fb7dfa52e95dbf6bb57cf11276c04763757ac
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sat Dec 16 23:28:49 2017 -0500

working for_edge iteration

commit 1661ff7adc92c6299e81b3c5f912b4494fe6a
Merge: 1b55381 9ae827b
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 16 23:07:56 2017 -0500

resolve merge conflicts

commit 1b553813d2ea2d7d26b2eb046201069dc2248f4f
Author: Seth Benjamin <sethjbenjamin@gmail.com>
digraphs work end to end

commit 8bd2620dc2922c6bf00b19b8d6211bfff1e40093
Author: Seth Benjamin <sethbenjamin@gmail.com>
Date: Sat Dec 16 22:03:06 2017 -0500

digraphs in parser

commit 9ae827bf0a3f6eb026cf2fe84b2123849811880e
Merge: ec03c15 bc87447
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Dec 16 20:35:53 2017 -0500

Merge branch 'master' of https://github.com/jessieliu1/giraph

commit ec03c15e0abed6e0e0c0a1c06baadb2f7d9fa2c
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Dec 16 20:35:45 2017 -0500

fix error with reinit

commit bc874473a2538da70e1e075cc3ffde06c08aa29
Author: Seth Benjamin <sethbenjamin@gmail.com>
Date: Sat Dec 16 19:58:42 2017 -0500

make testall check failing tests, add fail-vassign test

commit 8c640e1c4a71455e2c675b464fc61007243f11c2
Merge: 0df4e9c 3f0e888
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Dec 16 19:17:27 2017 -0500

pulled?

commit 0df4e9cd9081991de187eb9dde243c5abe144da6
Merge: 744e5a7 0537793
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Dec 16 19:15:31 2017 -0500

merge on my end

commit 3f0e88813c3a0c20db5a833fad0106052945847
Merge: 0537793 a318a3a
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sat Dec 16 19:05:52 2017 -0500

merge for real!'

commit 0537793119d423e915cbae1cee1421efee665a56
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sat Dec 16 16:57:27 2017 -0500

added some comments

commit a318a3a041d8bc5091951bdc394818c36e627a
fix to get sexpr for graphs

commit 1db15254635270b5879e338e71cbde09d057478f
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sat Dec 16 16:36:38 2017 -0500

working bfs with tests

commit b176f1f6855b06197f5da939cad54a6cc551a96b
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sat Dec 16 14:41:33 2017 -0500

parsing for undirected graphs

commit 389fd97b665a827963aebfb43ce8b2728b77a47
Merge: 4ba534a 9d03399
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sat Dec 16 10:58:36 2017 -0500

added parser helper fns and directed graphs

commit 4ba534a2ba18252ba1422d36142b95c23958ce6
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sat Dec 16 10:51:33 2017 -0500

Merge branch 'working' of https://github.com/jessieliu1/giraph into working

commit 1b18aaffc70d3c729cb208790230125f99407d65
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 16 06:15:24 2017 -0500

add graph.remove_edge() method

commit ff9b985627c1a8172cb4bf123938f6c69ccf54f
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 16 05:44:33 2017 -0500

add graph.add_edge() method

commit 976afbc1f6ec5cc41f24df88f66ea55dd85abd114
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 16 05:11:53 2017 -0500

add graph.remove_node() method with tests

commit 5dbac7e0faba2ce044500ddab82306f2ab863a60
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 16 05:11:23 2017 -0500
test for add_node when adding duplicate node

commit 1c917fbee6f322b6c827241b48870abf9bdf3f8
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 16 03:53:27 2017 -0500

add_node only adds if node is not already present

commit a10792e6922439b758ce178f53278e73e1ce79a7
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 16 02:33:05 2017 -0500

graph.add_node() works

commit 53c92d2490ea7ef6ba72b7cc6dd5c61ba83b96e3
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 16 02:12:44 2017 -0500

methods are called on expressions not strings

commit f4d18e9ff08f3988d88b66e61688b5ec20911f3
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 16 01:23:20 2017 -0500

