The GAL Programming Language:

A rapid prototyping language for graph algorithms

Athar Abdul-Quader
ama2115@columbia.edu

Shepard Saltzman
sms2195@columbia.edu

Albert Winters
ajw2124@columbia.edu

Oren B. Yeshua
oby1@columbia.edu

Contents

Chapter 1

Introduction

A graph G consists of a set of vertices V and a set of edges E each of which joins two of the vertices. This simple abstraction serves as a model for a multitude of real world systems and leads to a wide variety of useful and elegant algorithms for solving problems in those systems. However, despite the elegance of the formalism, implementing graph algorithms in a general purpose programming language can become quite cumbersome. Programmers must first make decisions about how to represent graphs, edges, and vertices, and then attempt to translate the algorithm into the desired language.

1.1 Audience

Students and researchers studying algorithmic theory should find GAL exceptionally useful in allowing them to quickly implement and experiment with the techniques they are investigating. GAL allows students to focus on algorithm concepts without worrying about time-consuming implementation details - helping to foster understanding of the algorithm as a whole, and perhaps facilitating the discovery of improvements where possible. GAL is also well suited for developers looking to quickly include graphs and graph algorithms in their software while avoiding the hassle of choosing an appropriate API and learning to use it.

1.2 Related work

While there are many graph and network algorithm APIs, like jGABL[*] and boost[†], they are all based on general purpose programming languages. As such, algorithms implemented using those APIs are encumbered with the syntax and nuances of the language, making them difficult to read and maintain. We have been unable to find a programming language developed specifically for computing with graphs.

[*]http://www.math.tu-berlin.de/jGABL/

[†]http://www.boost.org/libs/graph/doc/

1.3 Goals

1.3.1 Intuitive

The GAL language is terse and uncluttered by a myriad of non-essential symbols. GAL syntax closely mirrors popular pseudocode formats found in the algorithm literature making GAL code both easy to develop and intuitive to understand.

1.3.2 Concise

GAL programs should be concise in comparison to their counterparts in other languages. Because GAL is designed specifically for graph algorithms, it can provide significant LOC savings over the standard general purpose programming languages (both imperative and functional) when working with graphs. Built in data structures, operators, and functions for working with graphs and sets facilitate this goal.

1.3.3 Portable

The Java code produced by our GAL compiler can be quickly integrated into any Java application, providing the flexibility, scalability, and cross platform support that comes from using the Java language. Furthermore, GAL uses a simple set of primitives and built in functions that can be implemented in any general purpose programming language. While we will provide an implementation in Java, GAL compilers can be written for your preffered target language based on the GAL specification. Learning various APIs for different languages is time consuming and unnecesary.

1.3.4 Efficient

GAL will use appropriate data structures and algorithms in its internal representation and manipulation of graphs, sets, and queues in order to free the programmer from dealing with such issues. The focus of GAL is on rapid prototyping, so the core concern is to keep computationally efficient algorithms running efficiently when implemented in GAL. For fine tuning of constant factors, the code may be tightened in the target language.

1.4 Features

1.4.1 Data types

GAL includes graphs, sets, and queues as built-in types along with the familiar numbers, constants, strings, and booleans. The language is weakly typed for added flexibility and readability.

1.4.2 Control structures

Simple and familiar program flow control mechanisms like `while` and `for` loops and `if/else` statements are provided. Some language specific control structures are the `foreach` keyword as well as indentation to specify scope.

1.4.3 Comments

The traditional '//' signifies a single line comment, enabling a natural and readable embedding of annotation alongside the code.

1.4.4 A simple example

To get a sense of what a basic GAL program would look like, what follows is a depth-first traversal algorithm as it might be written in GAL.

```
DFS(G)
   foreach (u in G.V)
      u.visited <- FALSE          // initialize the vertices

   foreach (u in G.V)             // for all
      if (u.visited = FALSE)      // unconnected components of G
         DF-VISIT(u)              // begin traversal

DFS-VISIT(u)
   print(u)                       // output vertex info to the terminal
   u.visited <- TRUE              // mark node as visited
   foreach (e in u.edges)
      if (e.head.visited = FALSE)
         DFS-VISIT(e.head)        // recurse

MAIN()
   graph G                        // declare a graph
   G.V = {1,2,3,4,5}              // add vertices
   G.E = {(1,2),(1,3),(2,3)       // add edges
   G.E += {(4,5)}                 // add more edges
   DFS(G)                         // traverse
```

The simple program above demonstrates many of our GAL's features. Graphs, vertices, and edges are all basic types, but additional fields can be added to them on the fly as is the case with the `visited` field used to tag vertices in the example. Recursion is supported as in the call to `DFS-VISIT` and comments nicely annotate the code. The basic types are also printable for debugging or rapid prototyping purposes.

1.5 Summary

Programmers seeking to utilize graph algorithms in their software encounter a paradigm shift in which they must take the algorithms from the theoretical perspective in which they were discovered, and convert them into a functioning computer program. It is the aim of GAL to make this transition as simple and natural as possible. Algorithms implemented in GAL are human-readable, which allows this language to be used for teaching purposes as well as general graph algorithmic development. We hope our GAL will become a useful tool for studying, prototyping and implementing graph algorithms.

Chapter 2

Tutorial

This tutorial will walk you through a few simple programs to get you started on programming with GAL.

2.1 ”Hello, World!”

First, we’ll take a look at GAL’s Hello World program to get you started.

2.1.1 The Code

```
#
# World's First GAL Program
#
MAIN()
	println("Hello, World!")
```

2.1.2 Line-By-Line Walkthrough

```
#
# World's First GAL Program
#
```

These first three lines are comments, as indicated by the # marks in front of each one. Whenever GAL encounters a # mark, it ignores the rest of that line. Comments may start anywhere in a line, but there is no way to comment out an entire block except by commenting each line out individually.

```
MAIN()
```

Next, we declare the MAIN() function. Like in C++ and Java, the Main function is where the program begins. Notice that GAL does not ask you to declare a return type. Instead, you simply give it the name of a function and a list of arguments. Also notice the lack of braces – GAL uses indentation exclusively to determine scope. Each new block is exactly one tab deeper (no whitespace!) than its containing block.

```
        println("Hello, World!")
```

Now we use println, which converts its argument into a string and returns it to the console. It is possible to put any type of GAL variable into a print statement – GAL will automatically convert them to their string representation.

2.1.3 Making it Run

Now we just need to run the program. Save your file as HelloWorld.gal. Now type the following line to compile your program:

galc HelloWorld.gal

This will create a file called HelloWorld.jar. Now type:

java -jar HelloWorld.jar

If all goes well, this will run your HelloWorld program and print, ”Hello, World!” the terminal! Congratulations!

2.2 ”Hello, World!” Revisited

Now that you've got Hello World up and running, let's try a slightly more involved version that introduces GAL's primitive types and makes scoping a little clearer.

2.2.1 The Code

```
#
# World's Second GAL Program
#
MAIN()
        bool goodMood <- true
        string greetings += "Hello,"
        time <- 10
        if (goodMood and (time > 7))  #is the world awake yet?
                print(greetings + " World")
        println("!")
```

2.2.2 Line-By-Line Walkthrough

```
#
# World's Second Gal Program
#
```

Once again, we start a few comments to introduce our program.

```
MAIN()
```

Again, we declare our Main function. Function names, like variable names, can be any letter followed by any number of letters, numbers and underscores. For clarity, this tutorial uses all capital letters for its function names.

```
        bool goodMood <- true
        num time <- 10
        string greetings += "Hello,"
```

Now we'll declare one variable for each of GAL's three primitive types.

bool declares a boolean value, which holds either "true" or "false". $<-$ is the assignment operator, initializing goodMood's value to "true". Variables do not have to be initialized on declaration, although it is a good practice to do so to prevent unexpected behavior in case a variable is called before it is initialized.

num declares a floating-point number. GAL numbers support basic arithmetic operations, including mod.

string declares a string. In this case, I use the $+=$ operator to concatenate "Hello," to the end of the variable "greetings". Since strings automatically intialize to the empty string, in this case it is equivalent to simply assigning the variable the value of "Hello,".

```
        if (goodMood and (time > 7))  #is the world awake yet?
                print(greetings + " World")
        println("!")
```

if, as with most programming languages, takes a condition followed by a block. If the condition is true, the block is executed. In this case, "goodMood" has a value of "true", and "time" has a value of 10, making both sides of the *and* statement true. Therefore the condition evaluates to "true *and* true", is true.

print takes a string (including Java escape characters) and returns it to the console. In this case, "greetings" evaluates to "Hello," which is concatenated (via the + operator) with " World" to produce "Hello, World". Note that this line is indented to indicate it is in the *if*-block.

println is identical to *print*, but adds a newline to the end of the string. Note that it is at the same indentation as the *if* statement, so it will be executed regardless of whether the *if*'s conditional is true or not.

2.3 Depth-First Search

Next, we'll go over the code for a DFS traversal of a graph in GAL. Notice how similar GAL code looks to pseudocode!

2.3.1 The Code

```
DFS(G)
        foreach (u in G.V)
                u.visited <- false #initialize the vertices

        foreach (u in G.V)             # for all
                if (u.visited = false)  #unconnected components
                        DFS_VISIT(G, u) #begin traversal
DFS_VISIT(G, u)
        println(u) # output vertex info to the terminal
        u.visited <- true # mark node as visited
        foreach (v in adj(u))
                if (v.visited = false)
                        DFS_VISIT(G, v)        # recurse

MAIN()
        graph G                        # declare a graph
        G.V += {1,2,3,4,5,6,7,8}       # add vertices
        G.E += {(1,2),(1,4),(1,5)}     # add edges
        G.E += {(2,3),(2,4),(2,5)}
        G.E += {(3,4),(3,5),(3,6),(3,7),(3,8)}
        G.E += {(4,5)}
        DFS(G)                         # traverse

        show(G)
```

2.3.2 Line-By-Line Walkthrough

We'll start with the MAIN() function at the bottom, then look at DFS(G) and DFS_VISIT(G, u).

MAIN()

```
MAIN()
        graph G                        # declare a graph
```

As usual, we start by declaring a function. *graph* declares a new graph variable, one of GAL's composite types. Graphs are a collection of vertices and edges between those vertices, held in a graph's *V* and *E* fields respectively. So now we have a graph G, with G.V being an empty set of vertices and G.E being an empty set of edges.

```
        G.V += {1,2,3,4,5,6,7,8}        # add vertices
        G.E += {(1,2),(1,4),(1,5)}      # add edges
        G.E += {(2,3),(2,4),(2,5)}
        G.E += {(3,4),(3,5),(3,6),(3,7),(3,8)}
        G.E += {(4,5)}
```

Here we are populating the graph with vertices and edges. The first line, "G.V += 1,2,3,4,5,6,7,8", adds eight verticles, numbered 1 through 8, to G's vertex set. The next four lines add a total of twelve edges to the graph. While this could be done in one line, breaking it up like this makes the code neat.

V and E are unlike most sets in GAL – for starters, they only accept objects of a particular type, whereas other sets can take any mixture of objects. A graph's V set takes only *num* objects, while an E set takes only tuples of the form (*num*1, *num*2), with *num*1 being the head of the edge and *num*2 being the tail. While V and E can interact with other sets in all the usual ways (+ for union, − for the regular difference, and so on), they DO NOT use the < − operator. Whenever you want to add elements to a graph's V or E field, use + = instead, with V and E automatically being initialized to the empty set.

```
        DFS(G)                         # traverse

        show(G)
```

DFS(G) calls the DFS function declared above, using G as its argument. It is necessary to use G as an argument, as GAL is statically scoped.

show(G) is a special function that creates a visualization of the graph given as its argument. The visualization allows for options such as rearranging the nodes and coloring them via a vertex's "color" field. It is also possible to declare a "step()" function in your code, which will add a "step" button to the visualization to allow you to step through the execution of an algorithm. For more information, please see the GAL reference manual.

DFS(G)

```
DFS(G)
        foreach (u in G.V)
                u.visited <- false #initialize the verticies
```

The first line declares a new function, DFS(G). There is no problem with re-using "G" as a variable name, as this is separate scope from where G was declared before. Note that it is not necessary to declare a function's return type, nor the type(s) of its argument(s).

The next line introduces *foreach*, a keyword which establishes a loop that iterates over a collection of elements. "u" is simply a variable used to temporarily hold the value of whatever item the loop is currently up to. *in* is simply a keyword designed to make *foreach* more intuitive to use. G.V is our graph's vertex set. Altogether, this line establishes a loop that will go over all eight vertices from G.V and execute a block of code for each one. In other words, this will do something "for each vertex, u, in G.V".

Finally, we have the body of the loop. This line introduces the dot operator, which can be used on vertices and edges to assign them fields. These untyped fields are created as-needed. In this case, there is no need to declare a "visited" field earlier in the code – simply by assigning the value of "false" to it will create it if it doesn't already exist. As the comment indicates, this loop is sufficient for ensuring every vector in G has a "visited" field with value "false".

```
        foreach (u in G.V)             # for all
                if (u.visited = false)  #unconnected components
                        DFS_VISIT(G, u) #begin traversal
```

We're back to the indentation of the original *foreach*, meaning we're now outside its block. Since the temporary variable u was only valid within that scope, we're free to reuse it for this loop. Once again, we'll iterate over each vertex in G.V.

Next, we'll use an *if* statement to ensure that we only visit nodes that haven't yet been visited – that is, those whose "visited" fields are still "false". Note that unlike C++ and Java, we use "=" as the equality operator, since we don't use it as our assignment operator.

Finally, if the vertex we're looking for hasn't been visited yet, we'll visit it now by calling the DFS_VISIT function defined below it.

DFS_VISIT(G, u)

```
DFS_VISIT(G, u)
        println(u) # output vertex info to the terminal
        u.visited <- true # mark node as visited
```

As before, we declare a function and give names to its arguments. Again, since G and u are declared in separate scopes, there is no problem with reusing their names.

println(u) starts the function by printing the number of the node to console, followed by a newline. There is no problem putting a vertex, or any GAL object, directly into a print statement – print automatically turns any GAL object into a string using a predictable method. In this case, printing a vertex prints the number of that vertex. If you'd like to give your vertices more informative names, you could give each one a "name" field and print "u.name" instead.

At the same time that u is printed to the console, we set its "visited" field to "true" to prevent multiple visits.

```
        foreach (v in adj(u))
                if (v.visited = false)
                        DFS_VISIT(G, v) # recurse
```

Once again, we establish a loop with *foreach*. "adj(u)" is a built-in function that returns a set of all verticies that are adjacent to u. So in this case, we're iterative over every vertex "v" that has an edge leading to or from "u".

As before, we use an *if* statement to ensure that no vertex is visited multiple times.

Finally, we end with a recursive call to DFS_VISIT(G, v). This makes sure that we first visit v's neighbors before we return to this loop to continuing visiting u's neighbors.

2.4 Other GAL Features

Now that you know what a basic GAL program looks like, let's look at a few other features of the GAL language. This snippit of code will introduce you to wildcards, the length operator and range operator.

2.4.1 The Code

```
MAIN()
        graph G

        G.V += {1 .. 5}
        G.E += {(1,2),(2,3),(4,5),(1,5)}

        G.V[3].color <- "red"
        G[(2,3)].weight <- 0.66

        println("degree of 2: "+|G[2]|)
        println("out edges of 2: "+G.E[(2,?)])
        println("in  edges of 2: "+G.E[(?,2)])
        println("all  edges of 2: "+G.E[2])
        println("out adj to 2: "+G.V[(2,?)])
        println("in  adj to 2: "+G.V[(?,2)])
        println("all adj to 2: "+adj(G[2]))
```

2.4.2 Line-By-Line Walkthrough

```
MAIN()
        graph G

        G.V += {1 .. 5}
        G.E += {(1,2),(2,3),(4,5),(1,5)}
```

We start by declaring a Main function. To create G's vertices, we use the range operator, .., to quickly include all numbers between 1 and 5. When using the range operator, be sure to include a space between the operator and the first number – while whitespace is usually ignored in GAL, in this case it is necessary. This is followed by the addition of four edges to G.

```
        G[(1,2)].weight <- 1
        G[(2,3)].weight <- 0.66
        G[(4,5)].weight <- 2
        G[(1,5)].weight <- 0.50
```

Next, we'll use the dot operator to access *weight*, a special field of edges. Using *weight* as opposed to user-defined field has several advantages. First, it's possible to use the length operator, |[expression]|, to access an edge's weight. Also, when using the *show*(*G*) function, weight will be displayed on the edges.

```
        println("degree of 2: "+|G[2]|)
        println("weight (2,3): "+|G[(2,3)]|)
```

Here we see two more uses of the length operator. As indicated by the text, the first line returns the degree of 2, while the second line returns the weight of edge (2,3). The length operator can also fetch the number of elements in a set or queue, the length of a vector or the absolute value of a number.

```
        println("out edges of 2: "+G.E[(2,?)])
        println("in  edges of 2: "+G.E[(?,2)])
        println("all  edges of 2: "+G.E[2])
```

Now we'll print several lines of information using the wildcard symbol, ?. The first line prints all edges in G of the form (2, ?), i.e. all the edges that lead away from G's second vertex. (If you simply wanted to know how many edges lead away from 2, you could throw the length operator around G.E[(2,?)].) Similarly, the second line prints every edge that leads to 2, while the final line prints all edges connected to 2 in G, regardless of direction.

```
        println("out adj to 2: "+G.V[(2,?)])
        println("in  adj to 2: "+G.V[(?,2)])
        println("all adj to 2: "+adj(G[2]))
```

As you can see, the wildcard operator also works on a graph's vertex set. As might be expected, the first line prints all the vertices in G that have edges that come from 2, while the second prints all the vertices in G that have edges leading to 2. Like above, the final line prints all vertices connected to 2, regardless of the direction of the edges.

Thanks to these operators, the only difference between a directed and undirected graph in GAL is how you use it. If you want to work with an undirected graph, you can use adj(u) and G.E[u] to get all the vertices and edges connected to a vertex u, respectively. If you care about direction, you could instead use wildcards to ensure that you only travel edges of the proper direction.

Chapter 3

Reference Manual

3.1 Lexical Conventions

A GAL program consists of an ASCII text file with the .gal extension.

3.1.1 Comments

GAL recognizes single-line comments that begin with '#' and terminate at the end of the line. Multi-line comments can be achieved by commenting each line individually.

3.1.2 Whitespace

Comments and the ASCII space are ignored by the GAL compiler. However, characters normally ignored as whitespace play a significant role in the GAL language. The horizontal tab is not considered whitespace because it is used to indicate scope, and the line terminator is used to indicate the end of a statement.

In GAL, scope is indicated by the tab ('\ t') character. Every line in a GAL program must begin with zero or more tabs. Before a GAL program is compiled, it undergoes a scope preprocessing step. As the lines of the program are read by the preprocessor from first to last, a running count is kept of the *indentation level.* The indentation level begins at 0. If the number of tabs at the start of the next line is greater than the current indentation level, a new block is created and the indentation level is incremented by one. If the number of tabs at the start of the next line is less than the current indentation level, the current block is closed and the indentation level is decrimented by one. For a program to successfully compile, the indentation level must be zero when EOF is reached.

3.1.3 Tokens

There are five classes of tokens: identifiers, keywords, constants (immediates), operators, and separators.

Identifiers

Identifiers begin with a letter followed by any sequence of letters, digits and the underscore character. Two letters are considered the same if their ASCII characters are equal. Two identifiers are considered the same if every character in both identifiers match.

Keywords

The following identifiers are reserved as keywords and cannot be used as anything else:

```
set    vertex  edge   graph
queue  string  bool   num
in     foreach while  and
if     else    vector or
true   false
```

Constants

Constants (immediates) provide the GAL programmer with a convenient way to initialize each of the built in types.

Boolean constants `true` and `false` are the boolean constants representing their respective truth values.

Numeric constants A numeric constant consists of an integer part followed by an optional fractional part and optional exponent. The integer part is a string of one or more digits $(0 \ldots 9)$. The integer part may be followed by the fractional part, or the exponent part or both. The fractional part is a period ('.') which is followed by one or more digits. The exponent part is an 'e' followed by an optional sign ('$\pm$') followed by one or more digits.

String constants A string constant is a sequence of characters surrounded by double quotes. The double-quote marks are not considered part of the string and are omitted when processing the string. Double-quote marks may be added to a string with \" and similarly, the backslash can be escaped with itself like so \\.

Edge constants An edge constant reresents a directed edge between two vertices in a graph. For example, `(2,4)` represents and edge from vertex 2 to vertex 4. Edge constants should not be confused with `edge` objects. Edge constants are not associated with a particular graph but are used to modify or index a graph's vertex or edge sets. The values inside an edge constant cannot be arbitrary expressions. They can be number constants, variable names (that refer to vertex objects), or the wildcard '?' character.

Set constants A set constant is a convenient way to create a set object. The syntax is

$$\{expr_1, expr_2, \ldots, \}$$

and results in the creation of a new `set` with the members that result from evaluating each of the contained expressions.

Vector constants [1] A vector constant allows you to create a vector just as you would a set.

$$[expr_1, expr_2, \ldots,]$$

Operators

Arithmetic/Set Operators There are five arithmetic operators: +, -, *, /, and %. The '+' operator denotes addition while the '-' operator denotes subraction. The '*' operator denotes multiplication, and the '/' operator denotes division. The '%' operator denotes the remainder of the division of the first operand by the second operand. The operands must have arithmetic type. The result of using an arithemtic operator on two `num` types is the usual arithmetic operation.

The arithmetic operators perform double duty as set operators as well. The '+' operator indicates the set union operation while the '*' indicates intersection. The '-' operator indicates set difference while the '/' operator denotes symmetric difference. Similarly, '+=', '*=', '-=','/=' are defined in the usual way for sets as well. Furthermore, the `in` operator can be used to determine set membership.

Boolean Operators There boolean operators are as follows: and, or, >, <, <=, >= =, ! . The 'and' operator returns `true` when both the left and right operands evaluate to `true`, `false` otherwise. The 'or' operator returns `true` when either or both the left or right operand evaluate to `true`, `false` otherwise. The '=' operator returns `true` if left operand equals the right operand; otherwise, it returns `false`. The '!' operator is unary and returns `true` if its operand evaluates to `false`, `false` otherwise.

As in most programming languages, the `and` and `or` operators use a shortcut evaluation model. If the first operand of an `or` is true, the second is not evaluated. Similarly, if the first operand of an `and` is false, the second is not evaluated.

Assignment Operators The six assignment operators are as follows: $\leftarrow$, +=, -=, *=, /=, and %=. The '$\leftarrow$ ' operator assigns the value of the right operand to the left operand. '+=' and '-=' adds or subtracts the value of the right operand to the value of the left operand. '*=' and '/=' multiplies or divides the value of the right operand by the value of the left operand. '%=' places in the left operand the remainder of the left operand divided by the right operand.

