BAWK, A BINARY AWK

Kevin M. Graney
kmg2165@columbia.edu

Programming Languages and Translators
COMS W4115, Spring 2013, Professor Stephen Edwards
## Contents

<table>
<thead>
<tr>
<th>1 Introduction</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Overview</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Sample program</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 Language tutorial</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 A sample program</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Executing bawk files</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Walking through the example</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 Language reference manual</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Lexical Conventions</td>
<td>8</td>
</tr>
<tr>
<td>3.1.1 Comments</td>
<td>8</td>
</tr>
<tr>
<td>3.1.2 Identifiers</td>
<td>8</td>
</tr>
<tr>
<td>3.1.3 Keywords</td>
<td>8</td>
</tr>
<tr>
<td>3.1.4 Constants</td>
<td>8</td>
</tr>
<tr>
<td>3.1.5 String Literals</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Meaning of Identifiers</td>
<td>9</td>
</tr>
<tr>
<td>3.2.1 Function Name</td>
<td>9</td>
</tr>
<tr>
<td>3.2.2 Variable Name</td>
<td>9</td>
</tr>
<tr>
<td>3.2.3 Pattern Binding Variable Name</td>
<td>9</td>
</tr>
<tr>
<td>3.2.4 Special Identifiers</td>
<td>9</td>
</tr>
<tr>
<td>3.3 Statements</td>
<td>9</td>
</tr>
<tr>
<td>3.3.1 Block Statements</td>
<td>10</td>
</tr>
<tr>
<td>3.3.2 Pattern Statements</td>
<td>10</td>
</tr>
<tr>
<td>3.3.3 Expression Statements</td>
<td>10</td>
</tr>
<tr>
<td>3.3.4 Function Declaration</td>
<td>10</td>
</tr>
<tr>
<td>3.3.5 If statement</td>
<td>10</td>
</tr>
<tr>
<td>3.3.6 If/else statement</td>
<td>11</td>
</tr>
<tr>
<td>3.4 Expressions</td>
<td>11</td>
</tr>
<tr>
<td>3.4.1 Primary Expression</td>
<td>11</td>
</tr>
<tr>
<td>3.4.2 Function Calls</td>
<td>11</td>
</tr>
<tr>
<td>3.4.3 Multiplicative Operators</td>
<td>11</td>
</tr>
<tr>
<td>3.4.4 Additive Operators</td>
<td>12</td>
</tr>
<tr>
<td>3.4.5 Relational Operators</td>
<td>12</td>
</tr>
<tr>
<td>3.4.6 Equality Operators</td>
<td>12</td>
</tr>
</tbody>
</table>
3.4.7 Assignment Expressions .................................................. 12
3.5 Pattern Expressions .......................................................... 12
  3.5.1 Constants ...................................................................... 13
  3.5.2 Bindings ........................................................................ 13
  3.5.3 Pattern Variables .......................................................... 14
  3.5.4 String Literals ............................................................... 14

4 Project plan ................................................................. 15
  4.1 Project log ........................................................................ 16

5 Architectural design .................................................. 48
  5.1 Design decisions ............................................................... 48
  5.2 Execution phases .............................................................. 48
    5.2.1 Bytecode compiler ...................................................... 48
    5.2.2 Bytecode operations .................................................... 49
    5.2.3 Bytecode runtime ......................................................... 50
  5.3 Limitations ........................................................................ 50

6 Test plan ........................................................................ 51

7 Lessons learned ................................................................. 52
  7.1 Language design ............................................................... 52
  7.2 Scanner and parser generators ............................................ 52
    7.2.1 GraphViz output .......................................................... 52
  7.3 Compiler and bytecode design ............................................. 53
    7.3.1 Return statement problem .............................................. 53
    7.3.2 Lack of full support for string types ................................. 53

8 Source code listing ............................................................... 54
  8.1 Compiler and bytecode interpreter ...................................... 54
    8.1.1 Scanner & parser definitions ........................................... 54
    8.1.2 Interfaces .................................................................... 58
    8.1.3 Source Files ................................................................. 63
  8.2 Test files ........................................................................... 77
    8.2.1 test-arith.bawk ............................................................. 77
    8.2.2 test-cond1.bawk ............................................................ 77
    8.2.3 test-func1.bawk ............................................................ 78
    8.2.4 test-func2.bawk ............................................................ 78
    8.2.5 test-func3.bawk ............................................................ 78
    8.2.6 test-pat1.bawk ............................................................... 79
    8.2.7 test-pat2.bawk ............................................................... 79
    8.2.8 test-pat3.bawk ............................................................... 80
    8.2.9 test-pat4.bawk ............................................................... 80
    8.2.10 test-pat5.bawk .............................................................. 81
    8.2.11 test-pat6.bawk .............................................................. 81
    8.2.12 test-pat7.bawk .............................................................. 82
    8.2.13 test-pat8.bawk .............................................................. 82
Listings