all data is malloc'd c-side so it doesn't die when node names go out of scope

commit 9d03399c6d5273b04583c8be858be80216653181
Author: jessieliu <jliu997@gmail.com>
Date: Sat Dec 16 00:29:17 2017 -0500

added: can't do reinitialization of vars

commit 3d624667dce30c2414486cf1f41b8aad3c97c83f
Author: jessieliu <jliu997@gmail.com>
Date: Sat Dec 16 00:07:30 2017 -0500

add lt gt neq leq to parser

commit 4e5087aa0e4c86ec2170f88c12ea2f949775080a
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Fri Dec 15 23:58:33 2017 -0500

merge master into working

commit 8d5944b180aca52e51738671eb863409555e3e60
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Fri Dec 15 18:13:50 2017 -0500

for_node, node data tests

commit 0e97eb520b4dcf93cb01fde23940514cf6bbc07f
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Fri Dec 15 17:33:34 2017 -0500

for_node works
Merge branch 'master' of https://github.com/jessieliu1/giraph

new methods in graph.c

update testall to use graph.o, add test for scoped blocks

all bracketed blocks get their own scope in codegen

add neq token

remove check for printbig.o in testall.sh

Merge branch 'working' of https://github.com/jessieliu1/giraph into working

not quite good with return statements

testing setting, accessing node data
node methods data(), set_data() work

commit 6f9bc5bb926805d399acb429f3afcd80f10562e5
Merge: 3f011b8 4d569e6
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Thu Dec 14 23:17:10 2017 -0500

Merge branch 'master' of https://github.com/jessieliu1/giraph
commit 3f011b85310b4c8bc3899033fcff0df99225a1
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Thu Dec 14 23:13:58 2017 -0500

Revert "merge branches, changed Graph to Graph_Lit in sast"
This reverts commit c7aa0534f6c2de3a9ec1f31dfb987425eefe18e9, reversing changes made to 1b9a18b43ca6b9e0590ce10e4151da52d2c9e6d.

commit 5edbb7005409aed80733f8c52668a439b56c5cca
Merge: 639c2d3 33e6a13
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Thu Dec 14 23:07:08 2017 -0500

Merge branch 'master' into working
commit 33e6a134e83a112406843420a39cdeb2d1de0c8c
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Thu Dec 14 22:39:30 2017 -0500

fixed graph_lit sexpr/expr problem
commit 4d569e6930a5e1fa0de0a169fa8bb9aa28860fb0
Author: Daniel Benett <deb2174@columbia.edu>
Date: Thu Dec 14 22:24:36 2017 -0500

merge fixing (remove NodeTyp/EdgeTyp)
commit 639c2d3bd154ad890b65bcdc0d370b27cb25befc
Author: jessieliu1 <jliu997@gmail.com>
Date: Thu Dec 14 21:28:46 2017 -0500

fixed block failure, weird vassign failure though
commit c7aa0534f6c2de3a9ec1f31dfb987425eefe18e9
Merge: 1b9a18b 4e38f6a
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Thu Dec 14 20:01:22 2017 -0500

merge branches, changed Graph to Graph_Lit in sast
commit 4e38f6a78e0739e7970b9019dd46552920e21772
Author: jessieliu1 <jliu997@gmail.com>
Date: Thu Dec 14 16:10:25 2017 -0500
remove newsemant

commit 800580fc92eb0df17b5ffaf8726c9685c16c0c19
Author: jessieliu1 <jliu997@gmail.com>
Date: Thu Dec 14 16:07:55 2017 -0500
everything makes

commit cf65f92b3fe7db75ba049bb435c1a3583df07e64
Author: jessieliu1 <jliu997@gmail.com>
Date: Thu Dec 14 15:47:27 2017 -0500
this semant compiles

commit 1b9a1b4b43ca6b9e0590ce10e4151df052d2c9e6d
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Thu Dec 14 03:53:48 2017 -0500
when parsing graph literals, only add edge if not already present

commit 74ae017bfe016826c1a7615876360d2f5607
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Thu Dec 14 03:38:46 2017 -0500
supports multiple components in a graph literal, separated by semicolons

commit ea7cf2bc220660aa96daa49de6f9fcd2d6c0bd
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Thu Dec 14 02:07:14 2017 -0500
remove 'function' from scanner/parser

