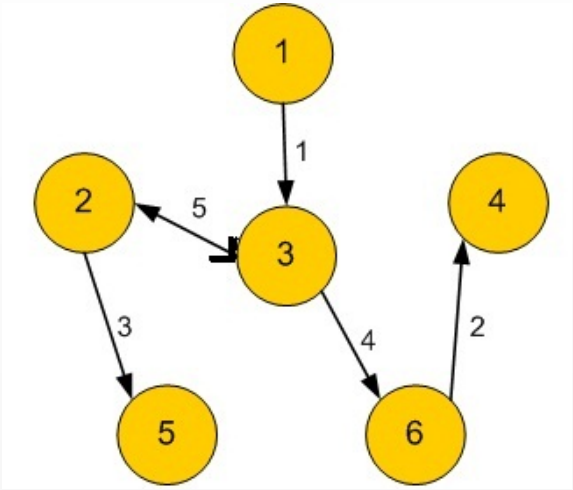
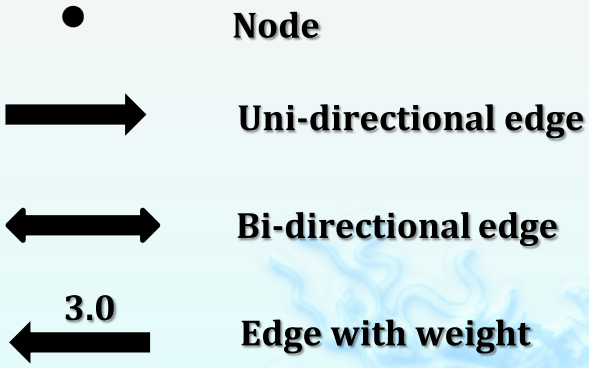
GRAPL: GRAph Processing Language