[1]Vector constants are not yet implemented in the current realease of the GAL compiler.

Access Operator The '.' operator is used to access fields in the left operand. If a field that does not currently exist is accesed, it is created on the fly and inherits its type from whatever is assigned to it.

Index Operator The '[]' operator is used to access an element by index within a `vector`, `graph` or a graph's vertex and edge sets.

Length Operator The length operator works by surrounding an expression with the pipe character as follows:

`|$expr$|`

Length is defined on most types giving the most natural representation of length (size of a queue, cardinality of a set, length of a string, absolute value of a num, etc.).

Range Operator The '..' operator specifies a range of values and provides a convenient shorthand for specifying sets and vectors. Given [num1]..[num2], where [num1] and [num2] are integers, GAL will interpret it as a comma separated list of all the integers between [num1] and [num2] inclusive. Placed in a set or vector constant, it adds the integers in the range as members to the set or vector.

Precedence The operator precedence rules follow those of the C language. Since this may result in unexpected order of operations when operating on sets, it is reccomended that parentheses be used to explicitly specify precedence in expressions involving sets (although it is by no means required). The precedence order is as follows (going from least tightly binding to most tightly binding): `in`, assignment operators, `or`, `and`, `!`, comparison operators, plus or minus, * or / or %, the length operator, access operator, and finally bracketed expressions (index operator).

3.1.4 Separators

The following characters are used as separators: ',' '.' '[' ']'.

3.1.5 Scope

The scope of an identifier begins after the declaration of the identifier and terminates at the end of the block in which the identifier is found. Blocks are determined by indentation, as opposed to C-style '{' and '}' characters.

3.2 Types

3.2.1 Nums, Bools, & Strings

There is only one numeric type in GAL. The `num` is internally represented as a double value. The `bool` type can take on either of the two boolean values `true` and `false`. A `string` is sequence of ASCII characters.

3.2.2 Sets

A `set` is an undordered collection of objects of any type. Sets support all basic group manipulation functions, such as indexing, concatenation and traversal. Sets support the range operator.

3.2.3 Vertices

A `vertex` is a very simple type most often used as an element of a graph. By default, a vertex contains just one field - `index` of type `num` (additional fields can be added using the '.' operator).

3.2.4 Edges

An `edge` is also used to represent an element of a graph. By default, an edge is an ordered pair of two indices, representing a directed edge from the vertex of one index to the vertex of the other. An edge contains three fields by default, each of type `num`. They are: `src`, `dest`, `weight`.

3.2.5 Graphs

Graphs are intended to concisely represent mathematical graphs. A `graph` simply consists of two sets V and E representing the vertices and edges of the graph respectively. These sets can be accessed with the '.' operator.

3.2.6 Vectors

A `vector` is similar to an array but has no prespecified size.

3.2.7 Queues

A `queue` is implemented as a priority queue. It maintains a list of (element, priority) pairs. Queues support the push, pop, enqueue, and change_priority operations. Push is given an element and a priority (of type `num`) to be added to the queue. Pop returns the element in the queue with the greatest priority. Change_priority changes the priority of an element already in the queue. If push is used without specifying a priority, the queue automatically assigns the element a priority higher than anything currently in the queue. This allows the queue to be used as a (LIFO) stack. Similarly, enqueue adds an element to the queue with a priority lower than any elment in the queue, allowing the queue to be used in FIFO fashion.

3.3 Control Structures

foreach

Foreach is a keyword used to generate loops. With "in", it can be used to run a block of code repeatedly, once on each element in a `set`, `path`, or `queue`. The order of iteration for a set is undefined, while that of path is in order from left to right, and that of queue is in order of priority from greatest to least. For example, a basic block of code to find the sum of weights on edges in a graph could be written as:

```
total = 0
foreach (edge in G.E)
   edge.weight += total
```

while

While is a keyword used to generate loops. While is a header with a condition that encloses a block of code. When that block is first reached and after each execution of that block, while's condition is checked. If the condition is true, the block is (re)executed, otherwise it is skipped.

if

If is a keyword used to create conditionals. If statements always include a condition and a block of code, and are optionally followed by an else block. When the if-block is reached, its condition is checked. If that condition is true, the block of code is executed, otherwise it is skipped.

else

Else is a keyword used to create conditionals. An else is a header to a block of code that immediately follows an if-block. If the if-block is executed, the else-block is skipped. Otherwise, the else-block is executed. An else may also be followed directly by an if.

3.4 Syntax

3.4.1 Grammar

$$\begin{aligned}
Program &\rightarrow Var\_decl \mid Func\_decl \\
Var\_decl &\rightarrow Type\ (<id> \mid Assignment) \\
Func\_decl &\rightarrow Func\_def\ '\{'\ Block\ '\}' \\
Func\_def &\rightarrow <id>\ '('\ Parm\_list\ ')'\ <stmt\_term> \\
Parm\_list &\rightarrow <id>\ Id\_list \mid \epsilon \\
Id\_list &\rightarrow ','\ <id>\ Id\_list \mid \epsilon \\
Block &\rightarrow Statement\ Block \mid \epsilon \\
Statement &\rightarrow (\ Expression \mid Func\_call\_stmt \mid Return\_stmt \\
&\mid If\_stmt \mid For\_stmt \mid Foreach\_stmt \\
&\mid While\_stmt \mid Var\_decl \mid '\{'\ Block\ '\}'\ )\ <stmt\_term> \\
If\_stmt &\rightarrow '\text{if}'\ '('\ Expression\ ')'\ '\{'\ Block\ '\}'\ (\ Else\_stmt \mid \epsilon\ ) \\
Else\_stmt &\rightarrow '\text{else}'\ (\ Block \mid If\_stmt\ ) \\
Foreach\_stmt &\rightarrow '\text{foreach}'\ '('\ <id>\ '\text{in}'\ LValue\ ')'\ '\{'\ Block\ '\}' \\
Func\_call\_stmt &\rightarrow <id>\ '('\ (\ Expression\ Exprn\_list \mid \epsilon\ )\ ')' \\
Return\_stmt &\rightarrow \text{return}\ Expression \\
While\_stmt &\rightarrow '\text{while}'\ '('\ Expression\ ')'\ '\{'\ Block\ '\}' \\
Expression &\rightarrow LValue \mid Assignment \mid '!'\ Expression \mid\ Expression\ Operator\ Expression \\
&\mid Immediate \mid '('\ Expression\ ')' \\
Exprn\_list &\rightarrow ','\ Expression\ Exprn\_list \mid \epsilon \\
LValue &\rightarrow LValue\ '.'\ LValue \mid\ <id>\ '['Expression']' \mid\ <id> \\
Assignment &\rightarrow LValue\ AssignOp\ Expression \\
Operator &\rightarrow '+' \mid '-' \mid '*' \mid '/' \mid '\%' \\
&\mid '>' \mid '<' \mid '>=' \mid '<=' \\
&\mid '=' \mid '!=' \\
AssignOp &\rightarrow '<-' \mid '+=' \mid '-=' \mid '*=' \mid '/=' \mid '\%=' \\
Immediate &\rightarrow Set\_const \mid Path\_const \mid Edge\_const \\
&\mid <numeric\ constant> \mid\ <string\ constant> \mid\ <boolean\ constant> \\
Type &\rightarrow \text{graph} \mid \text{set} \mid \text{vertex} \mid \text{edge} \mid \text{queue} \\
&\mid \text{path} \mid \text{num} \mid \text{string} \mid \text{bool} \\
Edge\_const &\rightarrow\ '('\ <numeric\ constant>','<numeric\ constant>\ ')' \\
Set\_const &\rightarrow '\ \{'\ (\ <numeric\ constant>\ Num\_list \mid Range\_const \parallel \epsilon\ )'\}' \\
Path\_const &\rightarrow '\ ('\ (\ <numeric\ constant>\ Num\_list \mid \epsilon\ )')' \\
Num\_list &\rightarrow ','\ <numeric\ constant>\ Num\_list \mid \epsilon \\
Range\_const &\rightarrow <numeric\ constant>\ '..'\ <numeric\ constant>
\end{aligned}$$

Chapter 4

Project Plan

4.1 Process

At the team's weekly meeting, we discussed what the next task on the agenda would be, and then assigned roles to each member of the group. The goal was to get through an entire compilation as quickly as possible, then flesh out the language features, and finally test and debug.

Google's suite of online tools was very helpful in coordination. The GAL Team's Google Calendar was used to schedule meetings, post deadlines, and get an idea of where we were in the project timeline and how much time was left. The team's Todo list was maintained in Google Documents and was jointly editable. At any given point in time, members could add or check off items from the todo list and everyone else would get the update in realtime on any web browser.

4.2 Style Guide

Our guidelines were simple. All blocks have their braces on their own lines. Variable and function names are lowercase and begin with 'm' if they are class members and 'o' if they are static class members. Also, the methods names for operators are lowercase versions of the associated operator's token name in the parser.

In GAL, function names should be uppercased - this helps to distinguish them from the standard library functions.

4.3 Timeline

9-26-06	Whitepaper completed
10-12-06	CVS running
10-15-06	First grammar draft
10-19-06	LRM completed
10-26-06	Eclipse IDE with CVS support and Ant running
11-09-06	Parser completed
11-16-06	AST completed
11-23-06	JGraphT integrated
11-30-06	Testing suite developed
12-14-06	Code Generation completed
12-17-06	All tests paased and demo prepared

4.4 Roles & Responsibilities

Athar Abdul-Quader	Language maven	grammar, parser, static semantic analysis & code gen
Shepard Saltzman		preprocessor, testing, language usage
Albert J Winters	Test suite developer	system arch & code gen
Oren B Yeshua	Team Lead	system arch, lang design, AST & code gen

4.5 Development Environment

GAL was developed on Linux in the CLIC laboratory at Columbia University. The majority of the source code was written in Java and compiled using version 1.5 of Sun's JDK. The Java code for the lexer and parser were generated by version 2.7 of ANTLR. Apache's Ant was used for the build system. Source code management was accomplished using CVS. Several editors (including GNU Emacs, Vim, and Eclipse) were used to develop the source code as each member of the team was more comfortable using his editor of choice.

4.6 Project Log

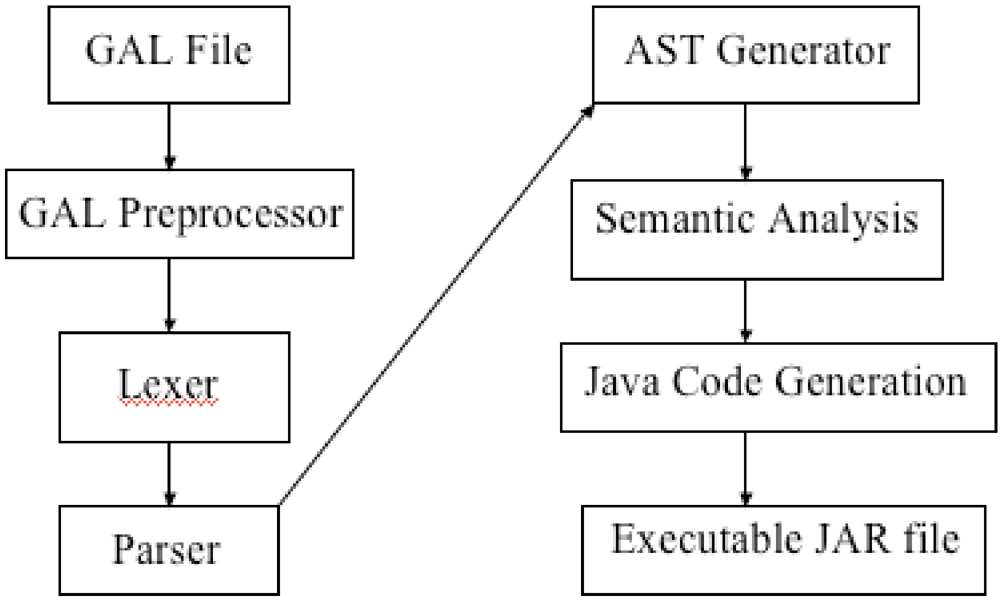
see `http://docs.google.com/View?docid=ddrk75sv_20dfm7rj`

Chapter 5

Architectural Design

The gal compiler, `galc`, is a JAVA program which runs the preprocesser, lexer, parser, semantic analyzer and code generator, and an ant script which, on top of that, takes the generated Java code and creates an executable JAR file.

Figure 5.1: The GAL Compiler



5.0.1 GAL Preprocessor

The GAL preprocessor is a python script that processes the indentation of a file, and adds appropriate braces when changing scope. If a line is begun with spaces instead of tabs, this script outputs an error (with the line number) to stderr. This component was implemented by Shepard. The `galc` compiler checks for the existence of a flag indicating that the input GAL program has braces. If this flag does not exist, then the compiler will first run this script on the input file. If the script outputted any errors, `galc` will output these to stdout and then stop compilation before it gets to the next stage (the lexer/parser). This integration into the `galc` compiler was handled by Jay.

5.0.2 Lexer, Parser, AST

The lexer and parser are ANTLR programs defining the tokens and grammar of the language. The lexer rules were implemented by Oren, and the parser rules by Athar. We then created a heterogeneous tree, that uses our node classes that extend the base ANTLR tree nodes. The benefit of using this is that it simplified code generation and semantic analysis. This was done directly in the parser, and was implemented by Oren. The `galc` compiler uses the output of the last stage (preprocessing) as input to the lexer, and then uses that as input to the parser. Then, before running any rules on the parser, it calls `parser.setASTNodeClass()` to tell ANTLR to use our classes, that extend the base `antlr.CommonAST` class, as the default tree nodes. It then runs the parser by calling the first rule in the grammar, `parser.program()`.

5.0.3 Semantic Analysis

Once the parser is run, we get the root of the generated AST by calling `parser.getAST()`, and casting it to the type that handles program nodes, `edu.columbia.plt.gal.ast.Prog`. We then use a SymbolTable (`edu.columbia.plt.gal.SymbolTable`) to store identifiers in their appropriate scopes. SymbolTable is our own data structure consisting of a Hashtable (which hashes from a String identifier to a generic type T), and a link to its parent's SymbolTable (`null` if it does not have a parent). Its methods include returning a entry in its table or any of its parents tables based off a String identifier, and checking to see if an identifier exists within the current scope. This SymbolTable object is given to the Prog node's `setEnv()` method, which sets its current SymbolTable to the object that was passed in, and then calls `setEnv` on all its children. This method is implemented in the base `edu.columbia.plt.gal.ast.Node` class, which every Node in our AST inherts. This is only overwritten in certain instances: when a variable is declared, when a new scope is entered, or when a variable is used. When a variable is declared, setEnv checks to see if it already exists in the current scope, and if it does not, then it adds it to the current SymbolTable. If it does, then an error is outputted to stderr giving the line number. When a new scope is entered, setEnv() creates a new SymbolTable, giving it a link to the current SymbolTable as its parent. This has two special cases: function declarations and foreach loops. Both of these have identifiers which should be added to its child's scope (ie, the scope of the block), so a new SymbolTable is created, populated with those identifiers, and given to the child block as its SymbolTable, rather than as its parent. Finally, when a variable is used, setEnv() checks to see if the variable has been defined in the current or any surrounding parent scopes, outputting an error to stderr if necessary.

In addition, a function SymbolTable is kept as a static member of the Prog node class. There is only one scope in which functions can be defined, so storing these is fairly simple. One caveat is that this function SymbolTable also needs to be able to examine the methods in our standard library, `edu.columbia.plt.gal.GalStdLib`. This is done using Java reflection, allowing us to examine the methods in the GalStdLib class. When a function is called, then, we check to see if it exists in this function SymbolTable (with the appropriate number of arguments), and if it does not, we output an error to stderr. The static semantic analysis portion was implemented by Athar.

We should note here that type checking is not covered in our semantic analyzer. GAL is weakly typed, so any errors in comparing or operating on different types will simply be populated into the generated Java code and caught at runtime.

5.0.4 Code generation and creation of an executable JAR

After semantic anlaysis, `galc` creates a Java file, adds the appropriate headers and class definition, and then calls the `gen()` method on the root of the AST. The root of our AST is a Prog node, whose children are variable declarations and function declarations. This node first generates all global variable declarations, and initializes them in a static block in the class, then generates all function declarations.

The back-end of our code generation is what makes the code generation itself very simple. Types are handled by an interface, `edu.columbia.plt.gal.IGalRef`, which is implemented by `GalVal` and `GalRef`. GalVal is an implementation of IGalRef which uses "pass-by-value" semantics, whereas GalRef uses "pass-by-reference". Each GalVal and GalRef object contain a `GalType` object, which is an abstract class which defines a number of operators between GalTypes. GalType is sub-typed by GalNum, GalString, GalBool, GalVertex, GalEdge, GalGraph, GalQueue, GalSet, and GalVector. Each of these classes implements GAL's basic operators in its own way (or not at all, if the operator is not applicable for that type). So, again, the difference between GalVal and GalRef is the semantics of the assignment operator. In GalRef, an assignment changes the reference of the object it contains, whereas in GalVal, an assignment changes the value of that object (that is, GalRef implements assignment by using a direct Java '=' assignment, whereas GalVal implements assignment by calling the assign() method on its object). A GalVal object could contain a GalNum, GalString, or GalBool object, and everything else would be contained in a GalRef object.

For variable declarations, as an example, `num a` would generate "IGalRef a = new GalVal(new GalNum())". Then an assignment, for example, `a <- b`, generates `a.assign(b)`. Other operators are handled similarly: `a < b` generates `a.lt(b)`, `a + b` generates `a.plus(b)`, etc. Both the front end and back end of code generation were implemented by Oren and Athar.

After all the code is generated, `galc` optionally runs an ant script which compiles the code into an executable JAR file. This ant script was written by Oren, who also integrated it into the compiler. If the developer gives the compiler a '-j' flag while compiling GAL code, this step is skipped and only the code generation occurs.

Chapter 6

Test Plan

6.1 Unit testing

We developed our own test suite in order to validate GAL code. We needed to test not only the core of the GAL language, but also the visual output that uses the JGraph API. In order to test the GAL language, we developed an automated test suite that compiled and ran GAL programs and compared the actual output to the expected output. To test the visual output, we used a GAL standard library function to display the graph and analyzed it to verify correctness. Jay developed the testing suite. He and Shep wrote the automated tests while Oren and Athar wrote the visual tests.

6.1.1 Automated Testing

The automated testing suite was developed primarily for two reasons: to locate incorrect design within the GAL language and to verify that a change in the language did not invalidate any previously functioning code. In order to test a GAL program, a user simply needs to add a file with the expected output of the program. This file should be the exact filename of the GAL file with ".out" appended to the filename. In other words, if test.gal were a program, then test.gal.out would be the expected output of the program. The testing script, test_gal_run, is located within the bin directory and searches for these output files. When it finds an output file, it compiles and runs the GAL program and compares the actual output of the file with the expected output contained in the ".out" file. If the results are the same, the test passes; otherwise, it fails and shows the inconsistency. There are over a dozen test cases that validate the GAL language. They range from simple assignment tests to more complicated scoping and array access issues.

6.1.2 Visual Testing

Visual testing was accomplished by using the JGraph library. In order to display any graph, the GAL standard library function call show() is used. It acts as a wrapper and gives the GAL language the ancillary benefit of displaying graphs visually. In order to test the visual aspect of the GAL language, we displayed simple graphs with a few vertices and edges. Once we were convinced the topology was correct, we tested more complicated graphs. First, we implemented the depth first search algorithm and displayed it. The displayed GAL graph was identical to the graph in Introduction to Algorithms by CLRS. We were confident that GAL worked. Excited by GAL's functionality, we tested Prim's minimum spanning tree and Dijkstra's shortest path algorithms. Again, GAL succeeded.

Assertions

Assertions were used in the code to maintain the integrity of the graph data structure. Since the graph (GalGraph.java) is an augmented version of a JGraphT graph, it was necessary to ensure that the vertex and edge sets never become inconsistent with one another. To that end we placed assertions in the code after any operation that modifies the graph. The assertions test the invariant specified for the graph (namely that vertex and edge sets we store match those stored by JGraphT), and were not violated in any of our tests on the final release.

6.1.3 Errors detected

A number of errors were detected by the testing suite, too many to enumerate. The most notable errors occurred when testing GAL nums. Specifically, we assigned various values to different nums and did simple arithmetic operations on them. The expected output and the actual output differed greatly and alerted us. It was at this point that we realized we wanted to pass some parameters by reference and some by value. To solve the problem, we modified the GAL language by adding a layer of indirection.

6.1.4 Test Cases

We attempted to exhaustively test the language with test cases for every type and operator. Then we chose expressions that we felt were representative of the power afforded by the language and tested those. With over 40 tests (each one of which exercises a significant portion of the GAL language), we can't include them all here. Below is a sample test case that proved especially useful in catching an error caused by a change to the semantics of our primitive types.

Automated Test of Gal num type

```
#
# A GAL program that tests numbers
#
# Editted by Shepard Saltzman

MAIN()
        num Iter
        num One
        num Two
        num Three
        num Four
        num Five
        num Six
        num Seven
        num Eight
        num Nine
        num Ten

        One <- 0
        Two <- 0
        Three <- 0
        Four <- 0
        Five <- 0
        Six <- 0
        Seven <- 0
        Eight <- 0
        Nine <- 0
        Ten <- 0

        #test addition
        One <- One + 1

        #test addition and subtraction
        Two <- One + One + One - One

        #test multiplication
        Three <- One * Two + One

        #test division
        Four <-  Two * Two * Two / (One + One)

        #test mod
        Five <- (Four * Two + Three) % (Three * Two)

        #test pluseq
        Six += (Two * Three)

        #test minuseq
        Seven <- Four * Three
        Seven -= Five

        #test timeseq
        Eight <- One
        Eight *= (Two * Four)

        #test diveq
        Nine <- Eight * Two + Two
        Nine /= Two

        #test modeq
        Ten <- Nine * Two + Three
        Ten %= (Five * Two + One)

        #Print the numbers
        println("Printing numbers 1 - 10")
        println(One)
        println(Two)
        println(Three)
        println(Four)
        println(Five)
        println(Six)
        println(Seven)
        println(Eight)
        println(Nine)
        println(Ten)

        println("Evaluating booleans")
        #test lt
        if(One < Two)
                println("1 < 2")
        else
                println("Error evaluating lt")

        #test gt
        if(Two > One)
                println("2 > 1")
        else
                println("Error evaluating gt")


        #test eq
        if(Three = Three)
                println("3 = 3")
        else
                println("Error evaluating eg")

        #test ge
        if(Four >= Four)
                println("4 >= 4")
        else
                println("Error evaluating ge")

        #test le
        if(Five <= Five)
                println("5 <= 5")
        else
                println("Error evaluating le")

        #test ne
        if(Six != Seven)
                println("6 != 7")
        else
                println("Error evaluating ne")
```

6.2 Integration Testing

6.2.1 GAL programs

The following programs were adapted from the classic *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein. The GAL implementations were demonstrated to function correctly by testing against both simple graphs on which the correct solution can be verified by inspection, as well as more complicated examples derived from the textbook.

