ABB	Technical Report Corporate Research	**2016/CHCRC/C/TR/356**

Issued by department CHCRC/C	Create date 5/31/2016	Page 1/10

Customer		Classification	Public
Doc. title	**Athena - A Tool for Feature Mining Based on Preprocessor Directives**		
Project name/ID		Status of document	Published
Main Author	Michael Wahler	Project ID (CRID) or Order no	
Additional Author(s)		Research Area	
Distribution			
Attested by Reviewer	Raphael Eidenbenz		
Approved by	Michael Wahler	Final Approval Date	6/1/2016

9ADB1-002_Technical_Report.dot Rev 2.11

Athena - A Tool for Feature Mining Based Preprocessor Directives

Summary:

Athena is a tool for feature mining based on preprocessor directives. To this end, it analyzes C/C++ code preprocessor directives (#if/#ifdef), extracts the symbols used in these directives, and computes a dependency graph. The resulting graph can be used for analyzing the variability of the code, e.g. in the context of software product lines (SPL).

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Purpose

The purpose of this report is to inform users about how to use Athena and developers about how to extend Athena.

Athena is a tool for feature mining based on preprocessor directives. To this end, it analyzes C/C++ code preprocessor directives (#if/#ifdef), extracts the symbols used in these directives, and computes a dependency graph. The resulting graph can be used for analyzing the variability of the code, e.g. in the context of software product lines (SPL).

The results of Athena may also help developers to improve test coverage by using the symbols found in a product-line-testing approach and to refactor their code to minimize the use of preprocessor flags.

Athena was developed as part of the CR project SPAN (NORRD ID 2851).

1.2 Definitions

SPL	Software Product Line
SVG	Scalable Vector Graphics
LLVM	The LLVM Project is a collection of modular and reusable compiler and toolchain technologies (http://llvm.org)

2 HOW TO USE ATHENA

Athena is a command-line program that runs on Linux.

2.1 Requirements

The following programs must be installed to run Athena:

- bash
- perl
- python
- graphviz

These programs are often installed by default in common Linux distributions, or they can be easily installed (e.g., using apt-get in Ubuntu Linux).

The tool is built for Linux and tested on 32-bit Ubuntu with a kernel version 3.13 and a 64-bit Ubuntu with a kernel version 4.2.

Athena can be downloaded from the ABB Codebits[1] website.

[1] https://codebits.pl.abb.com/athena

2.2 Quick Start

Assume this simple contrived C program *test.c* in folder *source*.

```
# ifdef __linux__
# ifdef __32bit__
int x;
# elif defined (__64bit__)
long x;
# endif
```

Running

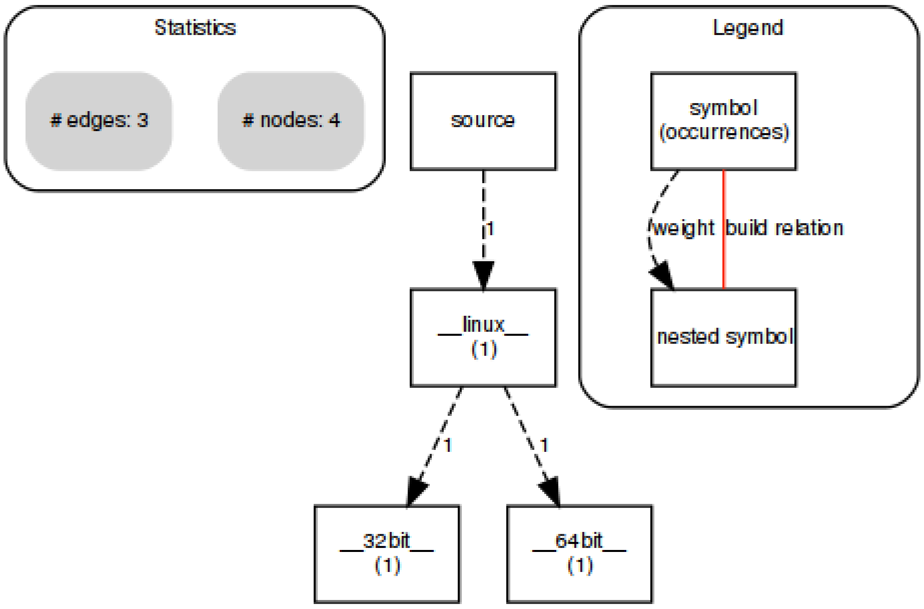
```
$ ./athena.sh source
```

starts Athena, which produces the following output:

```
* Checking dependencies....... ok
* Source directory........... source/
* Exclude directories......... 1 (see exclude_directories.txt)
* Scan header files........... no (use --include-headers to include
scanning of header files)
***************************************************************************
*
*
* Recursively searching source/ for Makefiles
* Symbol definitions (-D) written to tmp/makefiledefinitions.txt.
* Recursively searching source/ for preprocessor directives
* Temp output written to tmp/grep.txt (format: filename line# command)
* Running Python script to compute graph
* Running dot to visualize graph
* Graph written to graphs/source.svg
```

The resulting file, *graphs/source.svg* can be viewed with most web browsers or vector graphic programs such as *Inkscape*[2].

[2] https://inkscape.org/



Besides analyzing C/C++ files, Athena also scans for build instructions (e.g., Makefiles). When Athena finds dependencies between preprocessor symbols in the build instructions, it highlights these relations as colored edges in the graph.

