LARA Tutorial

1. Static Analysis

Tiago Carvalho, Pedro Pinto, João Bispo, Ricardo Nobre, Luís Reis, and João M.P. Cardoso

University of Porto, FEUP, Porto, Portugal

April 20th, 2016

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

ANTAREX 10^{18}

SPeCS
Special-Purpose Computing Systems, languages and tools

Objectives

- First steps on LARA
- Introduction to main constructs
- Retrieve static code information
- Generate reports

Some Information

MANET:

- LARA + Cetus*
- ANSI C
- http://specs.fe.up.pt/tools/manet/

- The presented examples
  - Dijkstra from MiBench
  - Disparity from San Diego Vision Benchmark Suite

*Cetus compiler: http://cetus.ecn.purdue.edu/

1.1 Target Language Report

- Goal: Report information regarding the target language
  - Join point: points of interest in the code
  - Attribute: information concerning a join point
  - Action: target code transformation

- Strategy
  - Use *Weaver* object
  - Show available join points and root
  - Iterate join points
    - List attributes, selects and actions

Weaver object members:

```
String   root
String[] joinpoints
String[] selectsOf( jpName )
String[] actionsOf( jpName )
String[] attributesOf( jpName )
```

1.2 Caller and Callee

- Goal: Report calls to, and from a target function

- Strategy
  - Select pairs functions and then calls
  - Filter call by target name
    - When our target function is called
  - Filter function by target name
    - When our target function is a caller

Example output from pretty report:

```
Functions that call enqueue
  dijkstra x2
Functions that are called by enqueue
  malloc x1
  fprintf x1
  exit x1
```

1.3 Static Call Graph

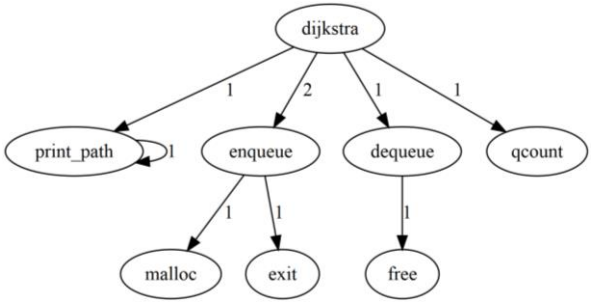
- Goal: Print a static call graph in dot format

- Strategy
  - Select pairs <function, call>
  - Count occurences of each pair
  - Iterate LaraObject and print

Example dot and resulting graph:

```
digraph static_call_graph {

            print_path->print_path [label="1"];
            enqueue->malloc [label="1"];
            enqueue->exit [label="1"];
            dequeue->free [label="1"];
            dijkstra->enqueue [label="2"];
            dijkstra->qcount [label="1"];
            dijkstra->dequeue [label="1"];
            dijkstra->print_path [label="1"];
}
```



1.4 Find Calls to Library Functions

- Goal: Report calls to functions from Math.h or stdio.h

- Strategy
  - Select function calls
    - Filter by name
  - Store function name and location
  - Print Report
  - Also: find and report system calls in loops

Example output:

```
[mibench_dijkstra.c]
  printf
    lines: 43 112 139 140 142
  fflush
    lines: 44
  fprintf
    lines: 55 151 152
  fopen
    lines: 156
  fscanf
    lines: 162
```

1.5 Static Code Report

- Goal: Generate an extended report about program structure

- Strategy
  - Collect
    - general information (files, functions, calls)
    - loop information
    - function information
  - Store in JavaScript objects
  - Pretty print

Example output:

```
-== [ Loop Information ] ==-

==============================
Type       Total    Innermost
------------------------------
For        5        4
While      2        1
Do-While   0        0
------------------------------
Total      7        5
==============================

Largest Loop Nest: 2 @ c_file.c : dijkstra : 124
```

Takeaway Points

- Easy to select points of interest in the code
- Attributes provide a wealth of information
- We can generate metrics, reports, understand an application, find potential problems, ...
- Can use JavaScript to complement the strategies