Depth First Search

```
<<<<<<< DFS.gal
queue Q # path

=======
# written by Athar Abdul-Quader
# adapted from example in white paper, which was written by Oren Yeshua

>>>>>>> 1.5
DFS(G)
        foreach (u in G.V)
                u.visited <- false         # initialize the vertices

        foreach (u in G.V)             # for all
                if (u.visited = false)      # unconnected components of G
                        DFS_VISIT(G, u)              # begin traversal

DFS_VISIT(G, u)
        #println(u)                     # output vertex info to the terminal
        enqueue(Q,u)
        u.visited <- true                # mark node as visited
        foreach (v in adj(u))
                if (v.visited = false)
                        DFS_VISIT(G, v)        # recurse

MAIN()
        graph G                        # declare a graph
        G.V += {1,2,3,4,5,6,7,8}             # add vertices
        G.E += {(1,2),(1,4),(1,5)}       # add edges
        G.E += {(2,3),(2,4),(2,5)}
        G.E += {(3,4),(3,5),(3,6),(3,7),(3,8)}
        G.E += {(4,5)}
        DFS(G)                         # traverse

        show(G)

step()
        vertex v
        if(|Q| > 0)
                v <- pop(Q)
                v.color <- "blue"
```

Listing 6.1: galc generated code: DFS.java

```
import static edu.columbia.plt.gal.GalStdLib.*;
import edu.columbia.plt.gal.*;

public class DFS extends GalStepper
{
public static IGalRef Q = new GalRef(new GalQueue());
```

```
static {
;
}
public static IGalRef DFS(IGalRef G)
{
try{
for (IGalRef u :  ((GalIterator)(((Iterable<IGalRef>)(G).access(”V”) ).iterator())).getLis
{
(u).setMember(”visited”, (new GalVal(new GalBool(false))));
}
for (IGalRef u :  ((GalIterator)(((Iterable<IGalRef>)(G).access(”V”) ).iterator())).getLis
{
if ( (((u).access(”visited”)).eq(new GalVal(new GalBool(false)))).getObj().castBool().getB
{
DFS_VISIT(G, u);
}
}
return null;
}catch(Exception e){e.printStackTrace();}return null;
}
public static IGalRef DFS_VISIT(IGalRef G, IGalRef u)
{
try{
enqueue(Q, u);
(u).setMember(”visited”, (new GalVal(new GalBool(true))));
for (IGalRef v :  ((GalIterator)(((Iterable<IGalRef>)adj(u) ).iterator())).getList() )
{
if ( (((v).access(”visited”)).eq(new GalVal(new GalBool(false)))).getObj().castBool().getB
{
DFS_VISIT(G, v);
}
}
return null;
}catch(Exception e){e.printStackTrace();}return null;
}
public static void main(String[] args)
{
try{
IGalRef G = new GalRef(new GalGraph());
((G).access(”V”)).pluseq(new GalRef(new GalSet(new GalVal(new GalNum(1)), new GalVal(new G
((G).access(”E”)).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(1,2)), new GalRef(ne
((G).access(”E”)).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(2,3)), new GalRef(ne
((G).access(”E”)).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(3,4)), new GalRef(ne
((G).access(”E”)).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(4,5)))));
DFS(G);
show(G, ((GalStepper)Class.forName(getClassName()).newInstance()));
}catch(Exception e){e.printStackTrace();}return;
}
public static IGalRef step()
{
try{
IGalRef v = new GalRef(new GalVertex());
if ( (((Q).length()).gt(new GalVal(new GalNum(0)))).getObj().castBool().getBool())
{
v.assign(pop(Q));
(v).setMember(”color”, (new GalVal(new GalString(”blue”))));
}
return null;
}catch(Exception e){e.printStackTrace();}return null;
}

public static String getClassName(){return "DFS";}
public void dynamicStep(){step();}
}
```

Dijkstra's Single Source Shortest Path Algorithm

```
# Dijkstra's algorithm
# written by Athar Abdul-Quader
# adapted from "Introduction to Algorithms" (CLRS)

num inf <- 999999

Dijkstra(G, r)
        queue Q
        set s
        foreach(u in G.V)
                u.d <-  inf
                u.parent <- "nil"
                insert(Q, u, u.d)
        decrease_key(Q, r, 0)
        r.d <- 0

        while (|Q| > 0)
                vertex u <- pop(Q)
                s += { u }
                foreach (v in adj(u))
                        edge e <- uedge(G, (u,v))
                        if (v.d > u.d + e.weight)
                                v.d <- u.d + e.weight
                                v.parent <- u
                                decrease_key(Q, v, v.d)

Extract_Tree(G)
        set edgeSet
        set complement
        foreach (v in G.V)
                if (v.parent != "nil")
                        vertex par <- v.parent
                        edgeSet += { uedge(G,(v, par)) }

        complement <- G.E - edgeSet
        #println( edgeSet )
        #G.E -= complement
        #G.E += edgeSet
        graph G2
        G2.V += {1 .. 9}
        G2[1].color <- "blue"
        G2.E += edgeSet
        show(G2)
```

```
MAIN()
        graph G
        G.V += {1 .. 9}
        G[1].color <- "red"
        G.E += {(1,2),(1,3),(2,3),(2,4)}
        G.E += {(3,5),(3,6),(5,6),(4,5)}
        G.E += {(4,7),(4,9),(7,8),(7,9)}
        G.E += {(8,9),(6,9)}
        show(G)
        G.E[(1,2)].weight <- 4
        G.E[(1,3)].weight <- 9
        G.E[(2,3)].weight <- 11
        G.E[(2,4)].weight <- 8
        G.E[(3,5)].weight <- 7
        G.E[(3,6)].weight <- 1
        G.E[(5,6)].weight <- 6
        G.E[(4,5)].weight <- 2
        G.E[(4,7)].weight <- 7
        G.E[(4,9)].weight <- 4
        G.E[(7,8)].weight <- 9
        G.E[(7,9)].weight <- 14
        G.E[(8,9)].weight <- 10
        G.E[(6,9)].weight <- 2

        Dijkstra(G, G[1])
        Extract_Tree(G)
```

Listing 6.2: galc generated code: DIJKSTRA.java

```
import static edu.columbia.plt.gal.GalStdLib.*;
import edu.columbia.plt.gal.*;

public class DIJKSTRA extends GalStepper
{
public static IGalRef inf = new GalVal(new GalNum());
static {
inf.assign(new GalVal(new GalNum(999999)));
}
public static IGalRef Dijkstra(IGalRef G, IGalRef r)
{
try{
IGalRef Q = new GalRef(new GalQueue());
IGalRef s = new GalRef(new GalSet());
for (IGalRef u :  ((GalIterator)(((Iterable<IGalRef>)(G).access("V") ).iterator())).getLis
{
(u).setMember("d", (inf));
(u).setMember("parent", (new GalVal(new GalString("nil"))));
insert(Q, u, (u).access("d"));
}
decrease_key(Q, r, new GalVal(new GalNum(0)));
(r).setMember("d", (new GalVal(new GalNum(0))));
while ( (((Q).length()).gt(new GalVal(new GalNum(0)))).getObj().castBool().getBool())
{
IGalRef u = new GalRef(new GalVertex());
u.assign(pop(Q));
(s).pluseq(new GalRef(new GalSet(u)));
```

```
for (IGalRef v :  ((GalIterator)(((Iterable<IGalRef>)adj(u) ).iterator())).getList() )
{
IGalRef e = new GalRef(new GalEdge());
e.assign(uedge(G, new GalRef(new GalEdge(u.getObj(),v.getObj()))));
if ( (((v).access("d")).gt(((u).access("d")).plus((e).access("weight")))).getObj().castBoo
{
(v).setMember("d", (((u).access("d")).plus((e).access("weight"))));
(v).setMember("parent", (u));
decrease_key(Q, v, (v).access("d"));
}
}
}
return null;
}catch(Exception e){e.printStackTrace();}return null;
}
public static IGalRef Extract_Tree(IGalRef G)
{
try{
IGalRef edgeSet = new GalRef(new GalSet());
IGalRef complement = new GalRef(new GalSet());
for (IGalRef v :  ((GalIterator)(((Iterable<IGalRef>)(G).access("V") ).iterator())).getLis
{
if ( (((v).access("parent")).neq(new GalVal(new GalString("nil")))).getObj().castBool().ge
{
IGalRef par = new GalRef(new GalVertex());
par.assign((v).access("parent"));
(edgeSet).pluseq(new GalRef(new GalSet(uedge(G, new GalRef(new GalEdge(v.getObj(),par.getO
}
}
complement.assign(((G).access("E")).minus(edgeSet));
IGalRef G2 = new GalRef(new GalGraph());
((G2).access("V")).pluseq(new GalRef(new GalSet(1,9)));
((G2).index(new GalVal(new GalNum(1)))).setMember("color", (new GalVal(new GalString("blue
((G2).access("E")).pluseq(edgeSet);
show(G2, ((GalStepper)Class.forName(getClassName()).newInstance()));
return null;
}catch(Exception e){e.printStackTrace();}return null;
}
public static void main(String[] args)
{
try{
IGalRef G = new GalRef(new GalGraph());
((G).access("V")).pluseq(new GalRef(new GalSet(1,9)));
((G).index(new GalVal(new GalNum(1)))).setMember("color", (new GalVal(new GalString("red")
((G).access("E")).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(1,2)), new GalRef(ne
((G).access("E")).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(3,5)), new GalRef(ne
((G).access("E")).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(4,7)), new GalRef(ne
((G).access("E")).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(8,9)), new GalRef(ne
show(G, ((GalStepper)Class.forName(getClassName()).newInstance()));
(((G).access("E")).index(new GalRef(new GalEdge(1,2)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(1,3)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(2,3)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(2,4)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(3,5)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(3,6)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(5,6)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(4,5)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(4,7)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(4,9)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(7,8)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(7,9)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(8,9)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(6,9)))).setMember("weight", (new GalVal(ne
Dijkstra(G, (G).index(new GalVal(new GalNum(1))));
Extract_Tree(G);
}catch(Exception e){e.printStackTrace();}return;
}

public static String getClassName(){return "DIJKSTRA";}
public void dynamicStep(){step();}
}
```

Prim's Minimum Spanning Tree Algorithm

```
#
# Prim's Minimum Spanning Tree Algorithm
#
# Adapted from "Introduction to Algorithms" (CLRS)
#
num inf <- 999999
graph mst

MST_Prim(G, r)
        queue Q
        foreach(u in G.V)
                u.key    <-  inf
                u.parent <- "nil"
                insert(Q, u, u.key)

        decrease_key(Q, r, 0)                # sets r to be the top of queue

        while(|Q| > 0)
                vertex u <- pop(Q)
                foreach(v in adj(u))
                        edge e <- uedge(G, (u,v))
                        num w <- e.weight
                        if( (v in Q) and w < v.key)
                                v.parent <- u
                                v.key <- w
                                decrease_key(Q, v, w)

Extract_Tree(G)
        set edgeSet
        set complement
        foreach (v in G.V)
                if (v.parent != "nil")
                        println(v.parent)
                        vertex par <- v.parent
                        edgeSet += { uedge(G,(v, par)) }

#        complement <- G.E - edgeSet
```

```
        #println( edgeSet )
        #println( complement )
        #G.E -= complement
        #G.E += edgeSet
        #println( G.E )
        graph G2
        G2.V += {1 .. 9}
        G2.E += edgeSet
#        show(G2)
        mst <- G2

MAIN()
        graph G
        G.V += {1 .. 9}
        G.E += {(1,2),(1,3),(2,3),(2,4)}
        G.E += {(3,5),(3,6),(5,6),(4,5)}
        G.E += {(4,7),(4,9),(7,8),(7,9)}
        G.E += {(8,9),(6,9)}
        show(G)
        G.E[(1,2)].weight <- 4
        G.E[(1,3)].weight <- 9
        G.E[(2,3)].weight <- 11
        G.E[(2,4)].weight <- 8
        G.E[(3,5)].weight <- 7
        G.E[(3,6)].weight <- 1
        G.E[(5,6)].weight <- 6
        G.E[(4,5)].weight <- 2
        G.E[(4,7)].weight <- 7
        G.E[(4,9)].weight <- 4
        G.E[(7,8)].weight <- 9
        G.E[(7,9)].weight <- 14
        G.E[(8,9)].weight <- 10
        G.E[(6,9)].weight <- 2

        MST_Prim(G, G[1])
        Extract_Tree(G)

step()
        show(mst)
```

Listing 6.3: galc generated code: MST_PRIM.java

```
import static edu.columbia.plt.gal.GalStdLib.*;
import edu.columbia.plt.gal.*;

public class MST_PRIM extends GalStepper
{
public static IGalRef inf = new GalVal(new GalNum());
public static IGalRef mst = new GalRef(new GalGraph());
static {
inf.assign(new GalVal(new GalNum(999999)));
;
}
public static IGalRef MST_Prim(IGalRef G, IGalRef r)
{
try{
```

```
IGalRef Q = new GalRef(new GalQueue());
for (IGalRef u : ((GalIterator)(((Iterable<IGalRef>)(G).access("V") ).iterator())).getLis
{
(u).setMember("key", (inf));
(u).setMember("parent", (new GalVal(new GalString("nil"))));
insert(Q, u, (u).access("key"));
}
decrease_key(Q, r, new GalVal(new GalNum(0)));
while ( (((Q).length()).gt(new GalVal(new GalNum(0)))).getObj().castBool().getBool())
{
IGalRef u = new GalRef(new GalVertex());
u.assign(pop(Q));
for (IGalRef v : ((GalIterator)(((Iterable<IGalRef>)adj(u) ).iterator())).getList() )
{
IGalRef e = new GalRef(new GalEdge());
e.assign(uedge(G, new GalRef(new GalEdge(u.getObj(),v.getObj()))));
IGalRef w = new GalVal(new GalNum());
w.assign((e).access("weight"));
if ( (((v).elementOf(Q)).and((w).lt((v).access("key")))).getObj().castBool().getBool())
{
(v).setMember("parent", (u));
(v).setMember("key", (w));
decrease_key(Q, v, w);
}
}
}
return null;
}catch(Exception e){e.printStackTrace();}return null;
}
public static IGalRef Extract_Tree(IGalRef G)
{
try{
IGalRef edgeSet = new GalRef(new GalSet());
IGalRef complement = new GalRef(new GalSet());
for (IGalRef v : ((GalIterator)(((Iterable<IGalRef>)(G).access("V") ).iterator())).getLis
{
if ( (((v).access("parent")).neq(new GalVal(new GalString("nil")))).getObj().castBool().ge
{
println((v).access("parent"));
IGalRef par = new GalRef(new GalVertex());
par.assign((v).access("parent"));
(edgeSet).pluseq(new GalRef(new GalSet(uedge(G, new GalRef(new GalEdge(v.getObj(),par.getO
}
}
IGalRef G2 = new GalRef(new GalGraph());
((G2).access("V")).pluseq(new GalRef(new GalSet(1,9)));
((G2).access("E")).pluseq(edgeSet);
mst.assign(G2);
return null;
}catch(Exception e){e.printStackTrace();}return null;
}
public static void main(String[] args)
{
try{
IGalRef G = new GalRef(new GalGraph());
((G).access("V")).pluseq(new GalRef(new GalSet(1,9)));
```

```
((G).access("E")).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(1,2)), new GalRef(ne
((G).access("E")).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(3,5)), new GalRef(ne
((G).access("E")).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(4,7)), new GalRef(ne
((G).access("E")).pluseq(new GalRef(new GalSet(new GalRef(new GalEdge(8,9)), new GalRef(ne
show(G, ((GalStepper)Class.forName(getClassName()).newInstance()));
(((G).access("E")).index(new GalRef(new GalEdge(1,2)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(1,3)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(2,3)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(2,4)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(3,5)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(3,6)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(5,6)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(4,5)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(4,7)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(4,9)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(7,8)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(7,9)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(8,9)))).setMember("weight", (new GalVal(ne
(((G).access("E")).index(new GalRef(new GalEdge(6,9)))).setMember("weight", (new GalVal(ne
MST_Prim(G, (G).index(new GalVal(new GalNum(1))));
Extract_Tree(G);
}catch(Exception e){e.printStackTrace();}return;
}
public static IGalRef step()
{
try{
show(mst, ((GalStepper)Class.forName(getClassName()).newInstance()));
return null;
}catch(Exception e){e.printStackTrace();}return null;
}

public static String getClassName(){return "MST_PRIM";}
public void dynamicStep(){step();}
}
```

Chapter 7

Lessons Learned

7.1 Athar "Cauchy-Schwarz" Abdul-Quader

The main issue I learned is to never try to re-invent the wheel. I had a number of issues getting the parser to work correctly, until I realized that the ANTLR definition of Java grammar is freely available online. This was an invaluable resource in fixing ambiguity errors. Another major issue that I will take from this is to always define clearly the semantics of your language before going in and implementing changes. We first wanted value semantics, then reference semantics, then we realized that reference semantics would imply that changing the value of a number would change, for instance, a set containing that number. Finally, we realized that we needed to redefine semantics to be value semantics for basic types and reference semantics for the more complicated ones.

7.2 Albert Jay "Laptop" Winters

The biggest lesson that I learned was the need for a solid base of communication. It is absolutely imperative that each member communicate frequently. When many people are working together on a project of this magnitude, it becomes quite difficult to inform each member of every change that was made. The best you can do is consciously try to update everyone whenever you have changed some aspect of the code that is relevant to the entire group. I thought this problem was trivial at first, but now realize that it isn't. The only advice I would give to others is to utilize available resources, such as developing code using a popular IDE and taking advantage of a modern code control system.

7.3 Oren "My-way-or-the-highway" Yeshua

7.3.1 Practical Issues

CVS and ant were invaluable in keeping everthing and everyone in sync and allowing the group to focus on language development rather than system issues. The initial setup did require many hours of work, but the extra time invested to do things correctly from the start, more than pays itself off in the end. CVS's merge facility was a little cumbersome - it might be worth looking into a more modern version control system or at least a good GUI front end to assist with merges (emacs' emerge is not bad, but still leaves something to be desired).

7.3.2 Semantics

To reinforce Athar's comment, we ran into the same problem with graphs. Graphs are the most complex high level structure in GAL, and we had quite a bit of difficulty implementing them. It wasn't until we realized that we hadn't set out an explicit semantics for the interaction between vertex sets, edge sets, sets, and constants, that we were able to proceed to debug and verify the correctness of our code.

7.3.3 It's not just for laughs

Every problem in Computer Science really can be solved by adding another level of indirection. This litle aphorism extricated us from several difficulties we encountered during development and shortcomings in our original design.

7.4 Shep "Sea World" Saltzman

I learned that Python is awesome, but it's no excuse for not knowing Perl. I learned that regular languages are not only a really neat concept, but a solid grasp of them is actually useful to coding. Who knew? I learned to balance researching issues on my own versus asking my teammates for help. A test program that fails to produce the predicted output could be an error on my part, a small slip in our documentation or, indeed, a serious issue with the language. I learned that there is never a bad time to reconfirm that what you have so far is working as intended, be it a single file or an entire grammar. I learned that "=" is not an assignment operator in GAL. I learned that "=" is not an assignment operator in GAL. I learned that "=" is not an assignment operator in GAL. I learned that "=" is not an assignment operator in GAL. I learned that "=" is not an assignment operator in GAL. I learned that "=" is not an assignment operator in GAL. Finally, I learned there is nothing more valuable than good teamwork and communication.