1.1 A simple bawk program .................................................. 3
2.1 Showing how RP works .................................................. 6
4.1 Git log for project development ....................................... 16
5.1 Possibly ambiguous code ............................................... 48
8.1 scanner.ml ................................................................. 54
8.2 parser.mly ............................................................... 56
8.3 parser_help.ml ........................................................... 58
8.4 ast_types.ml ............................................................. 59
8.5 bytecode_types.ml ..................................................... 59
8.6 ast.ml ........................................................................ 60
8.7 compile.ml ................................................................. 61
8.8 bytecode.ml ............................................................... 62
8.9 reader.ml ................................................................. 62
8.10 utile.ml ................................................................. 63
8.11 bawk.ml ................................................................. 63
8.12 parser_help.ml ............................................................ 64
8.13 ast.ml ................................................................. 65
8.14 compile.ml ............................................................... 68
8.15 bytecode.ml ............................................................. 72
8.16 reader.ml ............................................................... 75
8.17 utile.ml ................................................................. 76
8.18 Test program (test-arith.bawk) ....................................... 77
8.19 Expected output (test-arith.bawk.out) ............................ 77
8.20 Test program (test-cond1.bawk) ..................................... 77
8.21 Expected output (test-cond1.bawk.out) .......................... 77
8.22 Test program (test-func1.bawk) .................................... 78
8.23 Expected output (test-func1.bawk.out) .......................... 78
8.24 Test program (test-func2.bawk) .................................... 78
8.25 Expected output (test-func2.bawk.out) .......................... 78
8.26 Test program (test-func3.bawk) .................................... 78
8.27 Expected output (test-func3.bawk.out) .......................... 79
8.28 Test program (test-pat1.bawk) ...................................... 79
8.29 Expected output (test-pat1.bawk.out) ............................ 79
8.30 Test program (test-pat2.bawk) ...................................... 79
8.31 Expected output (test-pat2.bawk.out) ............................ 80
Chapter 1

Introduction

1.1 Overview

The bawk language, whose name is derived from ‘binary awk’, is intended to be a small, special-purpose language for the parsing of formatted binary files. In the spirit of awk, bawk will match rules in binary files and extract data based on these rules. Bawk will solve the problem of quickly decoding a binary file of a known format to extract information much the way awk solves this same problem for formatted text files.

The bawk interpreter will take two inputs for execution: (1) text in the bawk language, and (2) binary data to run the bawk program on. This workflow will be similar to that of a basic awk workflow, and the language itself is inspired by awk, with the same basic program construct. That is, a program will be a series of patterns and statements, with the statements executing when their respective pattern matches.

1.2 Sample program

A representative bawk program is shown below. This program operates on PNG format image files and will print two numbers, newline separated. The first number is the width of the image and the second number is the height.

Listing 1.1: A simple bawk program

```bash
/* print the size of a PNG file */

/*89 "PNG" 0d 0a 1a 0a/ {
  /*length:uint4 "IHDR"*/ {
    /width:uint4 height:uint4/ {
      print(width);
      print(height);
    }
  }
}
```
Chapter 2

Language tutorial

The bawk language is based on pattern matching bytes and extracting values from binary format data files.

2.1 A sample program

The sample program shown in §1.2 illustrates how to print the width and height of a given PNG image file using bawk. This is a perfect example of an application of the bawk language: a simple binary format file for which the extraction of integer data is desired. We will use this sample program in the next section to show how bawk is executed.

2.2 Executing bawk files

To execute a bawk file, call bawk with an -e flag and specify the file on which to operate. The bawk code is then to be provided via standard input.

./bawk -e [binary filename] < [bawk filename]

This command will compile and run the bawk program over the given binary file. The -e flag is optional and may be omitted. (The default operation is to execute the program if no flags are given.)

To execute the sample program, named tests/test-util1.bawk, we run bawk on a sample file, tests/lichtenstein.png, and see that the height and width of this file are both 512 pixels.

$ ./bawk -e tests/lichtenstein.png < tests/test-useful1.bawk
512
512

To see the bytecode instructions for a given program use instead the -c flag. With this flag, the bytecode instructions will be printed to standard output.