GRAPL TEAM
Ryan Turner, Andres Uribe
Di Wen, Lili Chen, Yi Yang

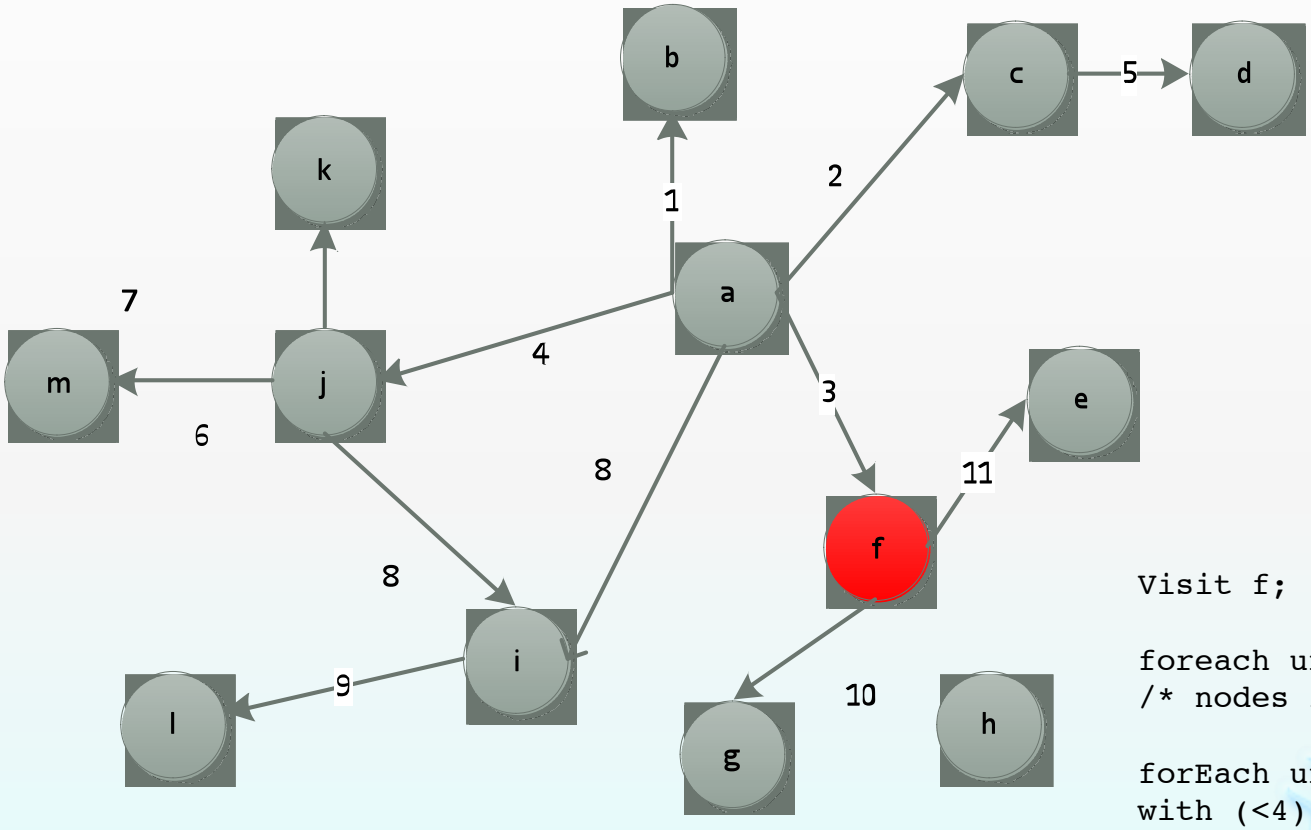
Overview

- GRAPL is a simple way to create and navigate though directed weighted graphs
- Target: people who know a little about programming language, but do not want to mess up with the pointers, references, complicated class structures.





Graph Traversal




```
Visit f;

foreach unvisited n to i { }
/* nodes i j a */

forEach unvisited n from a
with (<4) { }
/* nodes a b c */

forEach unvisited n to m with
<=8 { }
/* nodes a b c */
```

GRAPL Types

Type	Example
node	`node n1;` `n1 = start_node;`
number	`number x;` `x = 1.4;`
boolean	`boolean isVisited;` `isVisited = false;`

list	`list path_list;`
	`path_list = [a, b, c];`
	`path_list = d :: path_list;`

void	`void main() { }`

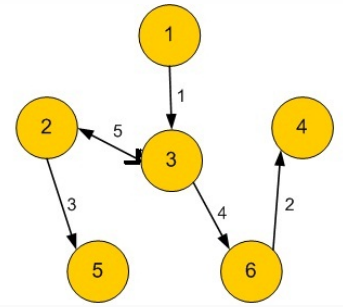
GRAPL Control Statements

Type	Example
graph creation (implicit node declaration)	`graph [a >> b <>3 c, b >> d, a <> e];`
while loop	`while (i < length(path_list)) { }`

if – then - else	`if (x > 5) then`
	`  visit n;`
	`else`
	`  visit p;`

forEach loop	`forEach unvisited p from start`
	`{ /* iterate over children */`
	`}`

GRAPL Key Features



Graph creation syntax is as simple and compact as possible

```
/* sample graph statements */
graph [n3 <<1 n1];
graph [n2 <>5 n3 >>4 n6, n6 >>2 n4];
graph [n2 >>3 n5];
```

Unlike other types, nodes do not have to be declared in advance; they are created implicitly in the graph statement. There is one global graph per GRAPL program.

ForEach loop makes typical graph navigation tasks easy

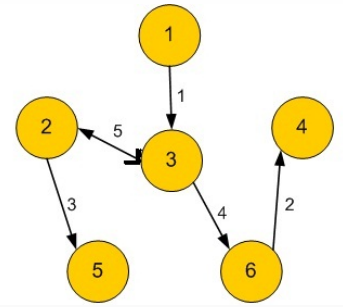
```
/* sample forEach */
forEach unvisited n from start with (< 4.0)
{
   path_list = n :: path_list;
}
```

forEach statement has optional qualifiers (visited, unvisited) and edge-weight predicate (> 4.0)

