BOUTALEB Elias Reda
A0128072L

CS 4215 Programming Language Implementation
Shell scripting mini-project using Camlp4

Over the last two weeks, I have been working on a domain-specific language allowing the use of shell scripting constructs within the OCaml language.
This has been done by writing a Camlp4 syntax extension, that converts a program using the new syntax into an OCaml abstract syntax tree, that is then transformed into expanded clear text OCaml code. That code is then compiled and executed.

A proper shell interface was considered, but time constraints made that feature unimplemented. Camlp4 was preferred over extension points, as the syntax one can create is more flexible, albeit limited by use of the OCaml lexer and tokens.
It is possible to implement a custom lexer with custom tokens for Camlp4.

Run the programs

The shell scripts used are in the *shell_scripts* folder.
Their OCaml translations are in the *ocaml_impl* folder.
The Camlp4 syntax extension with examples is in the *camlp4* folder.

Running *make* inside the *ocaml_script* folder will generate bytecode program for each example.

Running *make* inside the *camlp4* folder will generate a Camlp4 printer called *ed* outputing expanded equivalent OCaml code when a filename is given as an argument. (*./ed ws0.ml*)
There are 5 examples, with N = 0 to 4.

To see the generated code, *make wsN-code*
To execute it, *make run-wsN*

Syntax

```
cmd{echo "################"}     # built-in echo statement
files="*.ml"                     # define a mutable string
i=0                              # define a mutable integer value

array{wc}                        # define mutable array of strings
 wc{i}                           # array indexing

cmd{"wc -w temp"}                # execute command
cmd{"wc" ${files}}               # execute command with variable
cmd{"wc -w" ${files}
```

```
for (entry) in ("ls *.ml"): cmd{"wc" ${entry}}          # looping construct 1
for (i) in (0 9): cmd{"echo" wc{i}}

concat{ files = ${files} ${entry} }                     # strings can be concatenated

out{"cat" ${entry}}>"temp"                  # output of command can be redirected to a file
```

Issues and constraints

The for loop only accepts a single statement, not several. Being able to fuse str_items together would have been nice, but I did not know how to mess with the AST and Camlp4 API.
It is possible to grab several statements together using LIST1, but inserting them unquoted seems tricky to do.

The commands can accept only one variable. Again, inserting then unquoted while accessing them is hard.

There is no type checking. Only strings and integers are manipulated here.

Approach

Before building the syntax extension, some bash scripts were first expanded into their OCaml equivalents using utility functions. Then, for each script considered, I came up with an alternative way of representing shell syntax constructs.
For each new syntax construct, corresponding grammar entry and abstract representation were added, then the corresponding syntax node and its contents are defined with concrete syntax and expanded, caught expressions.