commit 9559d79ef96ababaa7b677c3f50a07c4d03a43f9
Author: jessieliu1 <jliu997@gmail.com>
Date: Thu Dec 14 01:15:41 2017 -0500
all the statement conversions

commit 4b2424d9624a122c99677b1c9fe1d649030da612
Author: jessieliu1 <jliu997@gmail.com>
Date: Thu Dec 14 01:13:38 2017 -0500
add block checking

commit 6ca82e2e2771c6823af85619bd52496f64f634
Author: jessieliu1 <jliu997@gmail.com>
Date: Thu Dec 14 01:08:44 2017 -0500
this semant compiles

commit af61a0c274867860bd9249128a166f731b6492f
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Thu Dec 14 00:51:32 2017 -0500
remove node and edge expr from ast, change graph expr to graph_lit

commit f0b65c2cda3d8ab43000604336f40b568971dc1
resolved formals type error with fdecls

commit 98906658506a332647450c478a4c7359417af0b06
Author: jessieliu1 <jliu997@gmail.com>
Date: Wed Dec 13 14:00:51 2017 -0500

need to fix weird sfdecl formal thing

commit ac42ad96b3a3fe2942384de822bd69e9107544
Merge: dee4742 b98e603
Author: jessieliu1 <jliu997@gmail.com>
Date: Wed Dec 13 13:29:43 2017 -0500

debugging node type errors

commit dee4742a9c94a2b859bd7668da7cac1df0d
Author: jessieliu1 <jliu997@gmail.com>
Date: Wed Dec 13 13:11:54 2017 -0500

add exception

commit 0b84dc38b54c74161724d0b7636a7dbec74b4b1
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sun Dec 10 06:59:36 2017 -0500

initializing node data in graph literals works in codegen

commit b98e6033171b6e020868e71b90bf0beb2d74f792
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 10 02:26:13 2017 -0500

added raise Failure functions

commit 5705c9f2b1551fe48bc2c7ad2decbbbee641fbd64
Merge: 21db32b f7415f6
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 10 02:11:49 2017 -0500

more sast print fixes

commit 21db32b422607f90740c09f81d89f53737927cc
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 10 02:10:12 2017 -0500

more sast print fixes

commit f7415f6f0852b827e212ed1f1156710d630620a5
Author: jessieliu1 <jliu997@gmail.com>
Date: Sun Dec 10 01:00:31 2017 -0500

sast print fixes

commit 9d80a930db2bdbe0ffbb733073243012f72f4f0
Merge: 83552bf e582101
added convert_ast function; resolve merge conflicts

commit 83552 bf5f5b760652b823854f7b9a86d75c835af
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Dec 10 00:36:52 2017 -0500

added convert_ast function; updated pretty-print

commit e582101e97ca5ff26bbe20ef69efdad8b077b55
Author: jessieliu1 <jliu997@gmail.com>
Date: Sun Dec 10 00:20:52 2017 -0500

finishing statement checks

commit e57a3d6d65b4ac0cc98a0bb39a746fe93ab8229c
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 9 22:13:51 2017 -0500

scanning and parsing graphs with data

commit e2709e577de39698b128a82e7af21ee7c69f2
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 9 21:44:45 2017 -0500

graph literals now surrounded by square brackets; single-node graphs work

commit e43fab55708620367439d1a4f58ed892dc7689f2
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 9 19:09:48 2017 -0500

small cleanup of graph codegen, fix node iterators in ast and parser

commit fe0aa9bb3f83613bb5de7c80015f8b152c297347
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 9 18:49:45 2017 -0500

graphs with cycles now work (still need to do parentheses and comma syntax)

commit e7a3323ceef1cbe7ad5bb845ceb43af30bebfb9
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Dec 9 03:36:31 2017 -0500

round out the stats

commit 18fe414983a610417c06bc3ae45af396fc3005b4
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 9 02:24:18 2017 -0500

