C Shell Project

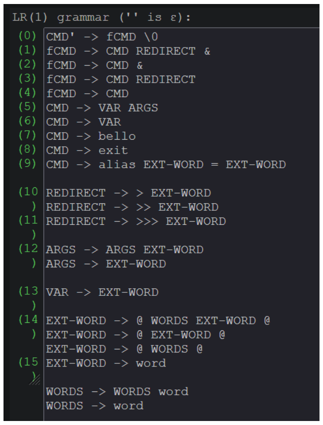
CMPE 322

Abdullah Umut Hamzaoğulları

13.12.2023

The language implied by the shell’s wanted rules is made sense of by explicitly writing out grammar rules, and parsing of any expression is achieved by implementing an LR(1) parser. Input strings are analyzed using regular expressions. Needed data structures like stacks are manually implemented.

Further information about this kind of a parser can be found on my Github, github.com/ultiminati because we had used this in CMPE 230 homeworks.



```
LR(1) grammar ('' is ε):
CMD' -> fCMD \0
fCMD -> CMD REDIRECT &
fCMD -> CMD &
fCMD -> CMD REDIRECT
fCMD -> CMD
CMD -> VAR ARGS
CMD -> VAR
CMD -> bello
CMD -> exit
CMD -> alias EXT-WORD = EXT-WORD

REDIRECT -> > EXT-WORD
REDIRECT -> >> EXT-WORD
REDIRECT -> >>> EXT-WORD

ARGS -> ARGS EXT-WORD
ARGS -> EXT-WORD

VAR -> EXT-WORD

EXT-WORD -> @ WORDS EXT-WORD @
EXT-WORD -> @ EXT-WORD @
EXT-WORD -> @ WORDS @
EXT-WORD -> word

WORDS -> WORDS word
WORDS -> word
```

I have written the rules for the wanted functionalities and turned into a finite state machine and stored the machine in parsingTable array.

The input has been tokenized with the usage of regex library and I have spent lots of time on this because of the late realization of non-standard regex, POSIX.

Then, I parsed and reduced the expressions following the rules of the grammar, and distinguished the different execution types. I have used fork and exec as wanted, implemented a findPath algorithm, and used execp afterwards. Background processes are implemented correctly with the usage of the NHANGUP flag. Redirection has been implemented with dup2 function. I spent too much time on the grammar side as I have an actual bug in the state generator website I have used. The >>> comment had not been implemented due to an unfortunate lack of time, but others hopefully should work fine. I have provided concise commands to the code, but to understand the theory of LR(1) parser, see my previous projects available in Github.