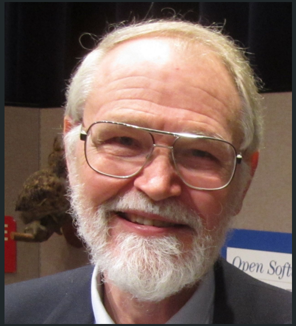
bawk



“bad awk”: a powerful text processing language

Ashley An, Christine Hsu,
Melanie Sawyer, Victoria Yang
PLT Fall 2018

Motivation

- Robust text processing language with intuitive C-like syntax
- Make it easy to analyze, read, and write to files
- Data-driven
- More verbose than awk
- Abstract away boilerplate code that repeatedly executes same actions over lines of a file
- Addition of mutable multidimensional arrays, easily mutable configuration variables

Tutorial – Run a bawk Program

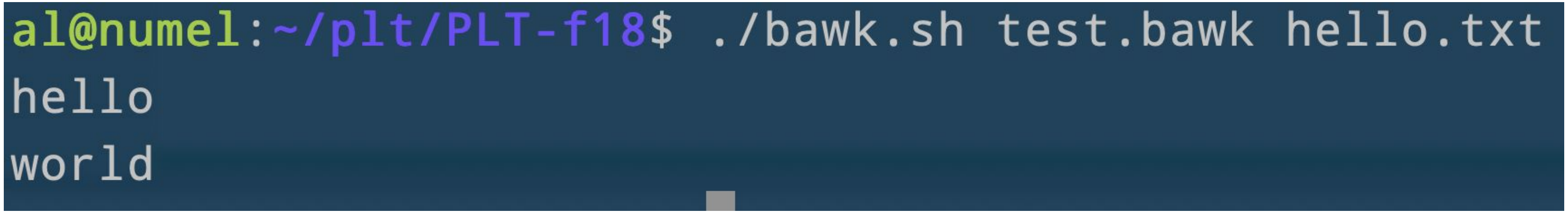
hello.bawk

```
BEGIN {}
LOOP {
    print($0);
}
END {}
```

input.txt

```
hello
world
```

```
./bawk.sh hello.bawk input.txt
./bawk.sh [.bawk file] [input file]
```



```
al@numel:~/plt/PLT-f18$ ./bawk.sh test.bawk hello.txt
hello
world
```

Tutorial – Program Structure

```
BEGIN {
        # function declarations and global variable declarations
}
LOOP {
        # loop over each line of a file; execute these statements for each
line
}
END {
        # execute these statements after we're done with the file
}
CONFIG {     # optional
        # set the field (word) separator & record (line) separator
}
```

Tutorial

Types

```
int a;
bool b;
string s;
rgx r;
string[] s_arr;
int[][][][][][] arr;
```

Operators

- field access ($)
- string concatenation (&)
- rgx, string, boolean comparison
- integer operations
- logical operations
- array access

Tutorial

Functions & Control Flow

```
int function (int a, int b) {
        while (a != b) {
                if (a > b) {
                        a = a - b;
                }
                else {
                        b = b - a;
                }
        }
        return a;
}
```

Control Flow

```
int i = 0;
arr = [1, 2, 3, 4, 5];

for ( i=0; i < 10; i++) {
        print(int_to_string(arr[i]));
}
```

- “**if**” statements do not require matching “else” blocks

Tutorial

Other Special Keywords

- NF – Number of Fields
- RS – Record Separator
- FS – Field Separator

Built-in Functions

- type conversion functions

  e.g. int_to_string
- array functions

  insert, delete, contains,
  length, index_of
- print
- nprint

Key Features – File Looping

```
LOOP {
        # everything in here is executed
        # once for each line of the file
}
```

- Continues looping until entire file is read through
- CONFIG block sets how the file will be looped through
  - Line separators are set with “RS”
  - Field separators are set with “FS”

Key Features – Field Access ($)

Access a specified field of a line

Set in CONFIG block:

- FS = Field Separator
    - FS = “,”
- RS = Record Separator
    - RS = “\r\n”

Sample Line: Another layer of indirection

```
print($0):
    >> Another layer of indirection
print($1):
    >> Another
print($2):
    >> layer
```