Appendix A

Code Listing

A.1 Lexer

```
class GalLexer extends Lexer;
options
{
   k = 2;
   testLiterals = false;
   charVocabulary = '\3'..'\377';
   importVocab = GalAntlr;
}

// Comments
COMMENT : ('#' (~('\n'|'\r'))*)
        {$setType(Token.SKIP);};

// Operators
PLUS    : '+' ;
MINUS   : '-' ;
TIMES   : '*' ;
DIV     : '/' ;
MOD     : '%' ;

PLUSEQ  : "+=" ;
MINUSEQ : "-=" ;
TIMESEQ : "*=" ;
DIVEQ   : "/=" ;
MODEQ   : "%=" ;

GT      : '>';
LT      : '<';
EQ      : '=';
GE    : ">=";
LE    : "<=";
NEQ     : "!=";
NOT     : '!';

ASSIGN : "<-";
ACCESS : '.' ;
RANGE  : "..";
LISTSEP: ',' ;
```

```
// Brackets
LPAREN        : '(' ;
RPAREN        : ')' ;
LBRACK        : '[' ;
RBRACK        : ']' ;
LBRACE        : '{' ;
RBRACE        : '}' ;
LENGTH  : '|' ;

WILDCARD : "?";

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


// Strings
STRING
   : '"'
      ( ESCSEQ | ~('"' | '\\' | '\n' | '\r') )*
      '"'
    ;

protected ESCSEQ
   : '\\' ('\\' | '"');

// Numbers
NUM:  I ('.' I)? (E)? ;

// Exponent
protected E: 'e'('+' | '-')? I ;

// Integer/Fraction Part
protected I: (DIGIT)+ ;

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

// Discarded whitespace
WS : ((' '| '\t')+) {$setType(Token.SKIP);};

// Newline (statement terminators)
NEWLINE
    : ( '\r' '\n'   // DOS
      | '\n'        // UNIX
      ) {newline();}
    ;
```

A.2 Parser

```
class GalParser extends Parser;
options
{
        k=3;
        exportVocab = GalAntlr;
        buildAST = true;
}

tokens
{
PROG<AST=edu.columbia.plt.gal.ast.Prog>;
VDECL<AST=edu.columbia.plt.gal.ast.VDecl>;
FDECL<AST=edu.columbia.plt.gal.ast.FDecl>;
PARMS<AST=edu.columbia.plt.gal.ast.Parms>;
ARGS<AST=edu.columbia.plt.gal.ast.Args>;
FCALL<AST=edu.columbia.plt.gal.ast.FCall>;
ID<AST=edu.columbia.plt.gal.ast.Id>;
BODY<AST=edu.columbia.plt.gal.ast.Body>;
EDGEC<AST=edu.columbia.plt.gal.ast.EdgeConst>;
SETC<AST=edu.columbia.plt.gal.ast.SetConst>;
NUM<AST=edu.columbia.plt.gal.ast.Id>;
STRING<AST=edu.columbia.plt.gal.ast.StringConst>;
"true"<AST=edu.columbia.plt.gal.ast.BoolConst>;
"false"<AST=edu.columbia.plt.gal.ast.BoolConst>;
TYPE;
PLUS<AST=edu.columbia.plt.gal.ast.BinaryOp>;
MINUS<AST=edu.columbia.plt.gal.ast.BinaryOp>;
TIMES<AST=edu.columbia.plt.gal.ast.BinaryOp>;
DIV<AST=edu.columbia.plt.gal.ast.BinaryOp>;
MOD<AST=edu.columbia.plt.gal.ast.BinaryOp>;
PLUSEQ<AST=edu.columbia.plt.gal.ast.BinaryOp>;
MINUSEQ<AST=edu.columbia.plt.gal.ast.BinaryOp>;
TIMESEQ<AST=edu.columbia.plt.gal.ast.BinaryOp>;
DIVEQ<AST=edu.columbia.plt.gal.ast.BinaryOp>;
MODEQ<AST=edu.columbia.plt.gal.ast.BinaryOp>;
GT<AST=edu.columbia.plt.gal.ast.BinaryOp>;
LT<AST=edu.columbia.plt.gal.ast.BinaryOp>;
EQ<AST=edu.columbia.plt.gal.ast.BinaryOp>;
GE<AST=edu.columbia.plt.gal.ast.BinaryOp>;
LE<AST=edu.columbia.plt.gal.ast.BinaryOp>;
NEQ<AST=edu.columbia.plt.gal.ast.BinaryOp>;
NOT<AST=edu.columbia.plt.gal.ast.UnaryOp>;
ASSIGN<AST=edu.columbia.plt.gal.ast.Assign>;
LBRACK<AST=edu.columbia.plt.gal.ast.Index>;
LENGTH<AST=edu.columbia.plt.gal.ast.UnaryOp>;
ACCESS<AST=edu.columbia.plt.gal.ast.AccessOp>;
"and"<AST=edu.columbia.plt.gal.ast.BinaryOp>;
"or"<AST=edu.columbia.plt.gal.ast.BinaryOp>;
"while"<AST=edu.columbia.plt.gal.ast.While>;
"if"<AST=edu.columbia.plt.gal.ast.If>;
"else"<AST=edu.columbia.plt.gal.ast.Else>;
"foreach"<AST=edu.columbia.plt.gal.ast.Foreach>;
"return"<AST=edu.columbia.plt.gal.ast.Return>;
WILDCARD<AST=edu.columbia.plt.gal.ast.Wildcard>;
}

program :
        ((varDecl | funDecl) NEWLINE! | NEWLINE!)* EOF!
        {#program = #([PROG,"_prog"], #program);};
```

```
varDecl!:
        t:type r:varDeclRHS
        {#varDecl = #([VDECL,"_vdecl"], t, r);};

varDeclRHS:
        ID | assignment ;

funDecl! :
        d:funDef b:block
        {#funDecl = #([FDECL,"_fdecl"], d, b);};

funDef :
        ID LPAREN! parmList RPAREN! (NEWLINE!)*;

parmList :
        (ID<AST=edu.columbia.plt.gal.ast.VarParm> (LISTSEP! ID<AST=edu.columbia.plt.gal.ast.VarParm> )*
        {#parmList = #([PARMS,"_parms"], #parmList);};

block :
        LBRACE! (statement | NEWLINE!)* RBRACE!
        {#block = #([BODY,"_body"], #block);};

statement :
        ( expression | varDecl | ifStmt | foreach_stmt | return_stmt | while_stmt) NEWLINE!;

ifStmt :
        "if"^ LPAREN! expression RPAREN! (NEWLINE!)* block (else_stmt)? ;

else_stmt :
        "else"^ (NEWLINE!)* block ;

foreach_stmt :
        "foreach"^ LPAREN! ID "in"! expression RPAREN! (NEWLINE!)* block;

return_stmt :
        "return"^ expression;

while_stmt :
        "while"^ LPAREN! expression RPAREN! (NEWLINE!)* block ;

funCall :
        ID LPAREN! exprList  RPAREN!
        {#funCall = #([FCALL,"_fcall"], #funCall);};

exprList :
        (expression (LISTSEP! expression)*)?
        {#exprList = #([ARGS, "_args"], #exprList);};

expression : assignExpr ("in"^<AST=edu.columbia.plt.gal.ast.BinaryOp> assignExpr)?;

assignExpr! :
        l:conditionalExpr (o:assignOp r:assignExpr)?
    {#assignExpr = (#o != null) ? #(o, l, r) : #l;};

conditionalExpr:
        boolExpr ("or"^ boolExpr)* ;
boolExpr        :
        boolTerm ("and"^ boolTerm)* ;

boolTerm        :
        (NOT^)? logicExpr ;

logicExpr        :
        arithExpr ( (GE^ | LE^ | GT^ | LT^ | EQ^ | NEQ^) arithExpr )? ;

arithExpr        :
        arithTerm ( ( PLUS^ | MINUS^ ) arithTerm )* ;

arithTerm        :
        arithTerm2 ( ( TIMES^ | DIV^ | MOD^ ) arithTerm2)* ;

arithTerm2        :
        (MINUS^<AST=edu.columbia.plt.gal.ast.UnaryOp>)? generalExpr;

generalExpr        :
        lValue
        | STRING
        | NUM<AST=edu.columbia.plt.gal.ast.NumConst>
        | "true"
        | "false"
        | set_const
        | edge_const
        | LPAREN! expression RPAREN!
        | LENGTH^ expression LENGTH!
    | funCall
    ;

assignment! :
        l:lValue o:assignOp r:expression
        {#assignment = #(o, l, r);};

lValue : ID<AST=edu.columbia.plt.gal.ast.VarCheck> ((ACCESS^ ID) | (LBRACK^ expression RBRACK!))*;

assignOp :
        ASSIGN | PLUSEQ | MINUSEQ | TIMESEQ | DIVEQ | MODEQ ;

type: ("graph" | "set"    | "vertex" |
       "edge"  | "queue"  | "num"    |
       "string"| "bool"   | "vector")
       {#type = #([TYPE,"_type"], #type);};

set_const :
        LBRACE! (exprList | range_const) RBRACE!
    {
        #set_const = #([SETC,"_setc"], #set_const);
    };

range_const :
        NUM RANGE^ NUM;

edge_const :
        LPAREN! (NUM|ID<AST=edu.columbia.plt.gal.ast.Var>|WILDCARD) LISTSEP! (NUM|ID<AST=edu.columbia.p
    {#edge_const = #([EDGEC, "_edgec"], #edge_const);};
```

A.3 Preprocessor

Listing A.1: galindent.py

```
#!/usr/local/bin/python

# GAL Indentation Program
# Written by Shepard Saltzman
# Usage: galident.py [filename] (-b)
# Removes all tabbed indentation from a file. Unless -b is set,
# adds brackets to maintain blocking. Sends an error to stderr
# if spacing and tabbed indentation is mixed.


import sys

brackets = True
count = 0
linenum = 0
f = open(sys.argv[1], 'r')
if (len(sys.argv) > 2):
    if ("-b" in sys.argv):
        brackets = False
for line in f:
    current = 0
    linenum += 1
    newline = ''
    if (len(line) == 1):
#       sys.stdout.write("\nBLANK line on line " + str(linenum))
        continue
    if ((line.strip('\t␣'))[0] == '#'):
        sys.stdout.write(line.strip('\t'))
        continue
    for char in range(len(line)):
        if (line[char] == '\t'):
            current += 1
        else:
            break
#   sys.stdout.write("\ncurrent = " + str(current))
#   sys.stdout.write("\nline[current] = " + line[current])
#   sys.stdout.write("\n")
    if (line[char] == '␣'):
        sys.stderr.write("\nWS␣on␣line␣" + str(linenum))
    if ((current < count) and brackets):
        newline = ("}\n" * (count - current))
        if(line[char:(char+4)] == "else"):
            newline = newline[0:len(newline)-1]
        count = current
    elif (current > count) and brackets:
        if (current == (count + 1)):
            newline = "{"
            count += 1
        else:
            sys.stderr.write("Too many tabs on line " + str(linenum))
#     if not ("else" in line):
#         sys.stdout.write("\n")
    newline = newline + line.strip('\t ')
    sys.stdout.write(newline)
while (count > 0) and brackets:
    sys.stdout.write("}\n")
    count = count - 1
sys.stdout.write("\n")
```

A.4 Compiler

Listing A.2: galc

```
#!/usr/local/bin/perl
#@author Oren Yeshua
#@author Jay Winters
use strict;
use Getopt::Std;
my $j;
my $b;

sub usage()
{print<<USAGE;

 *** GAL Compiler ***

 galc [options] <input file>.gal

 options:
 -j  output java source code only
 -b  if the input gal program has braces for scope
USAGE
}


my %opts; getopts('jb', \%opts);
$j = $opts{j} ? "false" :  "true";
$b = $opts{b} ? "true" : "false";
#make indent script executable
system("chmod +x $ENV{GAL_WD}/src/galindent.py");


    if(-T $ARGV[0] && $ARGV[0] =~ m/\.gal/)
    {
        system("java -classpath $ENV{GAL_WD}/lib/antlr.jar:$ENV{GAL_WD}/bin/class ".
          "galc $ARGV[0] ".$j . " $b");
    }
    else
    {
        usage();
    }
```

Listing A.3: galc.java

```
import java.io.*;
import java.lang.Process;
import java.lang.Runtime;
import antlr.CommonAST;
import antlr.collections.AST;
import edu.columbia.plt.gal.SymbolTable;
import edu.columbia.plt.gal.ast.Prog;
import edu.columbia.plt.gal.ast.Node;
/**
 * GAL compiler
 *
 * Installation requirements:
 *
 * In order to build and run executable jars,
 * $GAL_WD must be set to a directory containing at least
 *
 *  bin/
 *  lib/gal.jar
 *  lib/jgraph.jar
 *  lib/jgrapht-jdk1.5.jar
 *  make/galc.xml
 *
 * If these requirements are met, the jar will
 * be created and placed in
 *
 *  bin/launch/
 *
 * To execute the jar outside of the launch directory
 * (or to compile generated java code),
 * the 3 jar files from the lib directory must be added
 * to the classpath.
 *
 * @author Oren Yeshua
 * @author Jay Winters
 */
public class galc
{



    //Used to pipe the errors from galindent to standard error
    private static void pipe(final InputStream in, final PrintStream out,
                  final String prefix) {

            BufferedReader r
                = new BufferedReader(new InputStreamReader(in));
            try {
                boolean wasError = false;
                for (;;) {
                    String s = r.readLine();
                    if (s == null ) {
                        break;
                    }
                    out.println(prefix + s);
                    wasError = true;
                }
                //If there was an error, exit
                if(wasError) System.exit(-1);
            } catch (java.io.IOException e) {
                e.printStackTrace(System.err);
            }

    }


    public static void main(String[] args)
    {

        boolean createJar = Boolean.valueOf(args[1]);
        boolean hasBraces = Boolean.valueOf(args[2]);

        String className = null;

        File srcFile = null;
        File outFile = null;

        DataInputStream  dis = null;
        FileInputStream  fis = null;
        PrintWriter      out = null;

        try
        {
            srcFile = new File(args[0]);
            String fileName = srcFile.getName();
            className = fileName.substring(0, fileName.indexOf('.'));
            outFile = new File(srcFile.getParent(), className+".java");
            fis = new FileInputStream(srcFile);
            out = new PrintWriter(outFile);
          if(hasBraces)
          {
            // get the source file directly
            dis = new DataInputStream(fis);
           }
          else
           {
            // Get the input from the output of galindent
            String galindent = "python␣"+System.getenv("GAL_WD") + "/src/galindent.py␣" +
            Process run_indent = Runtime.getRuntime().exec(galindent);
              dis = new DataInputStream(run_indent.getInputStream());
              pipe(run_indent.getErrorStream(),System.err,"galindent␣ERROR␣");

           }

            // Create the lexer and parser
            GalLexer lexer = new GalLexer(dis);
            GalParser parser = new GalParser(lexer);
            parser.setASTNodeClass("edu.columbia.plt.gal.ast.Node");
            // Parse
            parser.program();

            // Get the AST from the parser
            Prog parseTree = (Prog)parser.getAST();

            // Create environment -- static semantic checking
            parseTree.setEnv(new SymbolTable<Node>(null), false);

            // Generate Java code
            out.println("import␣static␣edu.columbia.plt.gal.GalStdLib.*;");
            out.println("import␣edu.columbia.plt.gal.*;\n");

            out.println("public␣class␣"+className+"␣extends␣GalStepper");
            out.println('{');

            //GalTreeParser codeGenerator = new GalTreeParser();
            //System.out.println(codeGenerator.prog(parseTree));
            out.print(parseTree.gen());

            out.println("\npublic␣static␣String␣getClassName(){return␣\""+className+"\";}"
            out.println("public␣void␣dynamicStep(){step();}");
            out.println('}');
        }
        catch(Exception e)
        {
            //System.err.println("Exception: "+e);
            e.printStackTrace(); // For debug only
        }
        finally
        {
            try
            {
                dis.close();
                fis.close();
                out.close();
            }
            catch(IOException ignored){}
        }

        if(createJar)
        {
            // Compile java
            InputStream       ant_is = null;
            InputStreamReader ant_isr = null;
            BufferedReader    ant_br = null;

            try
            {
                String[] ant_cmd =
                    {"ant",
                     "-DsrcDir="+outFile.getAbsoluteFile().getParent(),
                     "-DsrcName="+className,
                     "-buildfile", System.getenv("GAL_WD")+
                      File.separator+"make"+
                      File.separator+"galc.xml"
                    };

                Process ant_proc = Runtime.getRuntime().exec(ant_cmd);
                ant_is = ant_proc.getInputStream();
                ant_isr = new InputStreamReader(ant_is);
                ant_br = new BufferedReader(ant_isr);
                ant_proc.waitFor();       // wait for build to complete

                String ant_line=null; // pipe stream
                while((ant_line = ant_br.readLine()) != null)
                {
                    System.out.println(ant_line);
                }
            }
            catch(Exception e)
            {
                System.err.println("Error compiling "+
                                   outFile.getName()+" : "+e);
            }
            finally
            {
                try
                {
                    ant_br.close();
                    ant_isr.close();
                    ant_is.close();
                }
                catch(IOException ignored){}
            }
        }
    }

    /**
     * For debugging purposes only
     */
    private static void printTypeTree(AST node)
    {
        if(node != null)
        {
            System.out.print("("+node.getClass().getName());
            for(AST child = node.getFirstChild();
                child != null;
                child = child.getNextSibling())
            {
                System.out.print(' ');
                printTypeTree(child);
            }
            System.out.print(')');
        }
    }
}
```

A.4.1 package edu.columbia.plt.gal

Listing A.4: ArrayType.java

```
package edu.columbia.plt.gal;

public class ArrayType extends Type
{
        public void ArrayType(Type sub)
        {
                currentType = new ArrayType();
                subType = sub;
        }
}
```

Listing A.5: BoolType.java

```
package edu.columbia.plt.gal;

public class BoolType extends Type
{
        public void BoolType()
        {
                currentType = new BoolType();
                subType = null;
        }
}
```

Listing A.6: EdgeType.java

```
package edu.columbia.plt.gal;

public class EdgeType extends Type
{
        public void EdgeType()
        {
                currentType = new EdgeType();
                subType = null;
        }
}
```

Listing A.7: GalBool.java

```
package edu.columbia.plt.gal;

/**
 * Gal Bool type
 * @author Athar Abdul-Quader
 */
public class GalBool extends GalType
{
        public static GalBool oTrue = new GalBool(true);
        public static GalBool oFalse = new GalBool(false);

        protected boolean mBool;
        public GalBool() { }

        public GalBool(boolean bool)
        {
                mBool = bool;
        }

        public boolean getBool()
        {
                return mBool;
        }

        public T getType()
        {
                return T.BOOL;
        }

        // assignment op
        public GalBool assign(GalType rhs)
        {
                GalBool bool = rhs.castBool();
                mBool = bool.getBool();
                if (mBool) return oTrue;
                else return oFalse;
        }

        // comparison
        public GalBool eq(GalType rhs)
        {
                if (rhs instanceof GalBool)
                {
                        GalBool bool = rhs.castBool();
                        if(mBool == bool.getBool())
                                return oTrue;
                }
                return oFalse;
        }

        public GalBool neq(GalType rhs)
        {
                if (rhs instanceof GalBool)
                {
                        GalBool bool = rhs.castBool();
                        if (mBool == bool.getBool())
                                return oFalse;
                }
                return oTrue;
        }

        // boolean ops
        public GalBool and(GalType rhs)
        {
                GalBool bool = rhs.castBool();
                if (mBool && bool.getBool())
                        return oTrue;
                else return oFalse;
        }

        public GalBool or(GalType rhs)
        {
                GalBool bool = rhs.castBool();
                if (mBool || bool.getBool())
                        return oTrue;
                else return oFalse;
        }
        public GalBool not()
        {
                if (!mBool)
                        return oTrue;
                else return oFalse;
        }

        public String toString()
        {
                return String.valueOf(mBool);
        }

        public boolean equals(Object o)
        {
                if (o instanceof GalBool)
                {
                        return mBool == ((GalBool)o).getBool();
                }
                return false;
        }

        public int hashCode()
        {
                return (new Boolean(mBool)).hashCode();
        }
}
```

Listing A.8: GalEdge.java

```
package edu.columbia.plt.gal;


/**
 * Gal Edge type
 * @author Oren B. Yeshua
 */
public class GalEdge extends GalType
{
    protected GalVertex mTail;
    protected GalVertex mHead;
    protected GalGraph mGraph;

    /**
     * String representation
     *
     * Stored explicitly for efficiency in visualization frame
     */
    protected String mStringRep;

    public GalEdge()
    {}

    public GalEdge(int inTail, int inHead)
    {
        mTail = new GalVertex(inTail);
        mHead = new GalVertex(inHead);
        computeStringRep();
    }

    public GalEdge(int inTail, String inWildcard)
    {
        mTail = new GalVertex(inTail);
        mHead = new GalVertex(inWildcard);
        computeStringRep();
    }

    public GalEdge(String inWildcard, int inHead)
    {
        mTail = new GalVertex(inWildcard);
        mHead = new GalVertex(inHead);
        computeStringRep();
    }

    public GalEdge(GalType inTail, GalType inHead)
    {
        mTail = inTail.castVertex();
        mHead = inHead.castVertex();
        computeStringRep();
    }

    public GalEdge(GalType inTail, String inWildcard)
    {
        this(inTail.castVertex(), new GalVertex(inWildcard));
    }

    public GalEdge(String inWildcard, GalType inHead)
    {
        this(new GalVertex(inWildcard), inHead.castVertex());
    }

    public T getType()
    {
        return T.EDGE;
    }

    public GalGraph getGraph()
    {
        return mGraph;
    }

    public GalType access(String id)
    {
        GalType rv = null;
        if(id.equals("head"))
            rv = mHead;
        else if(id.equals("tail"))
            rv = mTail;
        else
            rv = super.access(id);

        return rv;
    }
    public GalVertex getHead()
    {
        return mHead;
    }

    public GalVertex getTail()
    {
        return mTail;
    }

    public void setHead(GalVertex v)
    {
        mHead = v;
        computeStringRep();
    }

    public void setTail(GalVertex v)
    {
        mTail = v;
        computeStringRep();
    }

    public void setGraph(GalGraph g)
    {
        mGraph = g;
    }

    public GalType length()
    {
        GalType w = getMember("weight");
        if(w != null)
        {
            if(w.getType() == T.NUM)
                w = new GalNum().assign(w);
            else
                w = w.length();
        }
        else
            w = new GalNum(0);

        return w;
    }

    public String toString()
    {
        return "("+mTail+','+mHead+')'+(mGraph == null ? '*' : "")+(getMember("weight") !=
    }

    public String stringRep()
    {
        return mStringRep;
    }

    protected void computeStringRep()
    {
        mStringRep = "("+mTail.stringRep()+','+mHead.stringRep()+')';
    }

    public GalType constantify()
    {
        return new GalEdge(mTail.getIndex(), mHead.getIndex());
    }

    public boolean equals(Object o)
    {
        boolean rv = false;
        if(o instanceof GalEdge)
        {
            GalEdge e = (GalEdge)o;
            if(mTail.equals(e.mTail) && mHead.equals(e.mHead))
            {
                // Edge const - by value semantics
                if(mGraph == null && e.mGraph == null)
                {
                    rv = true;
                }
                else if(mGraph == e.mGraph)
                {// Edge - by reference semantics
                    rv = true;
                }
            }
        }
        return rv;
    }

    public int hashCode()
    {
        int code = super.hashCode();
        if(mGraph == null)
        {
//System.out.println("Why here?");
            code = (mHead.hashCode() << 16) + mTail.hashCode();
        }
        return code;
    }
}
```

Listing A.9: GalEdgeSet.java

```
package edu.columbia.plt.gal;

import edu.columbia.plt.gal.err.GalRuntimeException;

import java.util.Hashtable;
import java.util.HashSet;
import java.util.Iterator;
import java.lang.Iterable;

/**
 * Gal Edge Set (Set internal to graph)
 *
 * Performs indexing specific to an edge set
 * @author Oren B. Yeshua
 */
public class GalEdgeSet extends GalGraphSet
{
    public GalEdgeSet(GalGraph g){super(g);}

    public GalType index(GalType i)
    {
        GalType rv = null;
        switch(i.getType())
        {
        case NUM:
        case VERTEX:
            rv = mGraph.allEdgesOf(mGraph.index(i).castVertex());
            break;
        case EDGE:
            GalEdge e = i.castEdge();
            if(e.getTail().isWild())
            {
                rv = mGraph.incomingEdgesOf(mGraph.index(e.getHead()).castVertex());
            }
            else if(e.getHead().isWild())
            {
                rv = mGraph.outgoingEdgesOf(mGraph.index(e.getTail()).castVertex());
            }
            else
                rv = mElementIndex.get(i.stringRep());
            break;
        default:
            rv = mElementIndex.get(i.stringRep());
        }
        return rv;
    }

//      public GalSet getGalSet()
//      {
//       HashSet<GalType> edgeSet = new HashSet<GalType>();

//       for (GalType i : mElementIndex.values())
//       {
//               GalEdge e = new GalEdge(i.castEdge().getTail().getIndex(), i.castEdge().ge
//               edgeSet.add(e);
//       }

//       return new GalSet(edgeSet);

//      }

}
```