./bawk -c < [bawk filename]

If we compile our sample program for PNG files, we can see the bytecode generated, which is shown below.
89: Bin Sub 108: Skp
90: Bne 111 109: Skp
91: Ldp 110: Bra 116
92: Ldp 111: Skp
93: Beo 108 112: Beo 116
94: Rdb 4 113: Rdb 1
95: Str 3 114: Drp
96: Rdb 4 115: Bra 71
97: Str 4 116: Skp
98: Lod 3 117: Skp
99: Jsr -1 118: Bra 124
100: Lod 4 119: Skp
101: Jsr -1 120: Beo 124
102: Bra 108 121: Rdb 1
103: Skp 122: Drp
104: Beo 108 123: Bra 1
105: Rdb 1 124: Skp
106: Drp 125: Skp
107: Bra 92 126: Hlt

2.3 Walking through the example

The first line of the sample program in §1.2 matches the pattern 89 "PNG" 0d0a1a0a, where the string "PNG" is translated into ASCII bytes 50 4e 47. Bawk will scan through the input file until finding this pattern. On the first occurrence, the block statement following will be executed, and the nested patterns will attempt to match starting at the end of the match from the higher-level pattern. When a block exits, the pattern matching returns to executing at the position where it last began searching.

There is a special variable in bawk, RP, that can be used to retrieve or set the read pointer over the binary file. The value of RP represents the number of bytes from the beginning of the file. If we modify the example program to show us RP at various execution points it should become clear how the variable behaves.

Listing 2.1: Showing how RP works

```c
1  print(RP);
2  /89 "PNG" 0d 0a 1a 0a/ {           
3    print(RP);
4    /length:uint4 "IHDR"/ {        
5      print(RP);
6      /width:uint4 height:uint4/ {
7         print(RP);
8      }
9    print(RP);
10  }
11  print(RP);
12 }
13 print(RP);
14 RP + 100;
```
15 print(RP);
16 /fee/ { print(RP); }
17 print(RP);

Running this code produces the following output on lichtenstein.png.

0
8
16
24
16
8
0
0
15705
0

As you can see, the RP pointer resets at the end of each pattern matching block, and can also be manually advanced. Pattern matching always searches forward from RP.

In addition to pattern matching bawk supports dynamic variable scoping, functions, conditional statements, and mixed endianness. See the language reference manual in Chapter 3 for details.
Chapter 3

Language reference manual

3.1 Lexical Conventions

A program consists of a single bawk language character file. There are six classes of tokens: identifiers, keywords, constants, string literals, operators, and separators. White space characters are ignored except as they separate tokens or when they appear in string literals.

3.1.1 Comments

Comments follow the ANSI C style, beginning with /* and ending with */. They do not nest, and they cannot be present in quoted strings.

3.1.2 Identifiers

An identifier is a sequence of uppercase and lowercase letters A–Z, the numerals 0–9, and the underscore character. The first character of an identifier can not be a numeral.

\[
\text{identifier} ::= \text{letter} \\
\text{identifier} (\text{letter} | \text{digit})
\]

3.1.3 Keywords

The following identifiers are reserved for use as keywords, and should not be used otherwise: if, else, return, def, int1, int2, int4, uint1, uint2, uint4.

3.1.4 Constants

Bawk only supports a single kind of constant: the integer constant. The semantic meaning of integer constants depends on the context in which they occur. Except for inside of pattern expressions (see §3.5.1), the following is true. An integer constant consisting of a sequence of digits is taken to be in decimal. A sequence of hexadecimal digits prefaced by 0x is taken to be in hexadecimal (base 16).

3.1.5 String Literals

A string is a sequence of characters enclosed in double quotes. The only valid place for strings is inside of pattern expressions (see §3.5.4).
3.2 Meaning of Identifiers

Identifiers can have several different meanings based on how they are first used. Once an identifier is associated with a particular meaning it cannot be disassociated with that meaning.

3.2.1 Function Name

Previously unused identifiers become function names once a function is declared with a given name.

3.2.2 Variable Name

Previously unused identifiers become variable names once an assignment expression (see §3.4.7) executes with the new identifier on the left hand side. Variables have scope where they are first created in this way. Scoping is dynamic.

3.2.3 Pattern Binding Variable Name

Previously unused identifiers become binded variable names once a pattern expression with a binding pattern token (see §3.5.2) matches.

3.2.4 Special Identifiers

There are certain special identifiers that behave like variables, but whose meaning affects the program execution.

- RP – The file pointer variable indicates where in the file pattern statements begin matching (see §3.3.2). The value contains the number of bytes from the beginning of the file. At the start of the bawk program this value defaults to 0.

- LE – Force little-endian interpretation of integers and strings while pattern matching in the file. A value of 0 indicates big-endian interpretation and a value of 1 indicates little-endian interpretation. The default value is 0.