process and allocate nodes of graph literals as local variables

commit 82c8e6a9f914b9457e576f1f4e2bebda8b7922a4
Author: jessieliu1 <jliu997@gmail.com>
Date: Thu Dec 7 03:12:07 2017 -0500
not quite compiled, but building semant.ml

commit a54bfac30c3a98fb0029a874d7e628e8c98f7d9b
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 2 07:49:53 2017 -0500

graph codegen works for simple undirected graphs without data

commit ccd38f628a15f1e03624da4f88a767b9426af5d9
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 2 05:07:13 2017 -0500

begin C api for graph codegen

commit 4fb044aabb0d2103145a9df6da53f8886d
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 2 05:05:09 2017 -0500

check command line args of toexe.sh

commit 5557a6f6cb5d8e37bac7e2d641c475758060ac65
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Dec 2 03:35:57 2017 -0500

fix whitespace and pretty-printing of vdecl in ast

commit 321e3753d340f47c912fb6fe45f4847142d95b71
Author: jessieliu1 <jliu997@gmail.com>
Date: Thu Nov 30 01:48:25 2017 -0500

add a lil more to env

commit e8316aabb0d2103145a9df6da53f8886d
Author: jessieliu1 <jliu997@gmail.com>
Date: Wed Nov 29 03:06:13 2017 -0500

add env into semant, changes to sast and formatting for codegen/parser

commit 404cababc18221189b9a4c20cb9689a47bf1f2f5
Author: jessieliu1 <jliu997@gmail.com>
Date: Tue Nov 28 16:28:57 2017 -0500

more formatting

commit f785077d5a09e14e1e8d552f3bb1db6243748108
Author: jessieliu1 <jliu997@gmail.com>
Date: Tue Nov 28 16:26:42 2017 -0500

some formatting

commit 13e8ba3354f45710580ed81ea601b06381cfaa7a
Author: jenniferbi <jb3495@columbia.edu>
Date: Mon Nov 27 20:23:14 2017 -0500

skeleton for sast
Merge branch 'working' of https://github.com/jessieliu1/giraph into working

commit a359d0a17c0e6e2a0f003b2c27fdd6df895a8155
Author: jessieliu1 <jliu997@gmail.com>
Date: Mon Nov 27 15:26:56 2017 -0500

add failing tests with vassign

commit 18e70591df0b39c037a271f6bfe0f36d7a3595813
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Nov 26 14:27:05 2017 -0500

delare prints fn to fix unrecognized fn exception

commit 744e5a766c58cc52031cedd011f436f24c5a1ff0
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Nov 25 19:29:14 2017 -0500

some formatting

commit 2917af2264c967f6d4fb2d15ab5458f82decfd7b
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sat Nov 25 19:25:33 2017 -0500

minor changes, syntax

commit 3bef1a8d9f7681cd3e740e76255612e8a48f31ee
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sat Nov 25 19:20:05 2017 -0500

semantic checking adapted from microc

commit 3dc67082a2ea9eb896d4fbbcaab7a3e31d342437
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Nov 25 11:42:18 2017 -0500

actually resolve all merge conflicts

commit b8c1e04fb38520d3138207cbcc37fb51cee2a84b
Merge: ec08127 d281b53
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Nov 25 11:00:29 2017 -0500

resolve merge conflicts in parse and codegen

commit d281b537a229ea2c17033db2ea83ac02b7ab911f
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sat Nov 25 01:16:56 2017 -0500

one line var declaration and assignment

commit a74ebc760f6673e4b38d88d1a92bb3d758fd0b6e
Author: Seth Benjamin <sethjbenjamin@gmail.com>
correctly parsing and printing simple one-line undirected graphs

commit ec08127698f4d0ff3067072574f70d7fb1cbfaa9
Merge: 7873670 0350777
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Nov 18 01:14:36 2017 -0500

merging conflicts in parser

commit 03507779f0f3b9459bbdbcb33061de92add33500
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Sat Nov 18 01:10:52 2017 -0500

remove vdecls from fdecl; instead, local vars declared and parsed as statements in fnct body

commit 795d7f42d355e03d98dbcd917f3a2d50600d7179
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date: Fri Nov 17 21:57:43 2017 -0500

begin parsing graphs

commit eefc0afbd222210ff6c030789c528eb90e4c6795
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sat Nov 11 15:53:28 2017 -0500

rm toexe.sh in tests dir

commit 7435bfed97c06b3a4fe16a9876143e9d6f51f
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sat Nov 11 15:47:00 2017 -0500

remove a comment

commit 7873670f5d11b0aeb91267b17c52f27e14bd69c5
Merge: 4748fb1 a4cd358
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Nov 11 15:45:22 2017 -0500