Listing A.10: GalGraph.java

```
package edu.columbia.plt.gal;

import edu.columbia.plt.gal.err.GalRuntimeException;

import org.jgrapht.*;
import org.jgrapht.graph.*;
```

```java
import java.util.Set;
import java.util.HashSet;

/**
 * Gal Graph type
 * Wrapper for JGraphT DirectedGraph
 * @author Oren B. Yeshua
 */
public class GalGraph extends GalType implements EdgeFactory<GalVertex,GalEdge>
{
    /**
     * JGraphT graph
     */
    protected ListenableDirectedGraph<GalVertex, GalEdge> mJGraphT;

    protected GalGraphSet mVertexSet;
    protected GalGraphSet mEdgeSet;

    public GalGraph()
    {
        mJGraphT = new ListenableDirectedGraph<GalVertex, GalEdge>(
                       new DefaultDirectedGraph<GalVertex,GalEdge>(
                           this));

        mVertexSet = new GalVertexSet(this);
        mEdgeSet = new GalEdgeSet(this);
    }

    public T getType()
    {
        return T.GRAPH;
    }

    public DirectedGraph getJGraphT()
    {
        return mJGraphT;
    }

    /**
     * Insert graph element t into the graph
     */
    public GalType insert(GalType t)
    {
        GalType rv = null;

        switch(t.getType())
        {
        case NUM:
            rv = new GalVertex(t.castNum().getInt(), this);
            mJGraphT.addVertex(rv.castVertex());
            break;
        case VERTEX:
            rv = t;
            rv.castVertex().setGraph(this);
            mJGraphT.addVertex(rv.castVertex());
            break;
        case EDGE:
            GalEdge e = t.castEdge();
            e.setHead(index(e.getHead()).castVertex());
            e.setTail(index(e.getTail()).castVertex());
            e.setGraph(this);
            mJGraphT.addEdge(e.getTail(), e.getHead(), e);
            rv = mJGraphT.getEdge(e.getTail(),e.getHead());          ;
            break;
        default:
            throw new GalRuntimeException("Unable␣to␣insert␣"+t.getType()+"␣into␣graph.");
        }

        return rv;
    }

    /**
     * Remove graph element t from the graph
     */
    public void remove(GalType t)
    {
        switch(t.getType())
            {
            case VERTEX:
                // Remove adjacent edges from edge set
                for(GalEdge e : mJGraphT.edgesOf(t.castVertex()))
                    {
                        mEdgeSet.remove(e);
                    }
                mJGraphT.removeVertex(t.castVertex());
                break;
            case EDGE:
                mJGraphT.removeEdge(t.castEdge());
                break;
            default:
                throw new GalRuntimeException("Unable␣to␣remove␣"+t.getType()+"␣from␣graph
            }
    }

    public GalType index(GalType rhs)
    {
        GalType rv = null;
        switch(rhs.getType())
        {
        case NUM:
        case VERTEX:
            rv = mVertexSet.index(rhs);
            break;
        case EDGE:
            rv = mEdgeSet.index(rhs);
            break;
        default:
            throw new GalRuntimeException("Unable␣to␣index␣graph␣with␣"+rhs.getType()+".")
        }
        return rv;
    }
```

```
public GalType access(String id)
{
    GalType rv = null;
    if(id.equals("V"))
        rv = mVertexSet;
    else if(id.equals("E"))
        rv = mEdgeSet;
    else
        rv = super.access(id);

    return rv;
}

public GalType elementOf(GalType t)
{
    GalType rv = new GalBool(false);
    switch(t.getType())
    {
    case NUM:
    case VERTEX:
        rv = mVertexSet.elementOf(t);
        break;
    case EDGE:
        rv = mEdgeSet.elementOf(t);
        break;
    }
    return rv;
}

/**
 * Display in visualization frame
 */
public void show(GalStepper s)
{
    new GalVisFrame(mJGraphT, s);
}


public GalEdge createEdge(GalVertex tail, GalVertex head)
{
    return new GalEdge(tail,head);
}

public GalSet incomingEdgesOf(GalVertex v)
{
    return new GalSet(mJGraphT.incomingEdgesOf(v));
}

public GalSet outgoingEdgesOf(GalVertex v)
{
    return new GalSet(mJGraphT.outgoingEdgesOf(v));
}

public GalSet allEdgesOf(GalVertex v)
{
    return new GalSet(mJGraphT.edgesOf(v));
}

public GalSet incomingAdjacentVerticesOf(GalVertex v)
{
    Set<GalVertex> inVertices = new HashSet<GalVertex>();
    for(GalType t : incomingEdgesOf(v))
    {
        inVertices.add(t.castEdge().getTail());
    }

    return new GalSet(inVertices);
}

public GalSet outgoingAdjacentVerticesOf(GalVertex v)
{
    Set<GalVertex> outVertices = new HashSet<GalVertex>();
    for(GalType t : outgoingEdgesOf(v))
    {
        outVertices.add(t.castEdge().getHead());
    }

    return new GalSet(outVertices);
}

public GalSet allAdjacentVerticesOf(GalVertex v)
{
    return incomingAdjacentVerticesOf(v).pluseq(outgoingAdjacentVerticesOf(v));
}

public GalNum degreeOf(GalVertex v)
{
    GalVertex vert = index(v).castVertex();
    return new GalNum(mJGraphT.outDegreeOf(vert)+
                      mJGraphT.inDegreeOf(vert));
}

/**
 * Retrieves an edge from the graph as if it were undirected
 *
 * uedge(e)
 * @returns returns (e.tail,e.head) if present in the graph, otherwise (e.head,e.tail)
 */
public GalEdge uedge(GalType t)
{
    GalEdge inEdge = t.castEdge();
    GalType edge = mEdgeSet.index(inEdge);
    GalEdge e; //= mEdgeSet.index(inEdge).castEdge();
    if(edge == null)
    {
        e = mEdgeSet.index(new GalEdge(inEdge.castEdge().getHead(),inEdge.castEdge().g
    }
    else e = edge.castEdge();

    if(e == null)
        throw new GalRuntimeException("Undirected␣edge␣"+inEdge+"␣not␣found␣in␣graph."
    return e;
}

public String toString()
{
    StringBuilder sb = new StringBuilder("#G");
    sb.append(mVertexSet);
    sb.append(mEdgeSet);
    return sb.toString();
}

/**
 * Invariant assertion
 *
 * A GalGraph's GalVertexSet and GalEdgeSet should contain the same
 * vertices and edges as its JGraphT
 */
public boolean isConsistent()
{
    boolean rv = true;
    System.out.print("Asserting ...");

    // Check for vertex constants in the JGraphT
    for(GalVertex v : mJGraphT.vertexSet())
    {
        if(v.getGraph() != this)
        {
            System.err.println("Vertex const "+v+" detected in JGraphT");
            rv = false;
        }
    }

    // Check for edge constants in the JGraphT
    for(GalEdge e : mJGraphT.edgeSet())
    {
        if(e.getGraph() != this)
        {
            System.err.println("Edge const "+e+" detected in JGraphT");
            rv = false;
        }
    }

    // Check that the vertex sets are equivalent
    if(!(mJGraphT.vertexSet().containsAll(mVertexSet.getSet())))
    {
        System.err.println("JGraphT does not contain all VSet v's");
        System.err.println("VSet: "+mVertexSet.getSet());
        System.err.println("JGraphT: "+mJGraphT.vertexSet());
        rv = false;
    }

    if(!(mVertexSet.getSet().containsAll(mJGraphT.vertexSet())))
     {
        System.err.println("VSet does not contain all JGraphT v's");
        System.err.println("VSet: "+mVertexSet.getSet());
        System.err.println("JGraphT: "+mJGraphT.vertexSet());
        rv = false;
     }

    // Check that the edge sets are equivalent
    if(!(mJGraphT.edgeSet().containsAll(mEdgeSet.getSet())))
    {
        System.err.println("JGraphT does not contain all EdgeSet edges");
        rv = false;
    }

    if(!(mEdgeSet.getSet().containsAll(mJGraphT.edgeSet())))
    {
        System.err.println("EdgeSet does not contain all JGraphT edges");
        System.err.println("EdgeSet: "+mEdgeSet.getSet());
        System.err.println("JGraphT: "+mJGraphT.edgeSet());
        rv = false;
    }

    if(rv)
        System.out.println("Passed.");

    return rv;
}
}
```

Listing A.11: GalGraphSet.java

```
package edu.columbia.plt.gal;

import java.util.Hashtable;
import java.util.HashSet;
import java.util.Set;
import java.util.Iterator;
import java.lang.Iterable;

/**
 * Gal Graph Set (Set internal to graph)
 *
 * Maintains reference to parent graph
 * and lookup table for elements
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public abstract class GalGraphSet extends GalType implements Iterable<GalType>
{
    protected GalGraph mGraph;

    Hashtable<String, GalType> mElementIndex;

    public GalGraphSet(GalGraph g)
    {
        mElementIndex = new Hashtable<String, GalType>();
        mGraph = g;
    }

    public T getType()
    {
        return T.GRAPHSET;
    }

    public GalType index(GalType i)
    {
        return mElementIndex.get(i.stringRep());
    }

    /**
     * Union
     */
    public GalType pluseq(GalType rhs)
    {
        switch(rhs.getType())
        {
        case SET:
        case VECTOR:
            for(GalType t : (Iterable<GalType>)rhs)
            {
                addToGraph(t);
            }
            break;
        default:
            addToGraph(rhs);
        }

        // Check invariant
        assert mGraph.isConsistent() : "+=␣:␣graph␣inconsistency␣detected";

        return this;
    }


    /**
     * Difference
     */
    public GalType minuseq(GalType rhs)
    {
        switch(rhs.getType())
        {
        case SET:
        case VECTOR:
            for(GalType t : (Iterable<GalType>)rhs)
            {
                removeFromGraph(t);
            }
            break;
        default:
            removeFromGraph(rhs);
        }

        // Check invariant
        assert mGraph.isConsistent() : "-=␣:␣graph␣inconsistency␣detected";

        return this;
    }

    /**
     * Intersection
     * Returns the elements of rhs that are contained in this set
     */
    public GalType timeseq(GalType rhs)
    {
        GalSet aSymmDiff = div(rhs).minus(rhs);
        minuseq(indexSet(aSymmDiff));

        // Check invariant
        assert mGraph.isConsistent() : "*=␣:␣graph␣inconsistency␣detected";

        return this;
    }

    /**
     * Symmetric Difference
     */
    public GalType diveq(GalType rhs)
    {
//System.out.println("/G.V"+this);
//System.out.println("/rhs"+rhs);
        GalSet intersection = times(rhs);
//System.out.println("/int:"+intersection);
//System.out.println("/union:"+pluseq(rhs)); comment the next line to uncomment this one
pluseq(rhs);
        minuseq(indexSet(intersection));

        // Check invariant
        assert mGraph.isConsistent() : "/=␣:␣graph␣inconsistency␣detected";

        return this;
    }

    /**
     *
     * @return set The elements of rhs contained in this graph set
     */
    public GalSet indexSet(GalType rhs)
    {
        GalSet set;
        HashSet<GalType> indexed = new HashSet<GalType>();
        switch (rhs.getType())
        {
        case SET:
        case VECTOR:
                for(GalType i : (Iterable<GalType>)rhs)
                {
                        GalType j = index(i);
                        if (j != null)
                                indexed.add(j);
                }
                break;
        default:
                GalType r = index(rhs);
                if (r != null)
                        indexed.add(index(r));
        }

        return new GalSet(indexed);
    }

    /**
     * GalSet operators
     */

    /**
     * @return set A GalSet created from the objects in this set
     */
    public GalSet getGalSet()
    {
        HashSet<GalType> elementSet = new HashSet<GalType>();
        for (GalType i : mElementIndex.values())
                elementSet.add(i);

        return new GalSet(elementSet);
    }

    /**
     * @return java.util.Set<GalType> of objects in this set
     */
    public Set<GalType> getSet()
    {
        HashSet<GalType> elementSet = new HashSet<GalType>();
        for (GalType i : mElementIndex.values())
                elementSet.add(i);

        return elementSet;
    }

    protected GalType constantifySet(GalType t)
    {
        HashSet<GalType> constSet = new HashSet<GalType>();
        GalSet s = t.castSet();
        for(GalType elmt : s)
            constSet.add(elmt.constantify());

        return new GalSet(constSet);
    }

    public GalSet plus(GalType rhs)
    {
        return indexSet(((GalSet)constantifySet(getGalSet())).plus(constantifySet(rhs)));
    }

    public GalSet minus(GalType rhs)
    {
        return indexSet(((GalSet)constantifySet(getGalSet())).minus(constantifySet(rhs)));
    }
    public GalSet times(GalType rhs)
    {
//System.out.println("*:"+rhs);
//System.out.println("*:"+constantifySet(rhs));
//System.out.println("*:"+constantifySet(getGalSet()));
        return indexSet(((GalSet)constantifySet(getGalSet())).times(constantifySet(rhs)));
    }

    public GalSet div(GalType rhs)
    {
        return indexSet(((GalSet)constantifySet(getGalSet())).div(constantifySet(rhs)));
    }

    /**
     * Insert graph element t into the graph
     */
    protected void addToGraph(GalType t)
    {
        GalType toInsert = null;
        if(!mElementIndex.containsKey(t.stringRep()))
        {
//System.out.println("Adding to graph "+t);
            toInsert = mGraph.insert(t);
//System.out.println("Addeding to graphSet"+toInsert);
//System.out.println("equal? "+(toInsert.hashCode() == t.hashCode()));
//System.out.println(toInsert.hashCode()+","+ t.hashCode());
            mElementIndex.put(t.stringRep(), toInsert);
        }

        // Check invariant
        //      assert mGraph.isConsistent() : "addToGraph: graph inconsistency detected o
    }

    /**
     * Remove graph element t from the graph
     */
    protected void removeFromGraph(GalType t)
    {
        if(mElementIndex.containsKey(t.stringRep()))
           mGraph.remove(mElementIndex.remove(t.stringRep()));
    }

    /**
     * Used by graph to remove edges lost during vertex removal
     */
    public void remove(GalType t)
    {
        mElementIndex.remove(t.stringRep());
    }

    /**
     * Contains
     */
    public GalBool elementOf(GalType t)
    {
        return new GalBool(mElementIndex.containsKey(t.stringRep()));
    }

    public GalNum length()
    {
        return new GalNum(mElementIndex.size());
    }

    public String toString()
    {
        StringBuilder sb = new StringBuilder("{");
        Iterator<GalType> itr = iterator();
        if(itr.hasNext())
        {
            sb.append(itr.next());
            while(itr.hasNext())
            {
                sb.append(", ");
                sb.append(itr.next());
            }
        }
        sb.append('}');

        return sb.toString();
    }

    public Iterator<GalType> iterator()
    {
        return mElementIndex.values().iterator();
    }

}
```

Listing A.12: GalIterator.java

```
package edu.columbia.plt.gal;

import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;

/**
 * Iterator pass through
 * @author Oren B. Yeshua
 */
public class GalIterator implements Iterator<IGalRef>
{
    Iterator<GalType> mItr;

    public GalIterator(Iterator<GalType> itr)
    {
        mItr = itr;
    }

    public boolean hasNext()
    {
        return mItr.hasNext();
    }
    public IGalRef next()
    {
        return GalUtil.wrapGalType(mItr.next());
    }

    public void remove()
    {
        mItr.remove();
    }

    public List<IGalRef> getList()
    {
        ArrayList<IGalRef> lst = new ArrayList<IGalRef>();
        while(hasNext())
        {
            lst.add(next());
        }
        return lst;
    }
}
```

Listing A.13: GalNum.java

```
package edu.columbia.plt.gal;

/**
 * Gal Num type
 * @author Athar Abdul-Quader
 */
public class GalNum extends GalType
{

        protected double mNumber;

        public GalNum()
        {
                mNumber = 0;
        }

        public GalNum(double number)
        {
                mNumber = number;
        }

        public double getNumber()
        {
                return mNumber;
        }

    public int getInt()
    {
        return (int)mNumber;
    }

        public void setNumber(double number)
        {
                mNumber = number;
        }

    public T getType()
    {
        return T.NUM;
    }

    public GalType constantify()
    {
        return new GalVertex(getInt());
    }

        // assignment ops
        public GalNum assign(GalType rhs)
        {
                GalNum num = rhs.castNum();
                mNumber = num.getNumber();
                return this;
        }
        public GalNum pluseq(GalType rhs)
        {
                GalNum num = rhs.castNum();
                mNumber += num.getNumber();
                return this;
        }

        public GalNum minuseq(GalType rhs)
        {
                GalNum num = rhs.castNum();
                mNumber -= num.getNumber();
                return this;
        }

        public GalNum timeseq(GalType rhs)
        {
                GalNum num = rhs.castNum();
                mNumber *= num.getNumber();
                return this;
        }

        public GalNum diveq(GalType rhs)
        {
                GalNum num = rhs.castNum();
                mNumber /= num.getNumber();
                return this;
        }

        public GalNum modeq(GalType rhs)
        {
                GalNum num = rhs.castNum();
                mNumber %= num.getNumber();
                return this;
        }

        // arithmetic
```

```
public GalNum plus(GalType rhs)
{
        GalNum num = rhs.castNum();
        return new GalNum(mNumber + num.getNumber());
}

public GalNum minus(GalType rhs)
{
        GalNum num = rhs.castNum();
        return new GalNum(mNumber - num.getNumber());
}

public GalNum times(GalType rhs)
{
        GalNum num = rhs.castNum();
        return new GalNum(mNumber * num.getNumber());
}

public GalNum div(GalType rhs)
{
        GalNum num = rhs.castNum();
        return new GalNum(mNumber / num.getNumber());
}

public GalNum mod(GalType rhs)
{
        GalNum num = rhs.castNum();
        return new GalNum(mNumber % num.getNumber());
}

// unary operators
public GalNum length()
{
        return new GalNum(Math.abs(mNumber));
}

public GalNum minus()
{
        return new GalNum(-mNumber);
}
// boolean
public GalBool gt(GalType rhs)
{
        GalNum num = rhs.castNum();
        return new GalBool(mNumber > num.getNumber());
}

public GalBool lt(GalType rhs)
{
        GalNum num = rhs.castNum();
        return new GalBool(mNumber < num.getNumber());
}

public GalBool eq(GalType rhs)
{
        if (rhs instanceof GalNum)
```

```
                {
                        GalNum num = rhs.castNum();
                        return new GalBool(mNumber == num.getNumber());
                }
                return new GalBool(false);
        }

        public GalBool ge(GalType rhs)
        {
                GalNum num = rhs.castNum();
                return new GalBool(mNumber >= num.getNumber());
        }

        public GalBool le(GalType rhs)
        {
                GalNum num = rhs.castNum();
                return new GalBool(mNumber <= num.getNumber());
        }

        public GalBool neq(GalType rhs)
        {
                if (rhs instanceof GalNum)
                {
                        GalNum num = rhs.castNum();
                        return new GalBool(mNumber != num.getNumber());
                }
                else return new GalBool(true);
        }

        // toString()
        public String toString()
        {
            String str = null;
            int iVal = (int)mNumber;
            if(mNumber == iVal)
                str = String.valueOf(iVal);
            else
                str = String.valueOf(mNumber);
            return str;
        }

        // hashCode()
        public int hashCode()
        {
                return (new Double(mNumber)).hashCode();
        }

        // equals()
        public boolean equals(Object o)
        {
                if (o instanceof GalNum)
                {
                        return (eq( (GalNum)o).getBool());
                }
                return false;
        }
```

```
}
```

Listing A.14: GalQueue.java

```
package edu.columbia.plt.gal;
import org.jgrapht.util.FibonacciHeap;
import org.jgrapht.util.FibonacciHeapNode;
import java.util.Hashtable;
import java.lang.Iterable;

/**
 * Gal Queue type
 * @author Athar Abdul-Quader
 */
public class GalQueue extends GalType
{
        protected FibonacciHeap<GalType> mPriorityQueue;

        protected double highKey;
        protected Hashtable<GalType, FibonacciHeapNode<GalType> > mNodeTable;

        public GalQueue()
        {
                mPriorityQueue = new FibonacciHeap<GalType>();
                highKey = 0;

                mNodeTable = new Hashtable<GalType, FibonacciHeapNode<GalType> >();
        }

        public double getHighKey()
        {
                return highKey;
        }

        public void setHighKey(double high)
        {
                if (length().getNumber()==0)
                        highKey = 0;
                else
                        highKey = (highKey < high) ? high : highKey;
        }

        public FibonacciHeap<GalType> getQueue()
        {
                return mPriorityQueue;
        }

        public Hashtable<GalType, FibonacciHeapNode<GalType> > getNodeTable()
        {
                return mNodeTable;
        }

        public T getType()
        {
                return T.QUEUE;
        }
        // assignment operators
/*      public GalQueue assign(GalType rhs)
        {
                GalQueue queue = rhs.castQueue();
                mPriorityQueue.clear();
                mPriorityQueue = mPriorityQueue.union(mPriorityQueue, queue.getQueue());
                mNodeTable = queue.getNodeTable();
                highKey = queue.getHighKey();
                return this;
        }
*/
        public GalQueue pluseq(GalType rhs)
        {
                GalQueue queue = rhs.castQueue();
                mPriorityQueue = mPriorityQueue.union(mPriorityQueue, queue.getQueue());
                mNodeTable.putAll(queue.getNodeTable());
                setHighKey(queue.getHighKey());
                return this;
        }

        // union
        public GalQueue plus(GalType rhs)
        {
                GalQueue q1 = rhs.castQueue();
                GalQueue q2 = new GalQueue();
                q2.mPriorityQueue = mPriorityQueue.union(mPriorityQueue, q1.getQueue());
                Hashtable<GalType, FibonacciHeapNode<GalType> > table = new Hashtable<GalT
                table.putAll(mNodeTable);
                table.putAll(q1.getNodeTable());
                q2.setHighKey(highKey);
                q2.setHighKey(q1.getHighKey());
                return q2;
        }

        // contains
        public GalBool contains(GalType lhs)
        {
                return new GalBool(mNodeTable.containsKey(lhs));
        }

        public GalNum length()
        {
                return new GalNum(mPriorityQueue.size());
        }

}
```

