Shell Scripting (Waffle Language)

Rishi Reddy Cheruku
Department of Computer Science
University Of Akron
rc81@zips.uakron.edu

Project Goal

Our goal is to define type safe abstractions for writing a shell programming language for bash environment used by most of the native Linux distributions.

Motivation

Building a unique language which contains concepts of sql, bash commands, Functions and Modules will always make any developers life easy. We got an opportunity to implement bash commands in such a language known as Waffle is a challenging and interesting. Using the boost file system added a lot of strength in implementing the language and provided the feasibility to achieve what bash commands are capable off.

Introduction

Shell scripts provide us to program commands in chains and make the system execute them like batch files (a scripted event). Shell scripts are not just commands but programs in their own right. Scripting allows us to use programming functions directly within OS interface. The main aim is to write programs for basic commands that we are already familiar to.

Understanding Waffle Workflow

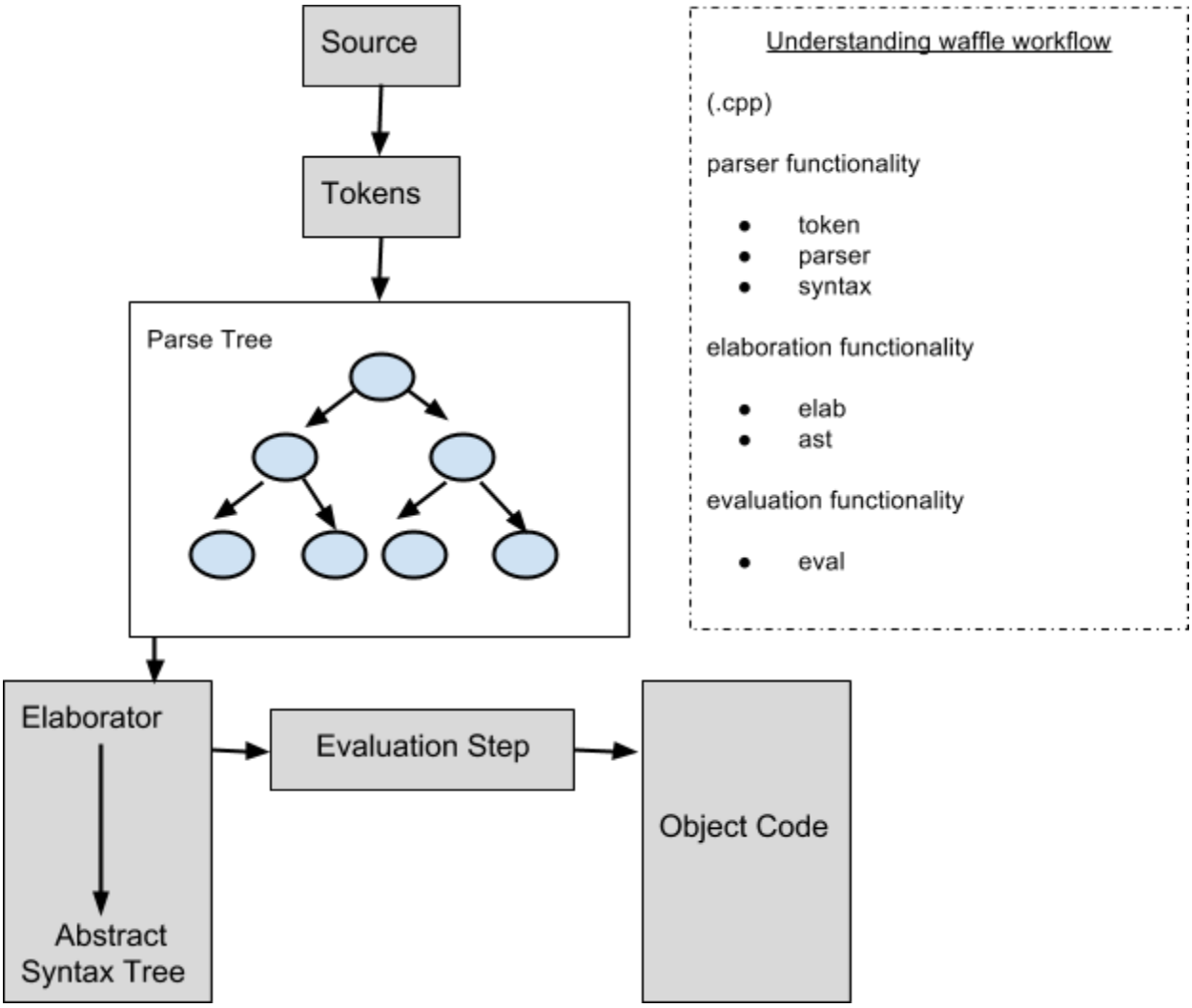

Figure 1: Steps involved in processing shell commands in waffle language.

Implementation