Key Features – Infinitely nested mutable arrays

```
int [][][] m;

m = [ [ [1, 2], [3, 4] ],  [ [5, 6], [7, 8] ] ];

m[0][0][0] = 0;                            # m = [ [ [0, 2], [3, 4] ],  [ [5, 6], [7, 8] ] ];

delete(m, 1);                              # m = [ [ [0, 2], [3, 4] ] ]

insert(m, 1, [ [9, 10], [11, 12] ] );      # m= [ [ [0, 2], [3, 4] ], [ [9, 10], [11, 12] ] ];
```

Key Features – Regex

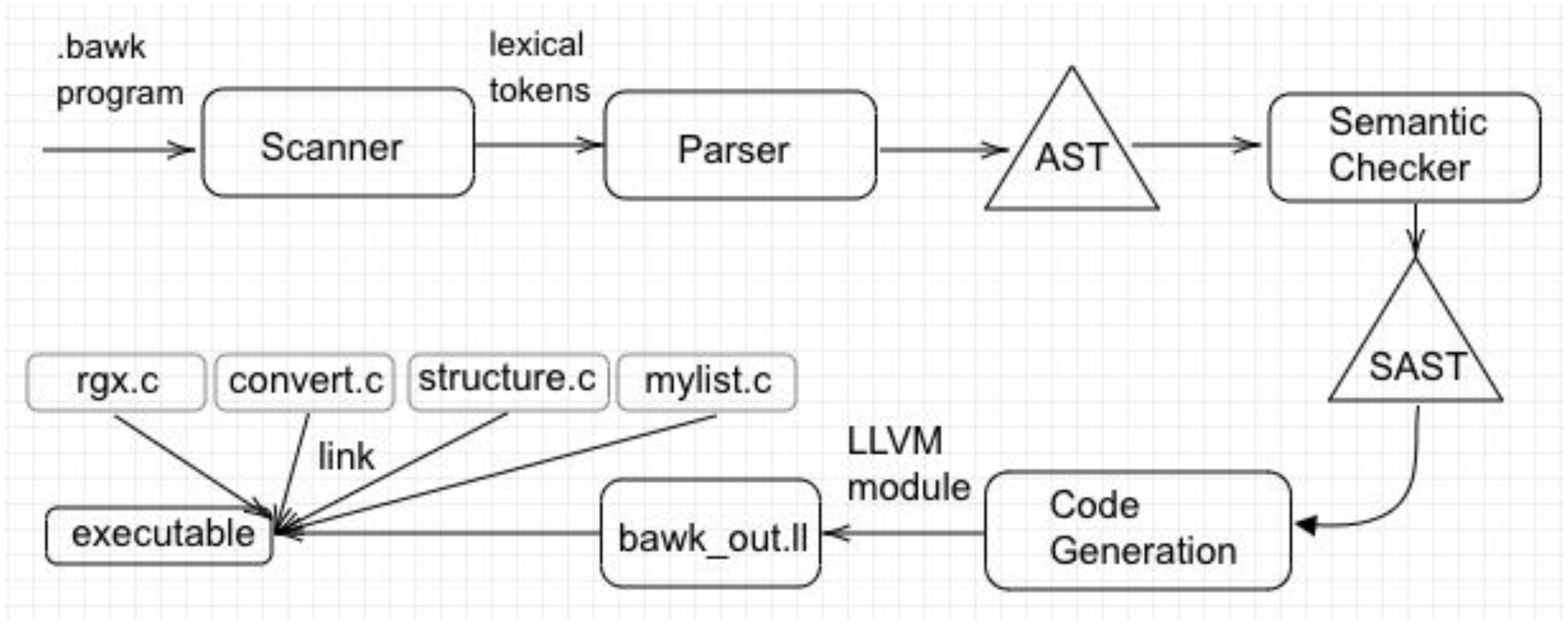
- POSIX regex pattern matching with wrapper functions
- Allows text filtering and expression comparisons

```
pattern = ‘i .[a-zA-Z]* plt’;

if (feeling ~ pattern) {

    print(feeling);
}
```

would match on “I love plt”, “I hate plt”, “I despise plt”, “I fear plt”, “I enjoy plt”