Listing A.15: GalRef.java

```
package edu.columbia.plt.gal;

import java.util.Iterator;

/**
 * Reference type
 * (used for by reference semantics during assignment)
 * @author Oren B. Yeshua
 */
public class GalRef implements IGalRef
{
    protected GalType mObj;

    public GalRef(GalType t)
    {
        mObj = t;
    }

    public GalType getObj()
    {
        return mObj;
    }

    public IGalRef getMember(String inId)
    {
        return GalUtil.wrapGalType(mObj.mMembers.get(inId));
    }

    public IGalRef setMember(String inId, IGalRef inMember)
    {
        mObj.mMembers.put(inId, inMember.getObj());
        return inMember;
    }

    /* Operator Stubs */

    /*
     * Arithmetic
     */
    public IGalRef plus(IGalRef t){return GalUtil.wrapGalType(mObj.plus(t.getObj()));}

    public IGalRef minus(IGalRef t){return GalUtil.wrapGalType(mObj.minus(t.getObj()));}

    public IGalRef times(IGalRef t){return GalUtil.wrapGalType(mObj.times(t.getObj()));}

    public IGalRef div(IGalRef t){return GalUtil.wrapGalType(mObj.div(t.getObj()));}

    public IGalRef mod(IGalRef t){return GalUtil.wrapGalType(mObj.mod(t.getObj()));}


    /*
     * Comparison
     */
    public IGalRef gt(IGalRef t){return GalUtil.wrapGalType(mObj.gt(t.getObj()));}

    public IGalRef lt(IGalRef t){return GalUtil.wrapGalType(mObj.lt(t.getObj()));}

    public IGalRef ge(IGalRef t){return GalUtil.wrapGalType(mObj.ge(t.getObj()));}

    public IGalRef le(IGalRef t){return GalUtil.wrapGalType(mObj.le(t.getObj()));}

    /**
     * Equal
     */
    public IGalRef  eq(IGalRef t)
    {
        return GalUtil.wrapGalType(mObj.eq(t.getObj()));
    }

    /**
     * Not Equal
     */
    public IGalRef  neq(IGalRef t)
    {
        return GalUtil.wrapGalType(mObj.neq(t.getObj()));
    }

    /*
     * Logical
     */
    public IGalRef and(IGalRef t){return GalUtil.wrapGalType(mObj.and(t.getObj()));}
    public IGalRef or(IGalRef t){return GalUtil.wrapGalType(mObj.or(t.getObj()));}


    /*
     * Assignment
     */
    public IGalRef assign(IGalRef t)
    {
        mObj = t.getObj();
        return this;
    }

    public IGalRef pluseq(IGalRef t){return GalUtil.wrapGalType(mObj.pluseq(t.getObj()));}

    public IGalRef minuseq(IGalRef t){return GalUtil.wrapGalType(mObj.minuseq(t.getObj()))

    public IGalRef timeseq(IGalRef t){return GalUtil.wrapGalType(mObj.timeseq(t.getObj()))

    public IGalRef diveq(IGalRef t){return GalUtil.wrapGalType(mObj.diveq(t.getObj()));}

    public IGalRef modeq(IGalRef t){return GalUtil.wrapGalType(mObj.modeq(t.getObj()));}


    /*
     * Other
     */
    public IGalRef index(IGalRef t){return GalUtil.wrapGalType(mObj.index(t.getObj()));}
    public IGalRef index(IGalRef t, boolean b){return GalUtil.wrapGalType(mObj.index(t.get
    public IGalRef setIndex(IGalRef i, IGalRef t){return GalUtil.wrapGalType(mObj.setIndex

    public IGalRef access(String inId)
    {
        return GalUtil.wrapGalType(mObj.access(inId));
    }

    public IGalRef elementOf(IGalRef t){return GalUtil.wrapGalType(mObj.elementOf(t.getObj

    /*
     * Unary
     */
    public IGalRef minus(){return GalUtil.wrapGalType(mObj.minus());}
    public IGalRef not(){return GalUtil.wrapGalType(mObj.not());}
    public IGalRef length(){return GalUtil.wrapGalType(mObj.length());}

    public boolean equals(Object o)
    {
        return mObj.equals(o);
    }

    public int hashCode()
    {
        return mObj.hashCode();
    }

    public Iterator<IGalRef> iterator()
    {
        return new GalIterator(mObj.iterator());
    }
}
```

Listing A.16: GalSet.java

```
package edu.columbia.plt.gal;

import java.util.Set;
import java.util.HashSet;
import java.util.Iterator;
import java.lang.Iterable;

/**
 * Gal Set type
 * @author Oren B. Yeshua
 * @author Athar Abdul-Quader
 */
public class GalSet extends GalType implements Iterable<GalType>
{
    private Set<GalType> mSet;

    public GalSet()
    {
        mSet = new HashSet<GalType>();
    }

    /**
     * Range constructor for use by SetConst
     */
    public GalSet(int low, int hi)
    {
        this();
        for(int r=low; r <= hi; r++)
        {
            mSet.add(new GalNum(r));
        }
    }

    /**
```

```
     * Varargs constructor for use by SetConst
     *
     * This is an exceptional case where we must allow a
     * GalType knowledge of the reference layer
     */
    public GalSet(IGalRef... members)
    {
        this();
        for(IGalRef t : members)
        {
            switch (t.getObj().getType())
            {
                case STRING:
                        mSet.add(new GalString().assign(t.getObj())); break;
                case NUM:
                        mSet.add(new GalNum().assign(t.getObj())); break;
                case BOOL:
                        mSet.add(new GalBool().assign(t.getObj())); break;
                default: mSet.add(t.getObj());
            }
        }
    }

    /**
     * Varargs constructor for use by SetConst
     */
    public GalSet(GalType... members)
    {
        this();
        for(GalType t : members)
        {
            mSet.add(t);
        }
    }

    /**
     * Set constructor
     */
    public GalSet(Set<? extends GalType> s)
    {
        this();
        for(GalType t : s)
        {
            mSet.add(t);
        }
    }

    public T getType()
    {
        return T.SET;
    }

    /**
     * Assignment operator
     */
    public GalSet assign(GalType rhs)
    {
        mSet = new HashSet<GalType>(); // first clean it out
        mSet.addAll(rhs.castSet().mSet);
        return this;
    }

    /**
     * Union
     */
    public GalSet pluseq(GalType rhs)
    {
        mSet.addAll(rhs.castSet().mSet);
        return this;
    }

    /**
     * Intersection
     */
    public GalSet timeseq(GalType rhs)
    {
        mSet.retainAll(rhs.castSet().mSet);
        return this;
    }

    /**
     * Difference
     */
    public GalSet minuseq(GalType rhs)
    {
        mSet.removeAll(rhs.castSet().mSet);
        return this;
    }

    /**
     * Symmetric difference
     */
    public GalSet diveq(GalType rhs)
    {
        GalSet tmp = new GalSet().assign(rhs);
        tmp.minuseq(this);
        minuseq(rhs);
        pluseq(tmp);
        return this;
    }

    /**
     * Union
     */
    public GalSet plus(GalType rhs)
    {
        GalSet set = new GalSet();
        set.assign(this);
        set.pluseq(rhs.castSet());
        return set;
    }
    /**
     * Intersection
     */
    public GalSet times(GalType rhs)
    {
        GalSet set = new GalSet();
        set.assign(this);
        set.timeseq(rhs.castSet());
        return set;
    }

    /**
     * Difference
     */
    public GalSet minus(GalType rhs)
    {
        GalSet set = new GalSet();
        set.assign(this);
        set.minuseq(rhs.castSet());
        return set;
    }

    /**
     * Symmetric Difference
     */
    public GalSet div(GalType rhs)
    {
        GalSet tmp = new GalSet().assign(rhs);
        tmp.minuseq(this);
        GalSet set = new GalSet();
        set.assign(this);
        set.minuseq(rhs.castSet());
        set.pluseq(tmp);
        return set;
    }

    public GalNum length()
    {
        return new GalNum(mSet.size());
    }

    public GalBool contains(GalType lhs)
    {
        return new GalBool(mSet.contains(lhs));
    }

    public String toString()
    {
        StringBuilder sb = new StringBuilder("{");
        Iterator<GalType> itr = iterator();
        if(itr.hasNext())
        {
            sb.append(itr.next());
            while(itr.hasNext())
            {
                sb.append(", ");
                sb.append(itr.next());
            }
        }
        sb.append('}');
        return sb.toString();
    }

    public Iterator<GalType> iterator()
    {
        return mSet.iterator();
    }
}
```

Listing A.17: GalStdLib.java

```
package edu.columbia.plt.gal;

import edu.columbia.plt.gal.GalType.T;
import edu.columbia.plt.gal.err.GalRuntimeException;

import org.jgrapht.util.FibonacciHeapNode;

/**
 * GAL Built-In Functions
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public class GalStdLib
{
        public static void print(IGalRef i)
        {
            GalType g = i.getObj();
            System.out.print(g);
        }

        public static void println(IGalRef i)
        {
            GalType g = i.getObj();
            System.out.println(g);
        }

       /*
        * Graph functions
        */

    /**
     * Adjacency
     *
     * @param t - a vertex
     * @return all vertices adjacent to t
     */
    public static IGalRef adj(IGalRef i)
    {
        GalType t = i.getObj();
        return GalUtil.wrapGalType(t.castVertex().adj());
    }
```

```
/**
 * Retrieves an edge from the graph as if it were undirected
 *
 * uedge(G, (t,h))
 * @returns returns (t,h) if present in the graph, otherwise (h,t)
 */
public static IGalRef uedge(IGalRef Gi, IGalRef ei)
{
   GalType G = Gi.getObj();
   GalType e = ei.getObj();

   if(!G.getType().equals(T.GRAPH))
       throw new GalRuntimeException("First parameter of uedge must be a graph.");
   if(!e.getType().equals(T.EDGE))
       throw new GalRuntimeException("Second parameter of uedge must be an edge or ed

   return GalUtil.wrapGalType(G.castGraph().uedge(e));
}

   public static void show(IGalRef i, GalStepper s)
   {
       GalType g = i.getObj();
       if(g instanceof GalGraph)
           g.castGraph().show(s);
       else
       {
           System.err.println("\"show\" not yet implemented for type"
                              +g.getClass());
           //TODO: implement IGalRef.getType() method for each type
       }
   }

  /**
   * Queue functions
   */
   public static void push(IGalRef i, IGalRef j)
   {
       GalType lhs = i.getObj();
       GalType rhs = j.getObj();

           if (lhs instanceof GalQueue)
           {
                   GalQueue queue = lhs.castQueue();
                   double key = 0;
                   if (!queue.getQueue().isEmpty())
                           key = queue.getQueue().min().getKey();
                   FibonacciHeapNode<GalType> node = new FibonacciHeapNode(rhs, key-
                   queue.getQueue().insert(node, key-1);
                   queue.getNodeTable().put(rhs, node);
                   queue.setHighKey(key);
           }
   }

   public static IGalRef pop(IGalRef i)
   {
       GalType queue = i.getObj();
```

```
                GalType lhs = queue.castQueue().getQueue().removeMin().getData();
                queue.castQueue().getNodeTable().remove(lhs);
                queue.castQueue().setHighKey(0);
                return GalUtil.wrapGalType(lhs);
        }

        public static void enqueue(IGalRef i, IGalRef j)
        {
            GalType lhs = i.getObj();
            GalType rhs = j.getObj();

                GalQueue queue = lhs.castQueue();
                double key = queue.getHighKey();
                FibonacciHeapNode<GalType> node = new FibonacciHeapNode(rhs, key+1);
                queue.getQueue().insert(node, key+1);
                queue.getNodeTable().put(rhs, node);
                queue.setHighKey(key+1);
        }

        public static void insert(IGalRef i, IGalRef j, IGalRef k )
        {
            GalType lhs = i.getObj();
            GalType rhs = j.getObj();
            GalType key = k.getObj();

                GalQueue queue = lhs.castQueue();
                //GalNum key = rhs.getMember("key").castNum();
                double doubleKey = key.castNum().getNumber();
                FibonacciHeapNode<GalType> node = new FibonacciHeapNode(rhs, doubleKey);
                queue.getQueue().insert(node, doubleKey);
                queue.getNodeTable().put(rhs,node);
                queue.setHighKey(doubleKey);
        }

        public static void decrease_key(IGalRef lhsr, IGalRef rhsr, IGalRef keyr)
        {
            GalType lhs = lhsr.getObj();
            GalType rhs = rhsr.getObj();
            GalType key = keyr.getObj();

                GalQueue queue = lhs.castQueue();
                double doubleKey = key.castNum().getNumber();
                FibonacciHeapNode<GalType> node = queue.getNodeTable().get(rhs);
                queue.getQueue().decreaseKey(node, key.castNum().getNumber());
        }

        public static IGalRef dequeue(IGalRef queue)
        {
                return pop(queue);
        }

        public static IGalRef peek(IGalRef queuer)
        {
            GalType queue = queuer.getObj();
            return GalUtil.wrapGalType(queue.castQueue().getQueue().min().getData());
        }

        public static IGalRef empty(IGalRef queuer)
        {
            GalType queue = queuer.getObj();

                return GalUtil.wrapGalType(new GalBool(queue.castQueue().getQueue().isEmpt
        }
}
```

Listing A.18: GalStepper.java

```
package edu.columbia.plt.gal;


/**
 * Stepable superclass
 * @author Oren B. Yeshua
 */
public class GalStepper
{
    public static IGalRef step()
    {
        System.out.println("Please␣don't␣print␣this.");
        return null;
    }

    public void dynamicStep(){};
}
```

Listing A.19: GalString.java

```
package edu.columbia.plt.gal;

/**
 * Gal String type
 * @author Athar Abdul−Quader
 * @author Oren B. Yeshua
 */
public class GalString extends GalType
{
        protected String mString;

        public GalString()
        {
        }

        public GalString(String str)
        {
                mString = str;
        }

        public String getString()
        {
                return mString;
        }
```

```
    public T getType()
    {
        return T.STRING;
    }


        // assignment operators
        public GalString assign(GalType rhs)
        {
                GalString str = rhs.castString();
                mString = str.getString();
                return this;
        }

        public GalString pluseq(GalType rhs)
        {
                mString += rhs.toString();
                return this;
        }

        // concatenation
        public GalString plus(GalType rhs)
        {
                return new GalString(mString + rhs.toString());
        }

        // comparison
        public GalBool eq(GalType rhs)
        {
                return new GalBool(equals(rhs)); //mString.equals(rhs.getString()));
        }

        public GalBool neq(GalType rhs)
        {
                return new GalBool(!equals(rhs)); //.getString()));
        }

        // length
        public GalNum length()
        {
                return new GalNum(mString.length());
        }

        // toString()
        public String toString()
        {
                return mString;
        }

        // equals()
        public boolean equals(Object o)
        {
                return (o instanceof GalString && (mString.equals( ((GalString)o).getStrin
        }
        // hashCode()
        public int hashCode()
        {
                return mString.hashCode();
        }
}
```

Listing A.20: GalType.java

```
package edu.columbia.plt.gal;

import java.util.HashMap;
import java.util.Iterator;
import java.lang.Iterable;

/**
 * Abstract Type class
 * All Gal types
 * inherit from this class.
 *
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public abstract class GalType implements Iterable<GalType>
{
    /**
     * Type enumeration
     */
    public enum T
    {
        BOOL,
        NUM,
        STRING,
        VERTEX,
        EDGE,
        GRAPHSET,
        GRAPH,
        VECTOR,
        SET,
        QUEUE
    }

    /**
     * Member variables are stored in
     * a map keyed by id
     */
    protected HashMap<String, GalType> mMembers;


    /**
     * Protected constructor forces use of
     * subclass constructor
     */
    protected GalType()
    {
        mMembers = new HashMap<String, GalType>();
    }
    public GalType getMember(String inId)
    {
        return mMembers.get(inId);
    }

    public GalType setMember(String inId, GalType inMember)
    {
        mMembers.put(inId, inMember);
        return inMember;
    }

    public abstract T getType();

    /* Operator Stubs */

    /*
     * Arithmetic
     */
    public GalType plus(GalType t){return null;}

    public GalType minus(GalType t){return null;}

    public GalType times(GalType t){return null;}

    public GalType div(GalType t){return null;}

    public GalType mod(GalType t){return null;}


    /*
     * Comparison
     */
    public GalType gt(GalType t){return null;}

    public GalType lt(GalType t){return null;}

    public GalType ge(GalType t){return null;}

    public GalType le(GalType t){return null;}

    /**
     * Equal
     */
    public GalType  eq(GalType t)
    {
        return new GalBool(equals(t));
    }

    /**
     * Not Equal
     */
    public GalType  neq(GalType t)
    {
        return new GalBool(!equals(t));
    }
    /*
     * Logical
     */
    public GalType and(GalType t){return null;}
    public GalType or(GalType t){return null;}


    /*
     * Assignment
     */
    public GalType assign(GalType t){return null;}

    public GalType pluseq(GalType t){return null;}

    public GalType minuseq(GalType t){return null;}

    public GalType timeseq(GalType t){return null;}

    public GalType diveq(GalType t){return null;}

    public GalType modeq(GalType t){return null;}


    /*
     * Other
     */
    public GalType index(GalType t){return null;}
    public GalType index(GalType t, boolean b){return null;}
    public GalType setIndex(GalType i, GalType t){return null;}

    public GalType access(String inId)
    {
        return mMembers.get(inId);
    }

    public GalType elementOf(GalType t) { return t.contains(this); }
    public GalType contains(GalType t){return null;}

    /*
     * Unary
     */
    public GalType minus(){return null;}
    public GalType not(){return null;}
    public GalType length(){return null;}


    /* Casting methods */

    public GalBool castBool() throws ClassCastException
    {
        GalBool bool = (GalBool) this;
        return bool;
    }

    public GalNum castNum() throws ClassCastException
    {
        GalNum num = (GalNum) this;
        return num;
    }

    public GalGraph castGraph() throws ClassCastException
    {
        GalGraph graph = (GalGraph) this;
        return graph;
    }

    public GalEdge castEdge() throws ClassCastException
    {
        GalEdge edge = (GalEdge) this;
        return edge;
    }

    public GalVertex castVertex() throws ClassCastException
    {
        GalVertex vertex = (GalVertex) this;
        return vertex;
    }

    public GalSet castSet() throws ClassCastException
    {
        GalSet set = (GalSet) this;
        return set;
    }

    public GalQueue castQueue() throws ClassCastException
    {
        GalQueue queue= (GalQueue) this;
        return queue;
    }

    public GalString castString() throws ClassCastException
    {
        GalString str = (GalString) this;
        return str;
    }

    public GalArray castVector() throws ClassCastException
    {
        GalArray arr= new GalArray();
        return arr;
    }

    // toString()
    public String toString()
    {
        return "toString()␣not␣yet␣implemented␣for␣" + this.getClass();
    }

    /**
     * Unique string representation overriden by GalVertex and GalEdge
     * used to hash vertices and edges
     */
    public String stringRep()
    {
        return toString();
    }

    /**
     * Used to turn a member of a graph (vertex/edge) into a constant
     */
    public GalType constantify()
    {
        return null;
    }


    public Iterator<GalType> iterator() { return null; }
}
```

Listing A.21: GalUtil.java

```
package edu.columbia.plt.gal;
/*
 * @author Oren B. Yeshua
 */
public class GalUtil
{
    public static IGalRef wrapGalType(GalType t)
    {
        IGalRef rv = null;

        switch(t.getType())
        {
        case BOOL:
        case NUM:
        case STRING:
            rv = new GalVal(t);
            break;
        default:
            rv = new GalRef(t);
        }
        return rv;
    }
}
```

Listing A.22: GalVal.java

```
package edu.columbia.plt.gal;

import java.util.Iterator;

/**
 * Reference type
 * (used for by reference semantics during assignment)
 *
 * @author Oren B. Yeshua
 */
public class GalVal implements IGalRef
```

```
{
    protected GalType mObj;

    public GalVal(GalType t)
    {
        mObj = t;
    }

    public GalType getObj()
    {
        return mObj;
    }

    public IGalRef getMember(String inId)
    {
        return GalUtil.wrapGalType(mObj.mMembers.get(inId));
    }

    public IGalRef setMember(String inId, IGalRef inMember)
    {
        mObj.mMembers.put(inId, inMember.getObj());
        return inMember;
    }

    /* Operator Stubs */

    /*
     * Arithmetic
     */
    public IGalRef plus(IGalRef t){return GalUtil.wrapGalType(mObj.plus(t.getObj()));}

    public IGalRef minus(IGalRef t){return GalUtil.wrapGalType(mObj.minus(t.getObj()));}

    public IGalRef times(IGalRef t){return GalUtil.wrapGalType(mObj.times(t.getObj()));}

    public IGalRef div(IGalRef t){return GalUtil.wrapGalType(mObj.div(t.getObj()));}

    public IGalRef mod(IGalRef t){return GalUtil.wrapGalType(mObj.mod(t.getObj()));}


    /*
     * Comparison
     */
    public IGalRef gt(IGalRef t){return GalUtil.wrapGalType(mObj.gt(t.getObj()));}

    public IGalRef lt(IGalRef t){return GalUtil.wrapGalType(mObj.lt(t.getObj()));}

    public IGalRef ge(IGalRef t){return GalUtil.wrapGalType(mObj.ge(t.getObj()));}

    public IGalRef le(IGalRef t){return GalUtil.wrapGalType(mObj.le(t.getObj()));}

    /**
     * Equal
     */
    public IGalRef  eq(IGalRef t)
    {
        return GalUtil.wrapGalType(mObj.eq(t.getObj()));
    }

    /**
     * Not Equal
     */
    public IGalRef  neq(IGalRef t)
    {
        return GalUtil.wrapGalType(mObj.neq(t.getObj()));
    }

    /*
     * Logical
     */
    public IGalRef and(IGalRef t){return GalUtil.wrapGalType(mObj.and(t.getObj()));}
    public IGalRef or(IGalRef t){return GalUtil.wrapGalType(mObj.or(t.getObj()));}


    /*
     * Assignment
     */
    public IGalRef assign(IGalRef t)
    {
        mObj.assign(t.getObj());
        return this;
    }

    public IGalRef pluseq(IGalRef t){return GalUtil.wrapGalType(mObj.pluseq(t.getObj()));}

    public IGalRef minuseq(IGalRef t){return GalUtil.wrapGalType(mObj.minuseq(t.getObj()))

    public IGalRef timeseq(IGalRef t){return GalUtil.wrapGalType(mObj.timeseq(t.getObj()))

    public IGalRef diveq(IGalRef t){return GalUtil.wrapGalType(mObj.diveq(t.getObj()));}

    public IGalRef modeq(IGalRef t){return GalUtil.wrapGalType(mObj.modeq(t.getObj()));}


    /*
     * Other
     */
    public IGalRef index(IGalRef t){return GalUtil.wrapGalType(mObj.index(t.getObj()));}
    public IGalRef index(IGalRef t, boolean b){return GalUtil.wrapGalType(mObj.index(t.get
    public IGalRef setIndex(IGalRef i, IGalRef t){return GalUtil.wrapGalType(mObj.setIndex

    public IGalRef access(String inId)
    {
        return GalUtil.wrapGalType(mObj.access(inId));
    }

    public IGalRef elementOf(IGalRef t){return GalUtil.wrapGalType(mObj.elementOf(t.getObj

    /*
     * Unary
     */
    public IGalRef minus(){return GalUtil.wrapGalType(mObj.minus());}
    public IGalRef not(){return GalUtil.wrapGalType(mObj.not());}
    public IGalRef length(){return GalUtil.wrapGalType(mObj.length());}

    public boolean equals(Object o)
    {
        return mObj.equals(o);
    }

    public int hashCode()
    {
        return mObj.hashCode();
    }

    public Iterator<IGalRef> iterator()
    {
        return null;
    }
}
```