Development Process & Tools

- Parser/Scanner development, output routed to grapl-printer (intermediate product)
- Compiler and java-printer development
- Testing
- Tools:
  - Eclipse with OcaIDE plug-in for Ocaml editing
  - Command-line with Makefile for Ocaml compilation
  - Netbeans for Java
  - Google code SVN repository

Implementation



example.gpl

```
/* GRAPL */
void main()
{
  node n1;
  node n2;
  node n3;
  node n4;
  node n5;
  node n6;
  graph [n3 <<1 n1];
  graph [n2 <>5 n3 >>4 n6];
  graph [n2 >>3 n5];
...
```

Scanner / Parser
scanning, parsing, syntactic checking

GRAPL-AST

Compiler
translation and semantic checking

JAVA-AST

Java Printer

```
/* Java */
import lib.*;
class GraplProgram
{ ArrayList<Node> nodes;
  ArrayList<Edge> edges; ...}
Node n1 = new Node("n1");
Node n2 = new Node("n2");
...
Node n6 = new Node("n6");
Graph g = new Graph();
g.addEdge(n1,n3,1);
g.addBedge(n3,n2,5);
...
```

GraplProgram.java

GRAPL Standard Library

Java Backend Classes

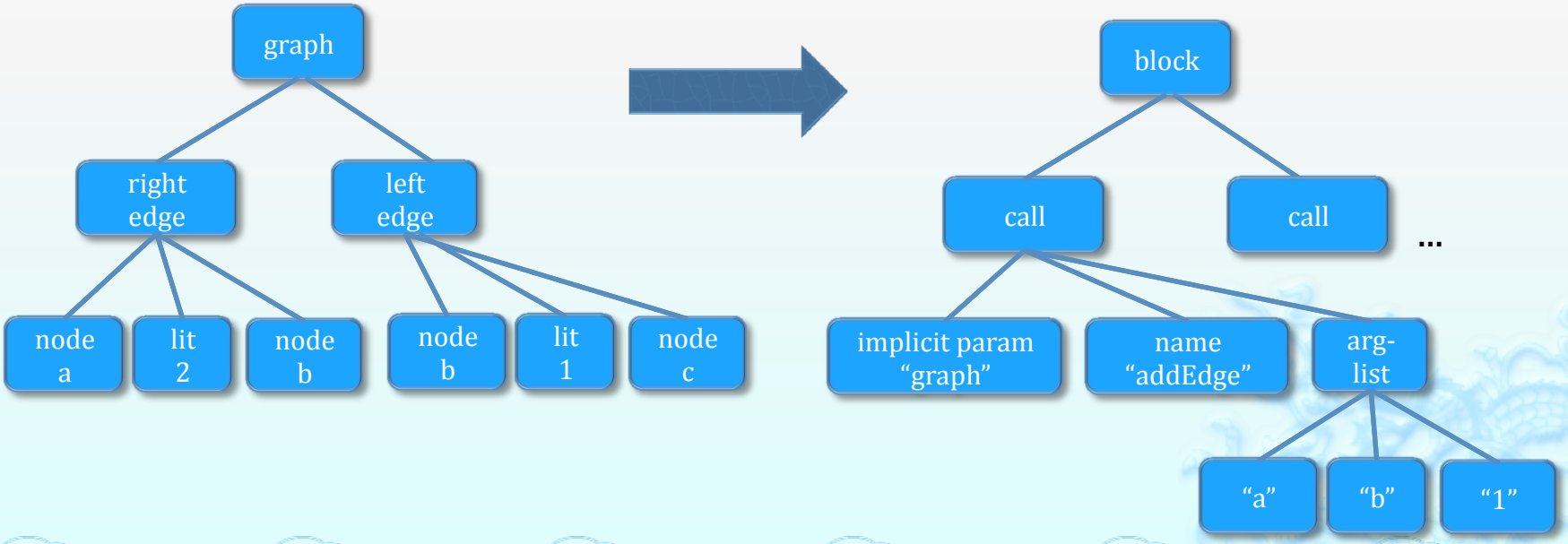
Java compilation

More on Compiler

- Compiler performs a translation from a grapl-ast to a java-ast while performing semantic checking along the way
- Many simple statements in GRAPL become function calls (or sequences of calls) to the Java backend library

```
/* GRAPL */
graph [a >>2 b << c];
```

```
/* Java */
graph.addEdge("a","b",2);
graph.addEdge("b","c",1);
```