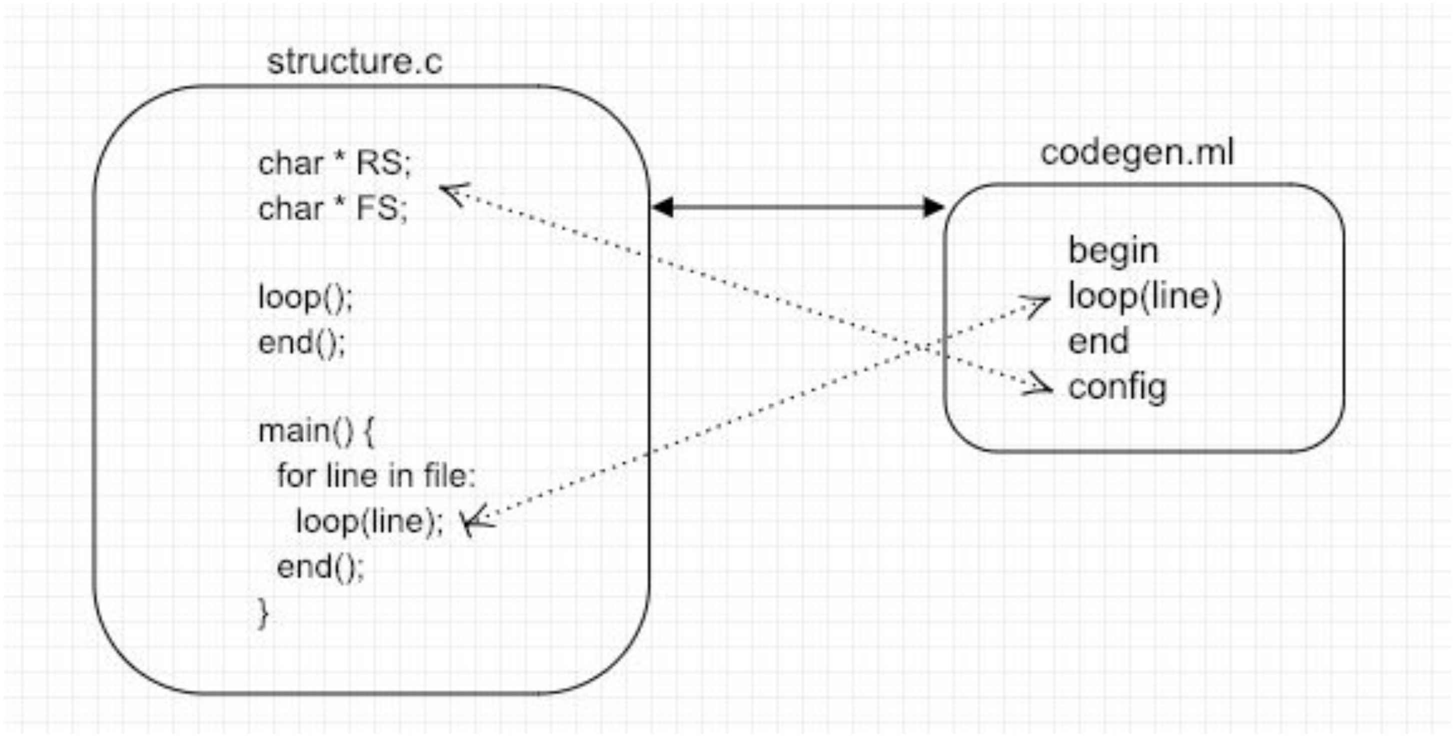
would not match on “I plt”, “I do not love plt”

System Architecture



- C libraries implement arrays, built-in conversion functions, regex, and main function

System Architecture



structure.c

```
char * RS;
char * FS;

loop();
end();

main() {
  for line in file:
    loop(line);
  end();
}
```

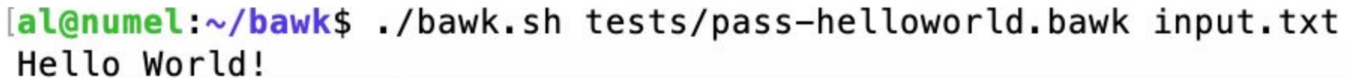
codegen.ml

```
begin
loop(line)
end
config
```