Merge branch 'master' of https://github.com/jessieliu1/giraph

commit 4748fb1f83493e38c661a322df3430367192038f
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Nov 11 15:45:19 2017 -0500

some format cleanup

commit a4cd358c35e60907dabe3ad72a0cb0f817bcbe255
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sat Nov 11 15:21:58 2017 -0500

toexe.sh script, run with './toexe.sh tests/test-name.gir'

commit 124a0188b42c97fee788aeb6d4c67ac1cae8e4d3
Author: Jennifer Bi <jb3495@columbia.edu>
added regression test script and test files

commit 2b5bae2f026cb379821bd3ca37d0f1925a33f91c
Author: Daniel Benett <deb2174@columbia.edu>
Date: Mon Oct 30 22:45:07 2017 -0400

    add printbig.c

commit efb5712c99bf42f7fd91c9dbdec7773b45453df1
Author: Daniel Benett <deb2174@columbia.edu>
Date: Mon Oct 30 22:40:41 2017 -0400

    Hello, World

commit a85064b6a33c5858000b976eaf650ed48e40e85b
Author: Seth Benjamin <sethjbénjamin@gmail.com>
Date: Mon Oct 30 04:20:45 2017 -0400

    add int, string, float, bool literals

commit b80a3045920a8934a3dcc5e0528da197897597
Author: Seth Benjamin <sethjbénjamin@gmail.com>
Date: Mon Oct 30 04:16:54 2017 -0400

    add hello_world test

commit bfb472e363fcf14d9e6288594601d08fade396cc
Author: Seth Benjamin <sethjbénjamin@gmail.com>
Date: Mon Oct 30 04:16:54 2017 -0400

    add print0 test source

commit 2295ddf6981d562d3fee83f12b22ed375dd5d
Author: Seth Benjamin <sethjbénjamin@gmail.com>
Date: Mon Oct 30 02:44:23 2017 -0400

    fix order of token rule in scanner - codegen for return.gir works

commit dc047a76ffcbf7339d31200c5f9ad7714b9c816d
Author: Seth Benjamin <sethjbénjamin@gmail.com>
Date: Mon Oct 30 02:36:26 2017 -0400

    remove char type

commit ce5ea36d181bd50522450d479ef14ede6783fc7
Author: Seth Benjamin <sethjbénjamin@gmail.com>
Date: Mon Oct 30 02:09:26 2017 -0400

    fix empty token error by parsing IDs in scanner

commit a7be6a8e8c347d8209b520b2cab381d932fc44b0
Merge: a88fc3b 24bb10b
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Oct 29 12:07:04 2017 -0400
updated giraph.ml skeleton

commit 24bb10be158e8f20ab5339be38b6918e9d7e87e4
Author: Jennifer Bi <jb3495@columbia.edu>
Date: Sun Oct 29 10:43:51 2017 -0400

updated codegen

commit 71f65d4a54f0ff32b90657eeab8a729ca7f807c5
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sat Oct 28 18:09:53 2017 -0400

skeleton for end-to-end codegen

commit 42c1040a8167db0686965649a33beb68ebc9bd2d
Merge: 3786ce7 e434213
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Oct 28 15:42:04 2017 -0400