Implementation for five basic file system commands ls, mkdir, rmdir, mv and cd.

Syntax and Semantics

Keywords: *ls, mkdir, rmdir, cd, mv* are the keywords declared in token file.

Tokens

ls_tok = ls
mkdir_tok = mkdir
rmdir_tok = rmdir
cd_tok = cd
mv_tok = mv

Parser

To parse the script commands without execution in order to check for syntax errors. We have defined functions for these five commands and each function returns tree after parsing the input string expression.

Parse an expression:

- ls Command - list the content of a directory.
    - ls-expr ::= 'ls' string-expr
- mkdir Command - create a directory.
    - mkdir-expr ::= 'mkdir' string-expr
- rmdir Command - remove a directory.
    - rmdir-expr ::= 'rmdir' string-expr
- cd Command - change working directory.
    - cd-expr ::= 'cd' string-expr string-expr
- mv Command - rename a directory or moves directory from one place to another
    - mv-expr ::= 'mv' string-expr

Parser file returns tree Ls_tree(k, t)/Mkdir_tree(k,t)/Rmdir_tree(k,t)/Cd_tree(k,t)/Mv_tree(k, t, t1) after parsing the input using tokens created in token file.

Note: If the input expression is not string-expr after command, the function returns error.

Elaboration Step:

In this step we are elaborating parse tree (Ls_tree * t) returned from parser as term using typing rules of Figure2.