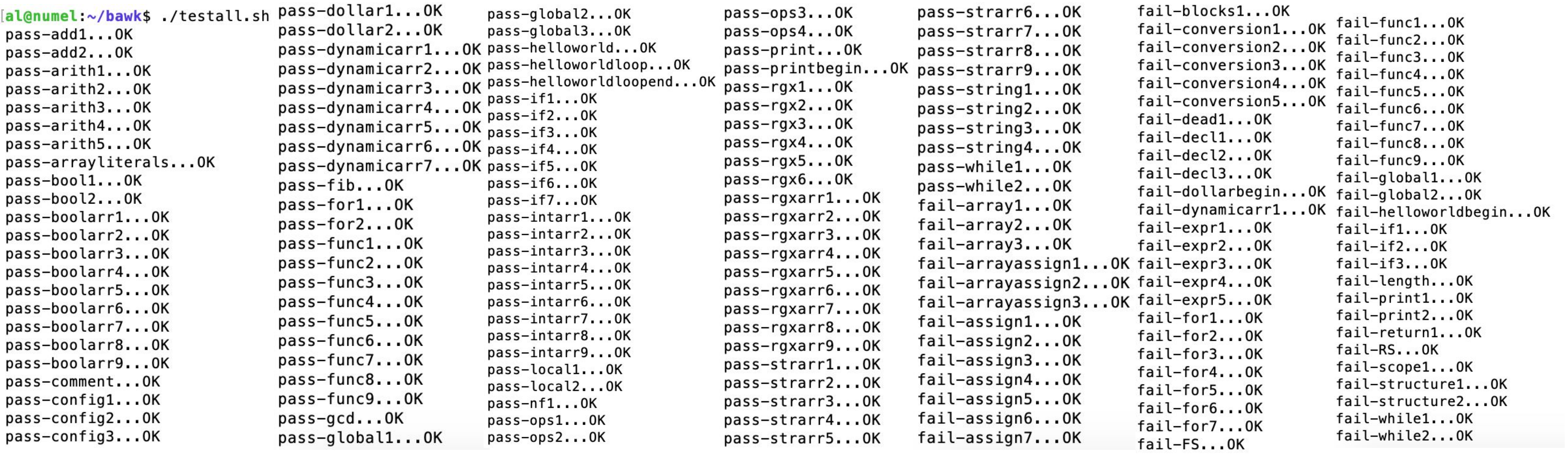
Testing

- Pass and fail tests for each stage of development
    - Lexer, parser, semantic checking, code generation
- Aim to pinpoint every feature of our language
- Check that the correct output / error messages are being generated
- Range from small tests (ex: basic operations) to larger tests (ex: file reading)
- Use bawk.sh [./bawk file] [input file] to run single test
- Use testall.sh to run all tests -> to automate running over 150 tests