Merge branch 'master' of https://github.com/jessieliu1/giraph

commit 3786ce75707996b1899a847b463c5aa0d61b8a2b
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Oct 28 15:41:46 2017 -0400

remove Int_Lit from ast

commit e43421375d5cfe968d6e48387fe8d240927c540a
Author: Daniel Benett <deb2174@columbia.edu>
Date: Sat Oct 21 20:41:36 2017 -0400

remove functions and fn_list

commit 597a87b8a72e2b7f26843ca97bf34a441c5bdcf0
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Oct 21 20:31:16 2017 -0400

resolve vdecl error in parser

commit 55c5a4367da69c45dbd8c3c850bc79ad18b7037
Author: jessieliu1 <jliu997@gmail.com>
Date: Sat Oct 21 20:11:00 2017 -0400

change the parser program rule

commit 2772e1989e7830da24dd7732c085ae8f545bc0b8
Author: Seth Benjamin <sethbenjamin@gmail.com>
Date: Sun Oct 15 23:53:15 2017 -0400

add break, continue
commit c58993f60d55e6ea7e1366cb19ac6aa2ff6c35a4
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date:  Sun Oct 15 22:00:12 2017 -0400
update edmonds_karp with for_edge, for_node

commit f1127d245b3a96227a9a5974fd53c87118b3dd0e
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date:  Sun Oct 15 21:24:11 2017 -0400
add function calls

commit bb51009cb5bdabf82c6a48560deee935cc347314
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date:  Sat Oct 14 22:27:47 2017 -0400
add for_node, for_edge, bfs, dfs

commit c31c8b7db1d9404ea2153b36f31e3df9e729749a
Author: jessieliu1 <jliu997@gmail.com>
Date:  Sat Oct 14 14:06:32 2017 -0400

Added node and edge to scanner/ast

commit d4fbff1bc5ceeb8a880db8406794a1089d92325
Author: Jennifer Bi <jenniferbi@dyn-160-39-151-135.dyn.columbia.edu>
Date:  Fri Oct 13 12:53:42 2017 -0400
fixed shift/reduce conflict, added missing token defs in parser

commit 8c713b642d03d9ab48e2c1c8c5aff330802ec9d6
Author: Jennifer Bi <jenniferbi@dyn-160-39-183-163.dyn.columbia.edu>
Date:  Fri Oct 13 10:28:36 2017 -0400

added Makefile

commit 70b6d83d38a1a55a602d7cc50ccd2eef73d79dd9
Author: jenniferbi <jb3495@columbia.edu>
Date:  Thu Oct 12 11:47:37 2017 -0400

basic parser and ast

commit b59ffe954d15d7f5be7f7512fbbc2a629840c1dd
Author: Seth Benjamin <sethjbenjamin@gmail.com>
Date:  Wed Oct 11 17:45:12 2017 -0400
add hello world, edmonds-karp example programs

commit df4d4fcef83f56581eb959fd395b712ce03e06d7
Author: jessieliu1 <jliu997@gmail.com>
Date:  Tue Oct 10 15:06:27 2017 -0400

Formatting for scanner

commit b1065bf5916569181e148ef52ebed32df0c0d622
fix digits and letters in scanner

commit 5e788d66042f8ef84b9f790db0fee2038d51d598
Author: Daniel Benett <deb2174@columbia.edu>
Date: Mon Oct 9 22:16:30 2017 -0400

forward slash

commit f9ccf5d0ffe9dfd8d515e8f680af8bed6ac16347
Author: Daniel Benett <deb2174@columbia.edu>
Date: Mon Oct 9 22:10:34 2017 -0400

updated scanner

commit 079f9df3305ee59bec57d3e0f542e2446c5dbe
Author: jenniferbi <jb3495@columbia.edu>
Date: Mon Oct 9 22:09:34 2017 -0400

Create scanner.mll

commit 700f3e1651b846ed7a35b3c3567c0c734fdddb4f75
Author: jenniferbi <jb3495@columbia.edu>
Date: Mon Oct 9 21:19:51 2017 -0400

Delete scanner

commit 625f1c173db6739e548808c9f95c98e2b3d6f65e
Author: jenniferbi <jb3495@columbia.edu>
Date: Mon Oct 9 21:19:29 2017 -0400

Scanner skeleton

commit 98c73f215ab831035f847af8710eb3670e39fd41
Author: Jessie Liu <jliu997@gmail.com>
Date: Mon Oct 9 21:17:55 2017 -0400

Create README.md

git repository can be found at: https://github.com/jessieliu1/giraph