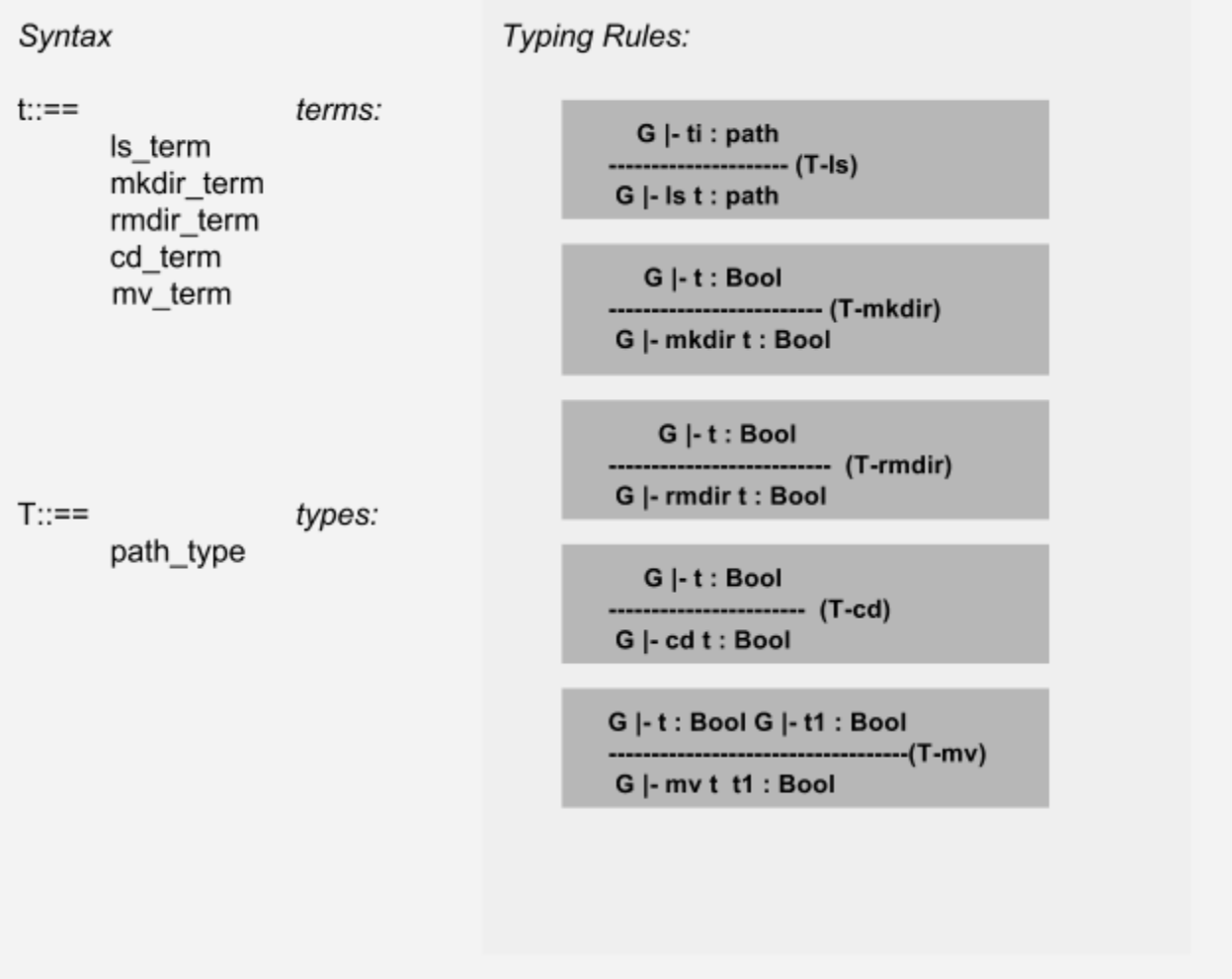

Figure 2: Shell Scripting Evaluation Rules.

***Abstract syntax tree**:*

The internal representation of our compiler is specified by an abstract syntax tree (ast file) in terms of statements, expressions or identifiers. An AST is usually the result of the syntax analysis phase of a compiler. This will allow us to verify the correctness of our program. Finally we render given term to output stream.

***Terms*:**

- init_node(ls_term, "ls");
- init_node(mkdir_term, "mkdir");
- init_node(rmdir_term, "rmdir");
- init_node(cd_term, "cd");
- init_node(mv_term, "mv");
- init_node(path_term, “path”);

Types:

- *init_node(path_type, "path-type");*

Evaluation Step

In this step we have evaluated ls, mkdir, rmdir, cd and mv terms using Boost Filesystem.

Boost Filesystem

This library simplifies working with files and directories by providing a class named boost::filesystem::path (central class) that allows to process paths. We have used available functions in this library to create directories and to validate file. This class is actually a typedef for boost::filesystem::basic_path<std::string>. We have linked boost libraries to CMakeLists.txt file to perform the file system operations. Using find_package command we have loaded boost components and the include directories are added to the INCLUDE_DIRECTORIES directory property for the current CMakeLists file and also to each target in it. Finally Boost_LIBRARIES are linked to tlx using target_link_libraries.

```
#Finds and loads settings from an external project.
find_package(Boost 1.55.0 COMPONENTS filesystem system REQUIRED)
…
include_directories(${Boost_INCLUDE_DIRS})
..
target_link_libraries(tlx ${Boost_LIBRARIES})
```

Evaluating ls term

We have implemented ls command using Term* eval_ls(Ls* t) method. The path term is stored in boost::filesystem::path variable as string and using the operator boost::filesystem::directory_iterator list of file (of string type) are pushed into abc (Term_seq). Finally list of files List of paths is returned.