Java Backend Architecture

- Compiler performs a translation from a grapl-ast to a java-ast while performing semantic checking along the way
- Many simple statements in GRAPL become function calls (or sequences of calls) to the Java backend library

```
import lib.*; // backend library

// globals common to all GRAPL programs
public static Graph graph;
public static GraplLib library;

// globals for this specific

public Class Example
{
   void main()
   {
      node n1;
      node n2;
      number x;
      x = 0;
      graph [a >> b >> c <> d];
...
```

```
class GraplLib
/* manages global state,
provides utilities */
```

```
class Node
/* node information
 */
```

```
class Edge
/* edge weight and
direction */
```

```
class Graph
/* maintains
relationships between
nodes and edges */
```

```
class List
/* implements an
ArrayList<node> */
```

GRAPL Standard Library

- The GRAPL Standard Library implements useful and often-needed functions like numChildren, reverse (for lists), depth-first-search, etc
- Standard Library functions (as opposed to built-in functions) are written in GRAPL
- They are precompiled and imported into every GRAPL program so that the user has automatic access to them

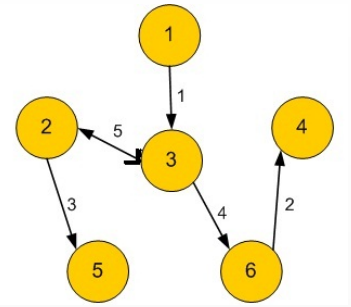
Summary & Challenges

- Implicit declaration of nodes in the graph statement along with a single global graph per program: a major can of worms
  - Nodes must be handled specially in the backend, sometimes as strings
  - The type of java-ast-node for a grapl-ast-node depends on the symbol table at compile time; it may become either a jast-identifier or a a jast-function-call to `graph` to retrieve a named node
  - Semantic checking cannot guarantee that all node references will be valid at run-time
- Appropriately initializing both Java objects and Java primitives at the right time
- Making standard library functions automatically available
  - Backend Java for standard library requires access to global variables in the GraplProgram file for full functionality

Lessons Learned

- Test early, test often
- Get Unix. Get it now.
- When all else fails, add another layer of indirection (e.g., a Java wrapper)
- Simple language restrictions != simple compiler features
- Use shell scripts

Team GRAPL



- Lili Chen – scanner, grapl- and java-printers, testing
- Ryan Turner – parser, compiler, team leader
- Andres Uribe – parser, compiler, standard library
- Di Wen – parser, testing suite
- Yang Yi – Java backend, standard library

Tutorial / Example

```
list dfs (node start, node finish) {
 list l;
 unvisitAll();
 return dfs_helper(start, finish,l);
}
list dfs_helper(node start, node finish, list l)
{
 visit start;
 if (start != finish) then
 {
  forEach unvisited n from start
  {
  n::l;
  if(n!=finish)then{
   dfs_helper(n, finish,l);
   }
   else
   { return l; }

  }}
 else
 {}
 return l;
}
```

```
void main(){
 graph [a >>b >>c>>e>>f];
 print(dfs(a, f));
}
```

Tutorial / Example

```
node n1;
void main()
{
graph [n1 <<1 n2 <>2 n3, n4 <>2 n2, n5 >>3 n2];

forEach unvisited n from n2 with ( < 2 ) {
 print(n);
        visit n;}
forEach unvisited n from n2 with ( <= 1 ) {
        print(n);
        unvisit n;}
visit n5;
visit n3;
forEach visited n to n2 with ( > 2 ) {
        print(n); }
forEach visited n to n2 with ( >= 3 ) {
        print(n); }
}
```