Testing



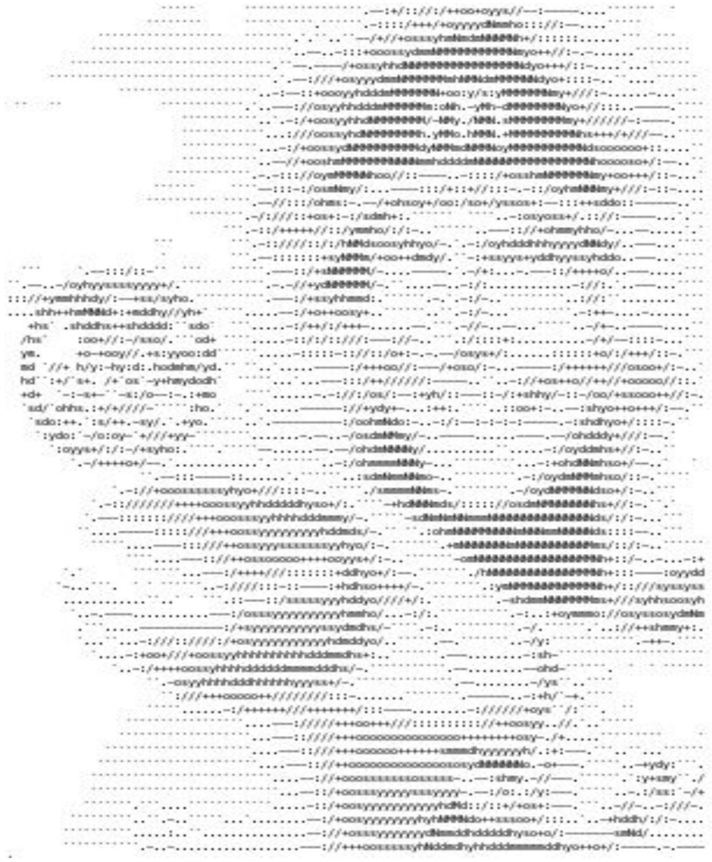
```
[al@numel:~/bawk$ ./bawk.sh tests/pass-helloworld.bawk input.txt
Hello World!
```



```
[al@numel:~/bawk$ ./testall.sh
pass-add1...OK
pass-add2...OK
pass-arith1...OK
pass-arith2...OK
pass-arith3...OK
pass-arith4...OK
pass-arith5...OK
pass-arrayliterals...OK
pass-bool1...OK
pass-bool2...OK
pass-boolarr1...OK
pass-boolarr2...OK
pass-boolarr3...OK
pass-boolarr4...OK
pass-boolarr5...OK
pass-boolarr6...OK
pass-boolarr7...OK
pass-boolarr8...OK
pass-boolarr9...OK
pass-comment...OK
pass-config1...OK
pass-config2...OK
pass-config3...OK
pass-dollar1...OK
pass-dollar2...OK
pass-dynamicarr1...OK
pass-dynamicarr2...OK
pass-dynamicarr3...OK
pass-dynamicarr4...OK
pass-dynamicarr5...OK
pass-dynamicarr6...OK
pass-dynamicarr7...OK
pass-fib...OK
pass-for1...OK
pass-for2...OK
pass-func1...OK
pass-func2...OK
pass-func3...OK
pass-func4...OK
pass-func5...OK
pass-func6...OK
pass-func7...OK
pass-func8...OK
pass-func9...OK
pass-gcd...OK
pass-global1...OK
pass-global2...OK
pass-global3...OK
pass-helloworld...OK
pass-helloworldloop...OK
pass-helloworldloopend...OK
pass-if1...OK
pass-if2...OK
pass-if3...OK
pass-if4...OK
pass-if5...OK
pass-if6...OK
pass-if7...OK
pass-intarr1...OK
pass-intarr2...OK
pass-intarr3...OK
pass-intarr4...OK
pass-intarr5...OK
pass-intarr6...OK
pass-intarr7...OK
pass-intarr8...OK
pass-intarr9...OK
pass-local1...OK
pass-local2...OK
pass-nf1...OK
pass-ops1...OK
pass-ops2...OK
pass-ops3...OK
pass-ops4...OK
pass-print...OK
pass-printbegin...OK
pass-rgx1...OK
pass-rgx2...OK
pass-rgx3...OK
pass-rgx4...OK
pass-rgx5...OK
pass-rgx6...OK
pass-rgxarr1...OK
pass-rgxarr2...OK
pass-rgxarr3...OK
pass-rgxarr4...OK
pass-rgxarr5...OK
pass-rgxarr6...OK
pass-rgxarr7...OK
pass-rgxarr8...OK
pass-rgxarr9...OK
pass-strarr1...OK
pass-strarr2...OK
pass-strarr3...OK
pass-strarr4...OK
pass-strarr5...OK
pass-strarr6...OK
pass-strarr7...OK
pass-strarr8...OK
pass-strarr9...OK
pass-string1...OK
pass-string2...OK
pass-string3...OK
pass-string4...OK
pass-while1...OK
pass-while2...OK
fail-array1...OK
fail-array2...OK
fail-array3...OK
fail-arrayassign1...OK
fail-arrayassign2...OK
fail-arrayassign3...OK
fail-assign1...OK
fail-assign2...OK
fail-assign3...OK
fail-assign4...OK
fail-assign5...OK
fail-assign6...OK
fail-assign7...OK
fail-blocks1...OK
fail-conversion1...OK
fail-conversion2...OK
fail-conversion3...OK
fail-conversion4...OK
fail-conversion5...OK
fail-dead1...OK
fail-decl1...OK
fail-decl2...OK
fail-decl3...OK
fail-dollarbegin...OK
fail-dynamicarr1...OK
fail-expr1...OK
fail-expr2...OK
fail-expr3...OK
fail-expr4...OK
fail-expr5...OK
fail-for1...OK
fail-for2...OK
fail-for3...OK
fail-for4...OK
fail-for5...OK
fail-for6...OK
fail-for7...OK
fail-FS...OK
fail-func1...OK
fail-func2...OK
fail-func3...OK
fail-func4...OK
fail-func5...OK
fail-func6...OK
fail-func7...OK
fail-func8...OK
fail-func9...OK
fail-global1...OK
fail-global2...OK
fail-helloworldbegin...OK
fail-if1...OK
fail-if2...OK
fail-if3...OK
fail-length...OK
fail-print1...OK
fail-print2...OK
fail-return1...OK
fail-RS...OK
fail-scope1...OK
fail-structure1...OK
fail-structure2...OK
fail-while1...OK
fail-while2...OK
```

Demo

```
./bawk.sh demo/demo.bawk demo/shuffled.txt
```



39元