Evaluating mkdir term

We have implemented mkdir command using Term* eval_mkdir(Mkdir* t) method. The path term is stored in boost::filesystem::path variable as string and using the operator boost::filesystem::create_directory, directory is created. Finally Boolean value True or False is returned.

Evaluating rmdir term

We have implemented mkdir command using Term* eval_rmdir(Rmdir* t) method. The path term is stored in boost::filesystem::path variable as string and using the operator is_directory we are checking whether input path is directory or not. If the input path is directory then using boost::filesystem::remove_all function directory is deleted and finally Boolean value True is returned. If the input path is not directory then False is returned.

Evaluating cd term

We have implemented cd command using Term* eval_cd(Cd* t) method. The path term is stored in boost::filesystem::path variable as string and using the operator is_directory we are checking whether input path is directory or not. If the input path is directory then using chdir function directory is changed and boost::filesystem::current_path will display current path of working directory. Then Boolean value True or False is returned.

Evaluating mv term

We have implemented mv command using Term* eval_mv(Mv* t) method.The old and new path terms are stored in variables of type boost::filesystem::path and using the basic operator boost::filesystem::rename(p,p1) the directory is moved to new path. Then Boolean value True or False is returned.

Results:

Test results for ls, mkdir, rmdir, cd and mv are shown below.

Syntax: <ls pathname>
Input: ls “CMakeFiles”

```
== parsed ==
ls "CMakeFiles";

== elaborated ==
ls "CMakeFiles";

== output ==
== result ==
[CMakeFiles/Makefile.cmake, CMakeFiles/Makefile2, CMakeFiles/progress.marks, CMa
keFiles/cmake.check_cache, CMakeFiles/waffle.dir, CMakeFiles/CMakeTmp, CMakeFile
s/CMakeOutput.log, CMakeFiles/TargetDirectories.txt, CMakeFiles/2.8.12.2, CMakeF
iles/CMakeDirectoryInformation.cmake]
```

Figure 3: output for ls command

Syntax: <mkdir pathname>
Input: mkdir “dummy1”

```
== parsed ==
mkdir "dummy1";

== elaborated ==
mkdir "dummy1";

== output ==
== result ==
true
```

Figure 4: output for mkdir command

Syntax: <rmdir pathname>
Input: rmdir “dummy1”

```
== parsed ==
rmdir "dummy1";

== elaborated ==
rmdir "dummy1";

== output ==
== result ==
true
```

Figure 5: output for rmdir command

Syntax: <cd pathname>
Input: cd “CMakeFiles”

```
== parsed ==
cd "CMakeFiles";

== elaborated ==
cd "CMakeFiles";

== output ==
"/home/rishi/Testing_waffle/waffle-master/build/CMakeFiles"
== result ==
true
```

Figure 6: output for cd command

Syntax: <mv oldpath newpath>
Input: mv “dummy3” “dummy”

```
== parsed ==
mv "dummy3" "dummy";

== elaborated ==
mv "dummy3" "dummy";

== output ==
== result ==
true
```

Figure 7: output for mv command.

Conclusion

This language adds new features to the Waffle and we have successfully implemented shell commands to operate on folders by using the boost file system.

Future Work

We have implemented ls command to list the content of given directory, ls pathname and the results can be improved by introducing additional options like

- ls with no option (takes current directory as default)
- ls with option -l (list file or directory, size, modified date and time, file or folder name and owner of file and it’s permission)

We are also thinking to implement other kind of pipe and filter systems. Pipes are used to run multiple commands from same command line.
Syntax: command1 | command2
Examples: $ ls -l | wc -l
Output of the first ls command is given as input to second wc command which will print number of files in current directory.
Also it would be interesting to work on file system operations.

References

[1] “Boost File System”, http://www.boost.org/doc/libs/1_57_0/libs/filesystem/doc/index.htm- 2014
[2] “Types and Programming Languages”, Benjamin C.Pierce, 2002
[3] “Dr. Andrew Sutton repository”, https://github.com/asutton/waffle, 2014