Listing A.23: GalVector.java

```
package edu.columbia.plt.gal;

import java.util.ArrayList;
import java.util.Iterator;
import java.lang.Iterable;

/**
 * Gal Vector type
 * @author Athar Abdul-Quader
 */
public class GalVector extends GalType implements Iterable<GalType>
{
        protected ArrayList<GalType> mVector;
        protected int length;

        public GalVector()
        {
                mVector = new ArrayList<GalType>();
                length = 0;
        }

        public GalVector assign(GalType rhs)
        {

                return new GalVector();
        }

        public GalType index(GalType rhs)
        {
                int i = rhs.castNum().getInt();

                return mVector.get(i);
        }

        public GalType setIndex(GalType i, GalType t)
        {
                int index = i.castNum().getInt();
                if (index + 1 > length)
                {
                        length=(index>=length) ? index+1 : length;
                        for (int j=mVector.toArray().length; j < length; j++)
                                mVector.add(null);
                }
                mVector.set(index, t);
                return t;
        }

        public GalType index(GalType rhs, boolean flag)
        {
                int i = rhs.castNum().getInt();
                if ((i >= mVector.toArray().length || mVector.get(i) == null) && flag==tru
                {
                        GalType vect = new GalVector();
                        length = (i>=length) ? i+1 : length;
                        for (int j=mVector.toArray().length; j < length; j++)
                                mVector.add(null);
                        mVector.set(i, vect);
                        return vect;
                }
                else // index returns non-null value
                {
                        return index(rhs);
                }
        }

        public GalNum length()
        {
                return new GalNum(length);
        }

    public String toString()
    {
        StringBuilder sb = new StringBuilder('{');
        Iterator<GalType> itr = iterator();
        if(itr.hasNext())
        {
            sb.append(itr.next());
            while(itr.hasNext())
            {
                sb.append(", ");
                sb.append(itr.next());
            }
        }
        sb.append('}');
        return sb.toString();
    }


        public Iterator<GalType> iterator()
        {
                return mVector.iterator();
        }
        public T getType()
        {
                return T.VECTOR;
        }

}
```

Listing A.24: GalVertex.java

```
package edu.columbia.plt.gal;


/**
 * Gal Vertex type
 *
 * @author Oren B. Yeshua
 * @author Athar Abdul-Quader
 */
public class GalVertex extends GalType
{
    protected boolean mIsWild;
    protected int mIndex;
    protected String mStringRep;
    protected GalGraph mGraph;

    public GalVertex() { }

    /**
     *  For use in code generation
     */
    public GalVertex(int inIndex)
    {
        mIndex = inIndex;
        mStringRep = Integer.toString(mIndex);
    }

    /**
     * Creates a new wildcard vertex
     *
     * A wildcard cannot be added to a graph.
     * It is used as syntactic sugar to query
     * the edge and vertex sets of a graph
     *
     * G.V(1,?) returns the set of vertices connected to 1 via an outgoing edge
     * G.E(1,?) returns the set of outgoing edges connected to 1
     */
    public GalVertex(String inWildcard)
    {
        mStringRep = inWildcard;
        mIsWild = true;
    }

    public GalVertex(int inIndex, GalGraph g)
    {
        this(inIndex);
        mGraph = g;
    }

    public T getType()
    {
        return T.VERTEX;
    }

    public int getIndex()
    {
        return mIndex;
    }

    public boolean isWild()
    {
        return mIsWild;
    }

    public GalGraph getGraph()
    {
        return mGraph;
    }

    public void setGraph(GalGraph g)
    {
        mGraph = g;
    }

    /**
     * @return Set of all vertices adjacent to this one
     */
    public GalSet adj()
    {
        return mGraph.allAdjacentVerticesOf(this);
    }

    /**
     * @return degree of vertex
     */
    public GalType length()
    {
        return mGraph.degreeOf(this);
    }

    public String toString()
    {
        return mStringRep+(mGraph == null ? '*' : "");
    }

    public String stringRep()
    {
        return mStringRep;
    }

    public GalType constantify()
    {
        return new GalVertex(getIndex());
    }

    public boolean equals(Object o)
    {
//System.out.println("Vertex.equals: "+this+" == "+o);

        boolean rv = false;
        if(o instanceof GalVertex)
        {
            GalVertex v = (GalVertex)o;
            if(mIndex == v.mIndex)
            {
                // Vertex const - by value semantics
                if(mGraph == null && v.mGraph == null)
                {
                    rv = true;
                }
                else if(mGraph == v.mGraph)
                {// Vertex - by reference semantics
                    rv = true;
                }
            }
        }
        return rv;
    }

    public int hashCode()
    {
        int code = super.hashCode();
        if(mGraph == null)
        {
            code = mIndex;
        }
        return code;
    }
}
```

Listing A.25: GalVertexSet.java

```
package edu.columbia.plt.gal;

import edu.columbia.plt.gal.err.GalRuntimeException;

import java.util.Hashtable;
import java.util.HashSet;
import java.util.Iterator;
import java.lang.Iterable;

/**
 * Gal Vertex Set (Set internal to graph)
 *
 * Performs indexing specific to a vertex set
 *
 * @author Oren B. Yeshua
 */
public class GalVertexSet extends GalGraphSet
{
    public GalVertexSet(GalGraph g){super(g);}

    public GalType index(GalType i)
    {
        GalType rv = null;
        switch(i.getType())
        {
        case EDGE:
            GalEdge e = i.castEdge();
            if(e.getTail().isWild())
            {
                rv = mGraph.incomingAdjacentVerticesOf(mGraph.index(e.getHead()).castVerte
            }
            else if(e.getHead().isWild())
            {
                rv = mGraph.outgoingAdjacentVerticesOf(mGraph.index(e.getTail()).castVerte
            }
            else
                throw new GalRuntimeException("Vertex␣set␣indexed␣by␣an␣edge␣constant␣with

            break;
        default:
            rv = mElementIndex.get(i.stringRep());
        }
        return rv;
    }

//      public GalSet getGalSet()
//      {
//       HashSet<GalType> elementSet = new HashSet<GalType>();
//       for (GalType i : mElementIndex.values())
//       {
//               GalVertex v = new GalVertex(i.castVertex().getIndex());
//               elementSet.add(v);
//       }

//       return new GalSet(elementSet);
//      }
}
```

Listing A.26: GalVisFrame.java

```
package edu.columbia.plt.gal;

import java.util.Map;
import java.util.HashMap;
import java.util.Random;
import java.awt.Rectangle;
import java.awt.geom.Rectangle2D;
import java.awt.Point;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JComponent;
import org.jgraph.JGraph;
import org.jgraph.graph.*;
import org.jgrapht.*;
import org.jgrapht.graph.*;
import org.jgrapht.ext.JGraphModelAdapter;

/**
 * Graph Visualizaton Frame
 * @author Oren B. Yeshua
 * @author Albert J. Winters
 */
public class GalVisFrame extends JFrame implements ActionListener
{
    private static final Dimension DEFAULT_SIZE = new Dimension(640,480);
    private static final Color     DEFAULT_BG_COLOR = Color.decode("#FAFBFF");
    private static int stepCount = 0;

    private JGraphModelAdapter<GalVertex,GalEdge> mAdapter;
    private JGraph mJGraph;
    private ListenableGraph<GalVertex,GalEdge> mJGraphT;
    private GalStepper mStepper;

    public GalVisFrame(ListenableGraph inGraph, GalStepper inStepable)
    {
        mAdapter = new JGraphModelAdapter<GalVertex, GalEdge>(inGraph);
        mJGraph = new JGraph(mAdapter);
        mJGraphT = inGraph;
        mStepper = inStepable;

        // Layout
        randomLayout();

        // Step
        JButton step_JB = new JButton("STEP");
        step_JB.addActionListener(this);
        add(step_JB, BorderLayout.SOUTH);

        mJGraph.setBackground(DEFAULT_BG_COLOR);
        mJGraph.setPreferredSize(DEFAULT_SIZE);
        add(mJGraph, BorderLayout.CENTER);
        setSize(DEFAULT_SIZE);
        pack();
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setVisible(true);
    }

    public void recolor(GalVertex v)
    {

        DefaultGraphCell cell = mAdapter.getVertexCell( v );
        Map              attr = cell.getAttributes( );
        Rectangle2D        b  = GraphConstants.getBounds( attr );

        //Set the color of the vertex
        String stringColor = null;
        GalType temp =  v.getMember("color");
        if(temp != null) stringColor = temp.toString();

        Color newColor=null;
        if(stringColor!=null && !(stringColor.equals("null")))
        {
            if(stringColor.toLowerCase().equals("blue"))  newColor = Color.blue;
            else if(stringColor.toLowerCase().equals("red"))  newColor = Color.red;
            else if(stringColor.toLowerCase().equals("black"))  newColor = Color.black;
            else if(stringColor.toLowerCase().equals("green"))  newColor = Color.green;
            else if(stringColor.toLowerCase().equals("orange"))  newColor = Color.orange;
            else if(stringColor.toLowerCase().equals("white"))  newColor = Color.white;
            else if(stringColor.toLowerCase().equals("yellow"))  newColor = Color.yellow;
            else newColor = Color.gray;
        }
        if(newColor != null)
            GraphConstants.setBackground(attr,newColor);
     }

   protected void setVertexPos(GalVertex v, Point p)
   {
        DefaultGraphCell  cell = mAdapter.getVertexCell( v );
        Map               attr = cell.getAttributes(  );
        Rectangle2D         b    = GraphConstants.getBounds( attr );

        GraphConstants.setBounds(attr,
                                 new Rectangle(p.x,p.y,
                                                   (int)b.getWidth(),
                                                   (int)b.getHeight()));
        Map cellAttr = new HashMap();
        cellAttr.put( cell, attr );
        mAdapter.edit( cellAttr,null, null, null );
        recolor(v);
    }

    protected void randomLayout()
    {
        Random rand = new Random(123454321);
        for(GalVertex v : mJGraphT.vertexSet())
        {
            int x = rand.nextInt(DEFAULT_SIZE.width - 200);
            int y = rand.nextInt(DEFAULT_SIZE.height - 100);

            setVertexPos(v,new Point(x,y));
        }
    }

    public void recolorGraph()
    {
        for(GalVertex v : mJGraphT.vertexSet())
            recolor(v);

        randomLayout();
    }

    public void actionPerformed(ActionEvent e)
    {
        //System.out.println("Step: "+(stepCount++));

        mStepper.dynamicStep();

        recolorGraph();
    }
}
```

Listing A.27: GraphType.java

```
package edu.columbia.plt.gal;

public class GraphType extends Type
{
        public void GraphType()
        {
                currentType = new GraphType();
                subType = null;
        }
}
```

Listing A.28: IGalRef.java

```
package edu.columbia.plt.gal;
import java.util.Iterator;
import java.lang.Iterable;

/**
 *
 * Interface for all types
 * GalNum and GalRef should implement this
 *
 * @author Oren B. Yeshua
 */
public interface IGalRef extends Iterable<IGalRef>
{
    public GalType getObj();

    public IGalRef getMember(String inId);
    public IGalRef setMember(String inId, IGalRef inMember);

    /* Operator Stubs */

    /*
     * Arithmetic
     */
    public IGalRef plus(IGalRef t);
    public IGalRef minus(IGalRef t);
    public IGalRef times(IGalRef t);
    public IGalRef div(IGalRef t);
    public IGalRef mod(IGalRef t);


    /*
     * Comparison
     */
    public IGalRef gt(IGalRef t);
    public IGalRef ge(IGalRef t);
    public IGalRef lt(IGalRef t);
    public IGalRef le(IGalRef t);

    /**
     * Equal
     */
    public IGalRef eq(IGalRef t);

    /**
     * Not Equal
     */
    public IGalRef  neq(IGalRef t);

    /*
     * Logical
     */
    public IGalRef and(IGalRef t);
    public IGalRef or(IGalRef t);


    /*
     * Assignment
     */
    public IGalRef assign(IGalRef t);
    public IGalRef pluseq(IGalRef t);
    public IGalRef minuseq(IGalRef t);
    public IGalRef timeseq(IGalRef t);
    public IGalRef diveq(IGalRef t);
    public IGalRef modeq(IGalRef t);


    /*
     * Other
     */
    public IGalRef index(IGalRef t);
    public IGalRef index(IGalRef t, boolean b);
    public IGalRef setIndex(IGalRef i, IGalRef t);

    public IGalRef access(String inId);
    public IGalRef elementOf(IGalRef t);

    /*
     * Unary
     */
    public IGalRef minus();
    public IGalRef not();
    public IGalRef length();

    public Iterator<IGalRef> iterator();
}
```

Listing A.29: NumType.java

```
package edu.columbia.plt.gal;
```

```
public class NumType extends Type
{
        public void NumType()
        {
                currentType = new NumType();
                subType = null;
        }
}
```

Listing A.30: QueueType.java

```
package edu.columbia.plt.gal;

public class QueueType extends Type
{
        public void QueueType(Type sub)
        {
                currentType = new QueueType();
                subType = sub;
        }
}
```

Listing A.31: SetType.java

```
package edu.columbia.plt.gal;

public class SetType extends Type
{
        public void SetType(Type sub)
        {
                currentType = new SetType();
                subType = sub;
        }
}
```

Listing A.32: StringType.java

```
package edu.columbia.plt.gal;

public class StringType extends Type
{
        public void StringType()
        {
                currentType = new StringType();
                subType = null;
        }
}
```

Listing A.33: SymbolTable.java

```
package edu.columbia.plt.gal;

import java.util.Hashtable;

/**
 * Symbol Table
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public class SymbolTable<T>
{
    private Hashtable<String, T> table;
    protected SymbolTable<T> parent;

    public SymbolTable(SymbolTable<T> parent)
    {
        this.parent = parent;
        table = new Hashtable<String, T>();
    }

    public void put(String id, T record)
    {
        table.put(id, record);
    }

    public T get(String id)
    {
        T record = null;
        for(SymbolTable<T> st = this;
            st != null;
            st = st.parent)
            {
                record = st.table.get(id);
                if(record != null)
                    break;
            }
        return record;
    }

    public boolean existsInCurrent(String id)
    {
        T record = null;
        record = table.get(id);
        if (record != null) return true;
        else return false;
    }
}
```

Listing A.34: Type.java

```
package edu.columbia.plt.gal;

public class Type
{
        protected Type currentType;
        protected Type subType;

        public boolean equals(Object other)
        {
                if (other.getClass().equals(this.getClass()))
                {
                        Type otherType = (Type) other;
                        if (currentType.getClass() != otherType.currentType.getClass())
                                return false;
                        if (subType != null)
                        {
                                if (otherType.subType == null)
                                        return false;
                                if (subType.getClass() != otherType.subType.getClass())
                                        return false;
                        }
                }
                else return false;

                return true;
        }
}
```

Listing A.35: VertexType.java

```
package edu.columbia.plt.gal;

public class VertexType extends Type
{
        public void VertexType()
        {
                currentType = new VertexType();
                subType = null;
        }
}
```

A.4.2 package edu.columbia.plt.gal.ast

Listing A.36: AccessOp.java

```
package edu.columbia.plt.gal.ast;

/*
 * Access (.) Operator
 * @author Oren B. Yeshua
 * @author Athar Abdul-Quader
 */
public class AccessOp extends BinaryOp
{
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(");
        code.append(getLValue().gen());
        code.append(").");
        code.append(oSymbolToName.get(getText()));
        code.append("(\"");
        code.append(getRValue().gen());
        code.append("\")");
        return code;
    }
}
```

Listing A.37: Args.java

```
package edu.columbia.plt.gal.ast;
```

```java
import java.util.ArrayList;
import java.util.List;
import antlr.collections.AST;
/*
 * Parameter list
 * @author Athar Abdul-QUader
 */
public class Args extends Node
{
    public List<Node> getParmList()
    {
        ArrayList<Node> params = new ArrayList<Node>();
        for (AST child = getFirstChild(); child != null; child = child.getNextSibling())
        {
                params.add((Node)(child));
        }
        return params;
    }
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        List<Node> params = getParmList();
        if (params.isEmpty())
                return code;
        code.append(params.get(0).gen());
        for (int i=1; i<params.size(); i++)
        {
                code.append(",␣");
                code.append(params.get(i).gen());
        }
        return code;
    }
}
```

Listing A.38: Assign.java

```java
package edu.columbia.plt.gal.ast;

/**
 * Assign (<-)
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public class Assign extends AssignOp
{
    public StringBuilder gen()
    {
        StringBuilder sb = new StringBuilder();

        Node lhs = getLValue();
        Node rhs = getRValue();
        //      sb.append("(");
        if (lhs instanceof AccessOp) // access
        {
                AccessOp aOp = (AccessOp)lhs;
                sb.append("(");
                sb.append(aOp.getLValue().gen());
```

```
                    sb.append(")");
                    sb.append(".setMember(\"");
                    sb.append(aOp.getRValue().gen());
                    sb.append("\",␣(");
                    sb.append(rhs.gen());
                    sb.append("))");
        }
        else if (lhs instanceof Index) // index
        {
                    sb.append("(");
                    Index ind = (Index)lhs;
                    Index tmp = ind;
                    if (tmp.getLValue() instanceof Index)
                    {
                            while (tmp.getLValue() instanceof Index)
                            {
                                    tmp = (Index) tmp.getLValue();
                                    sb.append("(");
                                    sb.append(tmp.getLValue().gen());
                                    sb.append(").index(");
                                    sb.append(tmp.getRValue().gen());
                                    sb.append(",true");
                                    sb.append(")");
                            }
                    }
                    else
                            sb.append(ind.getLValue().gen());
                    sb.append(")");
                    sb.append(".setIndex(");
                    sb.append(ind.getRValue().gen());
                    sb.append(",␣(");
                    sb.append(rhs.gen());
                    sb.append("))");
        }
        else
        {
            sb.append(lhs.gen());
            sb.append(".");
            sb.append(oSymbolToName.get(getText()));
            sb.append("(");
            sb.append(rhs.gen());
            sb.append(")");
        }
        //          sb.append(")");

        return sb;
    }
}
```

Listing A.39: AssignOp.java

```
package edu.columbia.plt.gal.ast;

/*
 * Assignment Operator
 */
public abstract class AssignOp extends BinaryOp
```

```
{
}
```

Listing A.40: BinaryOp.java

```
package edu.columbia.plt.gal.ast;

import java.util.Hashtable;

/*
 * Binary Operator
 * @author Oren B. Yeshua
 */
public class BinaryOp extends Node
{
    protected static Hashtable<String, String> oSymbolToName;

    static
    {
        oSymbolToName = new Hashtable<String, String>(17);
        oSymbolToName.put("+","plus");
        oSymbolToName.put("-","minus");
        oSymbolToName.put("*","times");
        oSymbolToName.put("/","div");
        oSymbolToName.put("%","mod");

        oSymbolToName.put(">","gt");
        oSymbolToName.put("<","lt");
        oSymbolToName.put(">=","ge");
        oSymbolToName.put("<=","le");
        oSymbolToName.put("=","eq");
        oSymbolToName.put("!=","neq");

        oSymbolToName.put("<-","assign");
        oSymbolToName.put("+=","pluseq");
        oSymbolToName.put("-=","minuseq");
        oSymbolToName.put("*=","timeseq");
        oSymbolToName.put("/=","diveq");
        oSymbolToName.put("%=","modeq");
        oSymbolToName.put("in","elementOf");
        oSymbolToName.put("[","index");
        oSymbolToName.put(".","access");

        oSymbolToName.put("and","and");
        oSymbolToName.put("or","or");
    }

    public Node getLValue()
    {
        return getChildNode();
    }

    public Node getRValue()
    {
        return getChildNode().getSiblingNode();
    }
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(");
        code.append(getLValue().gen());
        code.append(").");
        code.append(oSymbolToName.get(getText()));
        code.append("(");
        code.append(getRValue().gen());
        code.append(")");
        return code;
    }
}
```

Listing A.41: Body.java

```
package edu.columbia.plt.gal.ast;

import java.util.List;
import java.util.ArrayList;
import edu.columbia.plt.gal.SymbolTable;
import antlr.collections.AST;

/**
 * Body (block)
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public class Body extends Node
{
    public List<Node> getStmts()
    {
        List<Node> stmts = new ArrayList<Node>();

        for(AST child = getFirstChild();
            child != null;
            child = child.getNextSibling())
            {
                stmts.add((Node)child);
            }

        return stmts;
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        for(Node stmt : getStmts())
        {
            code.append(stmt.gen());

            // Don't append semicolon to statements that end in a block
            if (! (stmt instanceof If
                   || stmt instanceof Else
                   || stmt instanceof While
                   || stmt instanceof Foreach))
                code.append(";\n");
        }
        return code;
    }

    public void setEnv(SymbolTable<Node> parent, boolean flag)
    {
        if (flag)
        {
                env = new SymbolTable<Node>(parent);
        }
        else env = parent;

        for (Node stmt : getStmts())
        {
                stmt.setEnv(env, flag);
        }
    }
}
```

Listing A.42: BoolConst.java

```
package edu.columbia.plt.gal.ast;

/**
 * String Constant
 * @author Athar Abdul-Quader
 */
public class BoolConst extends Node
{
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("new␣GalVal(new␣GalBool(");
        code.append(getText());
        code.append("))");
        return code;
    }
}
```

Listing A.43: Diveq.java

```
package edu.columbia.plt.gal.ast;

/*
 * Divide-equals Operator
 * @author Jay Winters
 */
public class Diveq extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".diveq(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }
}
```

Listing A.44: Div.java

```
package edu.columbia.plt.gal.ast;

/*
 * Divide Operator
 * @author Jay Winters
 */
public class Div extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".div(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.45: EdgeConst.java

```
package edu.columbia.plt.gal.ast;

/**
 * Edge Constant
 * @author Oren B. Yeshua
 * @author Athar Abdul-Quader
 */
public class EdgeConst extends Node
{
    public Node getTail()
    {
        return (Node)getChildNode();
    }

    public Node getHead()
    {
        return (Node)getChildNode().getSiblingNode();
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("new GalRef(new GalEdge(");
        code.append(getTail().gen());
        code.append(',');
        code.append(getHead().gen());
        code.append("))");
        return code;
    }
}
```

Listing A.46: Else.java