2.3 Usage

The main executable of Athena is *athena.sh*. It can be invoked as follows:

```
athena.sh <folder> [--include-headers] [--edge-threshold=n] [--show-
weight=true|false]
```

- <folder> denotes the name of a folder (either absolute or relative folder names are possible). This folder is recursively searched for C/C++ files.
- --include-headers (optional) extends the analysis to header files. By default, only .c/.cc/.cpp files are scanned
- --edge-threshold=n (optional) can be used to eliminate edges with a weight less than n from the output graph. Using this parameter, a graph of the most important dependencies can be created.
- --show-weight=true|false (optional) can be used to toggle the display of edges in the graph. By default, the width of the edges corresponds to their weight (with a maximum width of 5, otherwise the graph gets too cluttered). By default, this parameter is set to true.

Athena can be further configured using the file *exclude_directories.txt*. All folders listed in this file (one folder name per line) are ignored. This is helpful when the headers of large libraries (e.g., boost) are inside the folder to be scanned.

2.4 Displaying Results

Athena produces a graph in the SVG format, which can be displayed by most web browsers and vector graphic programs such as Inkscape.

Athena comes with a file *index.html*, which realizes a minimal web interface using JavaScript and CSS. When opened in a browser, the web interface offers a menu to load SVG files. Once the SVG file has been loaded, the parents of each node can be highlighted by clicking on the name of the node.

Note that some web browsers (e.g., Google Chrome) do not permit the execution of JavaScript code in a local file. To circumvent this restriction, a simple web server can be started by running

```
$ python -m SimpleHTTPServer 8080
```

and opening http://localhost:8080 in the web browser.

3 HOW TO EXTEND ATHENA

This section is intended for developers who want to extend Athena or fix bugs.

3.1 Project Structure

Filename	Purpose
athena.sh	Main script. Searches for preprocessor directives, triggers the execution of the analysis (in Python), and generates a graph using graphviz.
athena_dirgraph.py	The main Python file. Parses the input generated from athena.sh and computes output in the *dot* format for graphviz.
athena_globalsymbols.py	Defines a few globally used variables.
athena_makeparser.py	Used by athena_dirgraph.py. Computes dependencies between symbols that are specified in the build instructions.
athena_stringoperations.py	Commonly used operations on strings. As an example, it defines a function that extracts a list of symbols from a statement *#if A* || *B==2*.
athena.readme	A short readme file.
demo.sh	A shell script that runs Athena on a simple C program and opens a web browser to view the resulting graph.
index.html	A simple web interface to viewing graphs and highlighting parent nodes.
styles.css	CSS stylesheet for the web interface.
graphhighlighter.js	JavaScript program that is used by index.html to compute parent node relations and highlight the nodes.

3.2 Suggestions for Improvement

3.2.1 Parser

Athena parses precompiler statements (#if, #ifdef, #elif) with a best-effort parser. This means that Athena tries to extract as many meaningful symbols as possible. There is however no proper parser (i.e., based on a grammar) in Athena. As a consequence, some symbols are not recognized (e.g., the commonly used symbol *0*). Athena would benefit from a proper parser to ensure that all precompiler symbols are properly extracted. Frameworks such as LLVM may be useful to this end.

The main parsing functions can be found in

- parse_input() in athena_dirgraph.py
- extract_symbols_from_statement() in athena_stringoperations.py

3.2.2 Build Instructions

Athena uses a simplistic approach for finding potential relations between precompiler symbols: it assumes that symbols defined on the same line in a Makefile denote a relation between the symbols. Symbols not defined on the same line are not related.

Example:

```
target1:
  gcc test.c –D SYMBOL1 –D SYMBOL2 –o test.o
target2:
  gcc test.c –D SYMBOL1 –D SYMBOL3 –o test-o
```

In this example, Athena assumes that SYMBOL1 and SYMBOL2 are related as well as SYMBOL1 and SYMBOL3.

The build instruction parsing functions can be found in

- Step 1 in athena.sh
- compute_constraints_as_dot() in athena_makeparser.py

3.2.3 Web Interface

The web interface works well for small graphs. There is however supported needed for displaying and navigating larger graphs, e.g.

- zooming in and out
- searching for symbol names
- folding/unfolding selected branches of the graph

These functionalities can be found in

- index.html

- graphhighlighter.js

4 CONCLUSIONS

Athena in its current form helps developers to see all preprocessor symbols used in #if/#ifdef statements in their code. It also shows the relations between the symbols and how often symbols/relations occur in the source code.

Athena is a side product from an ABB research project. Therefore, it has several limitations and bugs. Also, some parts of Athena contain simplifications. The next section gives an overview of these.

5 LIMITATIONS AND FUTURE WORK

5.1 Limitations

- symbols that clash with the keywords of Graphviz (*node, edge, graph, digraph, subgraph, strict*) are renamed to <name>___Athena
- Athena removes double quotes from symbols (e.g., *boost* uses the symbol *"string"* with double quotes)
- "0" as a symbol is not yet supported. The symbol "0" is however often used to comment out lines of code, e.g.

```
#if 0
<code here>
#endif
```

  Therefore, it should be supported in future versions of Athena.
- The generated graphs can get very large such that the *dot* tool of graphviz runs out of memory. Athena could warn the user that *dot* may file such that the user does not waste several minutes waiting for *dot* to fail.

5.2 Future Work

- Proper investigations about how symbols are related in build instructions (e.g., Makefiles) are needed (see Section 3.2.2)
- Athena in its current form "only" computes and displays a graph of preprocessor symbols and their relations. This graph can aid an expert in defining a feature model of the analyzed code.
  In the future, the generated graph could be automatically analyzed and condensed such that it represents more information and less data. This is however subject of future research.

5.3 Known Bugs

- It is not possible to use absolute path names in exclude_directories.txt.

Rev.	Chapter	Description	Date / Dep. / Name
01	All	Initial document	2016-05-31 / CHCRC-C5 / Michael Wahler
01a	some	changed copyright notice, added link to SPAN project, added links for LLVM and Inkscape, added link to ABB Codebits	2016-06-01 / CHCRC-C5 / Michael Wahler