```
package edu.columbia.plt.gal.ast;

/**
 * Else Statement
 * @author Athar Abdul-Quader
 */
public class Else extends Node
{
    public Body getBody()
    {
        return (Body)getChildNode();
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        code.append("else␣");
        code.append("{\n");
        code.append(getBody().gen());
        code.append("}\n");
        return code;
    }
}
```

Listing A.47: Eq.java

```
package edu.columbia.plt.gal.ast;

/*
 * Equals Operator
 * @author Jay Winters
 */
public class Eq extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".eq(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.48: Expr.java

```
package edu.columbia.plt.gal.ast;


/*
 * Expression
 */
public abstract class Expr extends Node
```

```
{
}
```

Listing A.49: FCall.java

```
package edu.columbia.plt.gal.ast;

import edu.columbia.plt.gal.SymbolTable;

/**
 * Function call
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public class FCall extends Node
{
    public String getName()
    {
        return getFirstChild().getText();
    }

    public Node getParms()
    {
        return (Node)(getFirstChild().getNextSibling());
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        code.append(getName());
        code.append("(");
        code.append(getParms().gen());
        if(getName().equals("show"))
           code.append(",␣((GalStepper)Class.forName(getClassName()).newInstance())");
        code.append(")");
        return code;
    }

    public void setEnv(SymbolTable<Node> parent, boolean flag)
    {
        super.setEnv(parent, flag);
        Integer n = Prog.oFuncTable.get(getName()+":"+ ((Args)getParms()).getParmList().si
        if (n == null)
        {
                System.err.println("Function␣\"" + getName() + "\"␣not␣declared␣in␣this␣sc
        }
    }

}
```

Listing A.50: FDecl.java

```
package edu.columbia.plt.gal.ast;

import edu.columbia.plt.gal.SymbolTable;

/**
```

```
 * Function declaration
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public class FDecl extends Node
{
    public Id getName()
    {
        return (Id)getChildNode();
    }

    public Parms getParms()
    {
        return (Parms)getName().getSiblingNode();
    }

    public Body getBody()
    {
        return (Body)getParms().getSiblingNode();
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        boolean isMain = getName().getText().equalsIgnoreCase("MAIN");
        String fName= getName().getText();

        if (isMain)
        {
                code.append("void␣main(String[]␣args)\n");
        }
        else
        {
                code.append("IGalRef␣");
                code.append(getName().gen());
                code.append("(");
                code.append(getParms().gen());
                code.append(")\n");
        }
        code.append("{\n");

        code.append("try{\n"); //catch exceptions for show

        code.append(getBody().gen());
        if(!(hasReturn() ||  isMain))
        {
            code.append("return␣null;\n");
        }

        code.append("}catch(Exception␣e){e.printStackTrace();}"); // catch exceptions for

        if(!(hasReturn() ||  isMain))
        {
            code.append("return␣null;\n");
        }
        else
```

```
            code.append("return;\n");


        code.append("}");
        return code;
    }
    public void setEnv(SymbolTable<Node> parent, boolean flag)
    {
        env = parent;
        SymbolTable<Node> current = new SymbolTable<Node>(parent);

        getParms().setEnv(current, flag);
        getBody().setEnv(current, false);
    }
}
```

Listing A.51: Foreach.java

```
package edu.columbia.plt.gal.ast;

import edu.columbia.plt.gal.SymbolTable;

/**
 * Foreach statement
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public class Foreach extends Node
{
    public Id getId()
    {
        return (Id)getChildNode();
    }

    public Node getExpr()
    {
        return getChildNode().getSiblingNode();
    }

    public Body getBody()
    {
        return (Body)getExpr().getSiblingNode();
    }

    /**
     * Create new scope and pass it to the body (create new scope flag = false)
     * (Expression, Id get parent scope)
     * TODO: insert variable into scope
     */
    public void setEnv(SymbolTable<Node> parent, boolean flag)
    {
        env = null;
        SymbolTable<Node> current = new SymbolTable<Node>(parent);

        getId().setEnv(parent, flag);
        getExpr().setEnv(parent, flag);
        current.put(getId().getText(), getId());
        getBody().setEnv(current, false);
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        code.append("for␣(IGalRef␣");
        code.append(getId().gen());
        code.append("␣:␣␣((GalIterator)(((Iterable<IGalRef>)");
        code.append(getExpr().gen());
        code.append("␣).iterator())).getList()␣)\n");
        code.append("{\n");
        code.append(getBody().gen());
        code.append("}\n");
        return code;
    }
}
```

Listing A.52: Gteq.java

```
package edu.columbia.plt.gal.ast;

/*
 * Greater than-equal Operator
 * @author Jay Winters
 */
public class Gteq extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".gteq(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.53: Gt.java

```
package edu.columbia.plt.gal.ast;

/*
 * Greater than Operator
 * @author Jay Winters
 */
public class Gt extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".gt(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.54: Id.java

```
package edu.columbia.plt.gal.ast;

/*
 * Identifier
 */
public class Id extends Node
{
    public StringBuilder gen()
    {
        return new StringBuilder(getText());
    }
}
```

Listing A.55: If.java

```
package edu.columbia.plt.gal.ast;

/**
 * If statement
 * @author Athar Abdul-Quader
 */
public class If extends Node
{
    public Node getCondition()
    {
        return getChildNode();
    }

    public Body getBody()
    {
        return (Body)getCondition().getSiblingNode();
    }

    public Else getElse()
    {
        return (Else)getBody().getSiblingNode();
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        code.append("if␣(␣(");
        code.append(getCondition().gen());
        code.append(").getObj().castBool().getBool())\n");
        code.append("{\n");
        code.append(getBody().gen());
        code.append("}\n");
        if (getElse() != null)
        {
                code.append(getElse().gen());
        }
        return code;
    }
}
```

Listing A.56: Index.java

```
package edu.columbia.plt.gal.ast;

/**
 * Index ([]) Operator
 * @author Athar Abdul-Quader
 */
public class Index extends BinaryOp
{
    public StringBuilder gen()
    {
        Node lhs = getLValue();
        StringBuilder code = new StringBuilder("(");
        if (lhs instanceof Index)
        {
                Index ind = (Index) lhs;
                code.append("(");
                code.append(ind.getLValue().gen());
                code.append(").");
                code.append(oSymbolToName.get(getText()));
                code.append("(");
                code.append(ind.getRValue().gen());
                code.append(",true");
        }
        code.append(getLValue().gen());
        code.append(").");
        code.append(oSymbolToName.get(getText()));
        code.append("(");
        code.append(getRValue().gen());
        code.append(")");
        return code;
    }
}
```

Listing A.57: Lteq.java

```
package edu.columbia.plt.gal.ast;

/*
 * Less than-equal Operator
 * @author Jay Winters
 */
public class Lteq extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".lteq(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.58: Lt.java

```
package edu.columbia.plt.gal.ast;

/*
 * Less than Operator
 * @author Jay Winters
 */
public class Lt extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".lt(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.59: Minuseq.java

```
package edu.columbia.plt.gal.ast;

/*
 * Minus-equals Operator
 * @author Jay Winters
*/
public class Minuseq extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".minuseq(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.60: Minus.java

```
package edu.columbia.plt.gal.ast;

/*
 * Minus Operator
 * @author Jay Winters
 */
public class Minus extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".minus(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.61: Modeq.java

```
package edu.columbia.plt.gal.ast;

/*
 * Mod-equals Operator
 * @author Jay Winters
 */
public class Modeq extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".modeq(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.62: Mod.java

```
package edu.columbia.plt.gal.ast;

/*
 * Modular Operator
 * @author Jay Winters
 */
public class Mod extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".mod(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.63: Neq.java

```
package edu.columbia.plt.gal.ast;

/*
 * Not equal Operator
 * @author Jay Winters
 */
public class Neq extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".neq(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.64: Node.java

```
package edu.columbia.plt.gal.ast;

import antlr.collections.AST;
import antlr.Token;
import edu.columbia.plt.gal.SymbolTable;

/**
 * Heterogeneous AST base class
 * @author Oren B. Yeshua
 * @author Athar Abdul-Quader
 */
public class Node extends antlr.CommonAST
{
    protected int lineNum;
    protected int colNum;

    public void initialize(Token tok) {
        super.initialize(tok);
        setLocation(tok.getLine(), tok.getColumn());
    }
    protected void setLocation(int line, int col)
    {
        lineNum = line;
        colNum = col;
    }

    public int getLine()
    {
        if (lineNum == 0 && getFirstChild() != null)
                return getFirstChild().getLine();
        else return lineNum;
    }
    public int getColumn() { return colNum; }

    public String getLineStr()
    {
        return "[Line:␣"+getLine()+",␣"+"Col:␣"+getColumn()+']';
    }

    protected SymbolTable<Node> env;

    public void setEnv(SymbolTable<Node> parent, boolean flag)
    {
        env = parent;
        for(AST c = getFirstChild(); c != null; c = c.getNextSibling())
        {
                if (c instanceof Node)
                        ((Node)c).setEnv(env, flag);
        }
    }

    /*
     * Generate Java code
     */
    public StringBuilder gen()
    {
        return new StringBuilder(getText());
    }

    /*
     * Convenience accessor
     */
    public Node getChildNode()
    {
        return (Node)down;
    }

    /*
     * Convenience accessor
     */
    public Node getSiblingNode()
    {
        return (Node)right;
    }

    /**
```

```
     * Check for return
     * TODO: only implement in relevant AST nodes
     */
    public boolean hasReturn()
    {
        boolean rv = false;
        for(AST c = getFirstChild(); c != null; c = c.getNextSibling())
        {
            if(c instanceof Node)
            {
                rv = rv || ((Node)c).hasReturn();
            }
        }
        return rv;
    }
}
```

Listing A.65: Not.java

```
package edu.columbia.plt.gal.ast;

/*
 * Not Operator
 * @author Jay Winters
 */
public class Not extends UnaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getValue().gen());
        code.append(".not());");
        return code;
    }


}
```

Listing A.66: NumConst.java

```
package edu.columbia.plt.gal.ast;

/**
 * Num Constant
 * @author Athar Abdul-Quader
 */
public class NumConst extends Node
{
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("new GalVal(new GalNum(");
        code.append(getText());
        code.append("))");
        return code;
    }
}
```

Listing A.67: Parms.java

```java
package edu.columbia.plt.gal.ast;

import java.util.ArrayList;
import java.util.List;
import antlr.collections.AST;
/**
 * Parameter list
 * @author Athar Abdul-Quader
 */
public class Parms extends Node
{
    public List<Node> getParmList()
    {
        ArrayList<Node> params = new ArrayList<Node>();
        for (AST child = getFirstChild(); child != null; child = child.getNextSibling())
        {
                params.add((Node)(child));
        }
        return params;
    }
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        List<Node> params = getParmList();
        if (params.isEmpty())
                return code;
        code.append("IGalRef␣");
        code.append(params.get(0).gen());
        for (int i=1; i<params.size(); i++)
        {
                code.append(",␣");
                code.append("IGalRef␣");
                code.append(params.get(i).gen());
        }
        return code;
    }
}
```

Listing A.68: Pluseq.java

```java
package edu.columbia.plt.gal.ast;

/*
 * Plus-equals Operator
 * @author Jay Winters
 */
public class Pluseq extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".pluseq(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.69: Plus.java

```
package edu.columbia.plt.gal.ast;

/*
 * Plus Operator
 * @author Jay Winters
 */
public class Plus extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".plus(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.70: Prog.java

```
package edu.columbia.plt.gal.ast;

import java.util.List;
import java.util.ArrayList;
import java.lang.reflect.*;
import edu.columbia.plt.gal.SymbolTable;
import edu.columbia.plt.gal.GalStdLib;
/**
 * Program node - represents an entire Gal Program
 * @author Athar Abdul-Quader
 * @author Oren B. Yeshua
 */
public class Prog extends Node
{
    static SymbolTable<Integer> oFuncTable = new SymbolTable<Integer>(null);

    public void setEnv(SymbolTable<Node> parent, boolean flag)
    {
        // first put in the built-in functions
        Class c = GalStdLib.class;
        Method[] methods = c.getMethods();
        for (int i=0; i < methods.length; i++)
        {
                oFuncTable.put(methods[i].getName()+":"+methods[i].getParameterTypes().len
        }
        for (Node decl : getDeclarations())
        {
                if (decl instanceof FDecl)
                {
                        FDecl func = (FDecl) decl;
                        oFuncTable.put(func.getName().getText()+":"+func.getParms().getPar
                }
        }

        env = parent;
        for (Node child = getChildNode(); child != null; child = child.getSiblingNode())
        {
                child.setEnv(env, flag);
        }

    }

    public List<Node> getDeclarations()
    {
        List<Node> decls = new ArrayList<Node>();

        for(Node child = getChildNode();
            child != null;
            child = child.getSiblingNode())
            {
                assert ((child instanceof VDecl) || (child instanceof FDecl)) : "Every␣chi
                decls.add(child);
            }

        return decls;
    }

    public List<Node> getFuncDeclarations()
    {
        List<Node> fDecls = new ArrayList<Node>();

        for (Node child : getDeclarations() )
        {
                if (child instanceof FDecl)
                        fDecls.add(child);
        }
        return fDecls;
    }

    public List<Node> getVarDeclarations()
    {
        List<Node> varDecls = new ArrayList<Node>();

        for (Node child : getDeclarations() )
        {
                if (child instanceof VDecl)
                        varDecls.add(child);
        }
        return varDecls;
    }
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        for (Node decl : getVarDeclarations() )
        {
                code.append("public static ");
                code.append( ((VDecl)decl).getDeclarer());
                code.append(";\n");
        }

        code.append("static {\n");
        for (Node decl : getVarDeclarations() )
        {
                code.append( ((VDecl)decl).getAssigner());
                code.append(";\n");
        }
        code.append("}\n");

        for(Node decl : getDeclarations())
        {
            if (decl instanceof FDecl)
            {
                code.append("public static ");
                code.append(decl.gen());
                code.append("\n");
            }
        }

        return code;
    }
}
```

Listing A.71: Return.java

```
package edu.columbia.plt.gal.ast;

/**
 * Return statement
 * @author Athar Abdul-Quader
 */
public class Return extends Node
{
    public StringBuilder gen()
    {
        StringBuilder sb = new StringBuilder("return (");
        sb.append(getChildNode().gen());
        sb.append(")");
        return sb;
    }

    public boolean hasReturn()
    {
        return true;
    }
}
```

Listing A.72: SetConst.java

```
package edu.columbia.plt.gal.ast;

import edu.columbia.plt.gal.SymbolTable;
import antlr.collections.AST;


/**
 * Set Constant
 * @author Oren B. Yeshua
 * @author Athar Abdul-Quader
 */
public class SetConst extends Node
{

    public boolean isRange()
    {
        return !(getFirstChild() instanceof Args);
    }

    public String[] getRange()
    {
        String[] range = new String[2];
        AST start = getFirstChild().getFirstChild();
        range[0] = start.getText();
        range[1] = start.getNextSibling().getText();
        return range;
    }

    public Args getParms()
    {
        return (Args)getFirstChild();
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("new␣GalRef(new␣GalSet(");

        if(isRange())
        {
            String[] range = getRange();
            code.append(range[0]);
            code.append(',');
            code.append(range[1]);
        }
        else
        {
            code.append(getParms().gen());
        }
        code.append("))");

        return code;
    }

    /**
     * Overriden for leaf of heterogeneous AST
     */
    public void setEnv(SymbolTable<Node> parent, boolean flag)
    {
        env = parent;
    }
}
```

Listing A.73: Stmt.java

```
package edu.columbia.plt.gal.ast;

/*
 * Statement base class
 */
public abstract class Stmt extends Node
{
}
```

Listing A.74: StringConst.java

```
package edu.columbia.plt.gal.ast;

/**
 * String Constant
 * @author Athar Abdul-Quader
 */
public class StringConst extends Node
{
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("new GalVal(new GalString(");
        code.append(getText());
        code.append("))");
        return code;
    }
}
```

Listing A.75: Timeseq.java

```
package edu.columbia.plt.gal.ast;

/*
 * Times-equals Operator
 * @author Jay Winters
 */
public class Timeseq extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".timeseq(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }
}
```

Listing A.76: Times.java

```
package edu.columbia.plt.gal.ast;

/*
 * Times Operator
 * @author Jay Winters
 */
public class Times extends BinaryOp
{

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(" + getLValue().gen());
        code.append(".times(");
        code.append(getRValue().gen());
        code.append("));");
        return code;
    }


}
```

Listing A.77: UnaryOp.java

```
package edu.columbia.plt.gal.ast;

import java.util.Hashtable;

/**
 * Unary Operator
 * @author Athar Abdul-Quader
 * @author Jay Winters
 */
public class UnaryOp extends Node
{
    private static Hashtable<String, String> mSymbolToName;

    static
    {
        mSymbolToName = new Hashtable<String, String>(2);
        mSymbolToName.put("!","not");
        mSymbolToName.put("|","length");
        mSymbolToName.put("-","minus");
    }

    public Node getValue()
    {
        return getChildNode();
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("(");
        code.append(getValue().gen());
        code.append(").");
        code.append(mSymbolToName.get(getText()));
        code.append("()");
        return code;
    }
}
```

Listing A.78: VarCheck.java

```
package edu.columbia.plt.gal.ast;

import edu.columbia.plt.gal.SymbolTable;

/**
 * Identifier
 * @author Athar Abdul-Quader
 */
public class VarCheck extends Id
{
        public void setEnv(SymbolTable<Node> parent, boolean flag)
        {
                env = parent;
                if (env.get(getText()) == null)
                {
                        System.err.println("Variable " + getText() + " not declared in thi
                }
        }
}
```

Listing A.79: Var.java

```
package edu.columbia.plt.gal.ast;

/*
 * Variable
 *
 * For input to a GalType constructor
 *
 * @author Oren B. Yeshua
 */
public class Var extends Node
{
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder(getText());
        code.append(".getObj()");
        return code;
    }
}
```

Listing A.80: VarParm.java

```
package edu.columbia.plt.gal.ast;

import edu.columbia.plt.gal.SymbolTable;
```

```
/**
 * Identifier
 * @author Athar Abdul-Quader
 */
public class VarParm extends Id
{
        public void setEnv(SymbolTable<Node> parent, boolean flag)
        {
                env = parent;
                env.put(getText(), this);
        }
}
```

Listing A.81: VDecl.java

```
package edu.columbia.plt.gal.ast;

import edu.columbia.plt.gal.SymbolTable;
import antlr.collections.AST;
import java.util.HashMap;

/**
 * Variable declaration
 * @author Athar Abdul-Quader
 */
public class VDecl extends Node
{
    static HashMap<String, String> prims;
    static HashMap<String, String> objects;

    static {
        objects = new HashMap<String, String>();
        objects.put("graph", "GalGraph");
        objects.put("edge", "GalEdge");
        objects.put("set", "GalSet");
        objects.put("queue", "GalQueue");
        objects.put("vertex", "GalVertex");
        objects.put("vector", "GalVector");

        prims = new HashMap<String, String>();
        prims.put("num", "GalNum");
        prims.put("bool", "GalBool");
        prims.put("string", "GalString");
    }

    public String getConstructor()
    {
        StringBuilder sb = new StringBuilder();
        String type = getTypeName();
        if (objects.containsKey(type))
        {
                sb.append("GalRef(new␣");
                sb.append(objects.get(type));
                sb.append("())");
                return sb.toString();
        }
        else
        {
                sb.append("GalVal(new␣");
                sb.append(prims.get(type));
                sb.append("())");
                return sb.toString();
        }

        //return type;
    }

    public Node getInitializer()
    {
        return (Node)(getFirstChild().getNextSibling());
    }

    public String getId()
    {
        Node init = getInitializer();
        if(init instanceof BinaryOp || init instanceof Assign)
        {
            return init.getChildNode().getText();
        }
        return init.getText();
    }

    public String getTypeName()
    {
        return getFirstChild().getFirstChild().getText();
    }

    public StringBuilder getDeclarer()
    {
        StringBuilder code = new StringBuilder();
        code.append("IGalRef␣");
        code.append(getId());
        code.append("␣=␣new␣");
        code.append(getConstructor());
        return code;
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        code.append(getDeclarer());

        Node init = getInitializer();
        if (init instanceof BinaryOp || init instanceof Assign)
        {
                code.append(";\n");
                code.append(init.gen());
        }
        return code;
    }

    public StringBuilder getAssigner()
    {
        StringBuilder code = new StringBuilder();
        Node init = getInitializer();
        if (init instanceof BinaryOp || init instanceof Assign)
        {
                code.append(init.gen());
        }
        return code;
    }

    public void setEnv(SymbolTable<Node> parent, boolean flag)
    {
        env = parent;
        String id = getId();
        if (env.existsInCurrent(id))
                System.err.println("Variable␣" + id + "␣already␣exists␣in␣current␣scope."+
        env.put(id, this);

        // Check for legal type
        String type = getTypeName();
        if(!(prims.containsKey(type) || objects.containsKey(type)))
        {
            System.err.println("\""+type + "\"␣is␣not␣a␣recognized␣GAL␣type."+getLineStr()
        }
    }
}
```

Listing A.82: While.java

```
package edu.columbia.plt.gal.ast;

/**
 * While statement
 * @author Athar Abdul-Quader
 */
public class While extends Node
{
    public Node getCondition()
    {
        return getChildNode();
    }

    public Body getBody()
    {
        return (Body)getCondition().getSiblingNode();
    }

    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder();
        code.append("while␣(␣(");
        code.append(getCondition().gen());
        code.append(").getObj().castBool().getBool())\n");
        code.append("{\n");
        code.append(getBody().gen());
        code.append("}\n");
        return code;
    }
}
```

Listing A.83: Wildcard.java

```
package edu.columbia.plt.gal.ast;

/*
 * Wildcard
 *
 * @author Oren B. Yeshua
 */
public class Wildcard extends Node
{
    public StringBuilder gen()
    {
        StringBuilder code = new StringBuilder("\"");
        code.append(getText());
        code.append('\"');
        return code;
    }
}
```

A.4.3 package edu.columbia.plt.gal.err

Listing A.84: GalException.java

```
package edu.columbia.plt.gal.err;

public class GalException extends RuntimeException
{
    public GalException(String msg){super(msg);}
}
```

Listing A.85: GalParseException.java

```
package edu.columbia.plt.gal.err;

public class GalParseException extends GalException
{
    public GalParseException(String msg){super(msg);}
}
```

Listing A.86: GalRuntimeException.java

```
package edu.columbia.plt.gal.err;

public class GalRuntimeException extends GalException
{
    public GalRuntimeException(String msg){super(msg);}
}
```