3.3 Statements

The expressions described in §3.4 are a specific form of statement in the bawk language. There are three different types of statements, and a list of statements forms a bawk program. The three different types of statements are: expressions (described in §3.4), block statements, and pattern statements.

\[
\text{〈program〉 ::= 〈statement-list〉}
\]

\[
\text{〈statement-list〉 ::= \epsilon}
\]

\[
\begin{align*}
\text{\&} & \text{〈statement-list〉 ::= \epsilon} \\
\text{\&} & \text{〈statement-list〉 ::= 〈statement〉} \\
\text{\&} & \text{〈statement-list〉 (statement)}
\end{align*}
\]
\( \langle \text{statement} \rangle \) \( ::= \langle \text{expression-statement} \rangle \) \\
| \( \langle \text{block-statement} \rangle \) \\
| \( \langle \text{pattern-statement} \rangle \) \\
| \( \langle \text{function-declaration} \rangle \) \\

### 3.3.1 Block Statements

Block statements are used to combine a sequence of statements into a single statement. The sequence of statements inside the block is executed in order when the block itself is executed. Variables first used inside the block are locally scoped, but names from outside the block are also available.

\( \langle \text{block-statement} \rangle \) \( ::= \{ \langle \text{statement-list} \rangle \} \)

### 3.3.2 Pattern Statements

Pattern statements allow a type of pattern matching to be performed on the binary file. Pattern statements consist of a pattern expression (see §3.5) and a statement. If the pattern expression matches the data file at the location of RP at the time of evaluation of the pattern statement, then the corresponding statement is executed. If the pattern does not match, then the statement is not executed.

\( \langle \text{pattern-statement} \rangle \) \( ::= / \langle \text{pattern-expression} \rangle / \langle \text{statement} \rangle \)

### 3.3.3 Expression Statements

An expression statement is an expression followed by a semicolon. Expressions are described in §3.4.

\( \langle \text{expression-statement} \rangle \) \( ::= \langle \text{expression} \rangle ; \)

### 3.3.4 Function Declaration

A function declaration is a statement declaring a new function. Function declarations consist of an identifier to serve as the function name, followed by an open parenthesis, an optional comma separated list of identifiers to serve as function parameter names, a closing parenthesis, and a statement to serve as the function body.

\( \langle \text{function-declaration} \rangle \) \( ::= \text{def} \langle \text{identifier} \rangle (\langle \text{function-decl-params} \rangle) \langle \text{statement} \rangle \)

\( \langle \text{function-decl-params} \rangle \) \( ::= \epsilon \) \\
| \( \langle \text{identifier} \rangle \) \\
| \( \langle \text{function-decl-params} \rangle , \langle \text{identifier} \rangle \)

### 3.3.5 If statement

If the expression provided to the if statement evaluates to a non-zero value then the subsequent statement is executed. If the expression evaluates to a value of zero, then the statement is not executed.

\( \langle \text{if-statement} \rangle \) \( ::= \text{if} (\langle \text{expression} \rangle) \langle \text{statement} \rangle \)
3.3.6 If/else statement

If/else statements work like if statements, with an additional second statement. If the expression evaluates to a non-zero value then the first statement is executed and the second is not. If the expression evaluates to zero then the first statement is not executed and the second is executed.

\[
\text{if-statement} ::= \text{if} \left( \text{expression} \right) \text{'else'} \left( \text{statement} \right)
\]

3.4 Expressions

3.4.1 Primary Expression

\[
\text{primary-expression} ::= \text{constant} \\
| \text{identifier} \\
| \text{string-literal} \\
| \text{function-call} \\
| \left( \text{expression} \right)
\]

\[
\text{expression} ::= \text{assignment-expression} \\
| \text{additive-expression}
\]

3.4.2 Function Calls

A function call is an identifier, known as the function name, followed by a pair of parenthesis containing a, possibly empty, comma-separated list of expressions. These expressions constitute arguments to the function. When functions are called a copy of each argument is made and used within the function, that is, all function calls pass arguments by value.

\[
\text{function-call} ::= \text{identifier} \left( \text{function-call-params} \right)
\]

\[
\text{function-call-params} ::= \epsilon \\
| \text{expression} \\
| \text{function-call-params}, \text{expression}
\]

3.4.3 Multiplicative Operators

The multiplicative operators, * and /, are left associative. The result of * is multiplication of the two operands, and the result of / is the quotient of the two operands. The quotient with a second operator of zero yields an undefined result.

\[
\text{multiplicative-expression} ::= \text{primary-expression} \\
| \text{multiplicative-expression} * \text{primary-expression} \\
| \text{multiplicative-expression} / \text{primary-expression}
\]
3.4.4 Additive Operators

The two additive operators, + and −, are left associative. The expected arithmetic operation is performed on integers. The result of the + operator is the sum of the operands, and the result of the − operator is the difference of the operands.

\[
\langle \text{additive-expression} \rangle ::= \langle \text{multiplicative-expression} \rangle \\
| \langle \text{additive-expression} \rangle + \langle \text{multiplicative-expression} \rangle \\
| \langle \text{additive-expression} \rangle - \langle \text{multiplicative-expression} \rangle
\]

3.4.5 Relational Operators

The relational operators are all left associative. The operators all produce 0 if the specified relation is false and 1 if the specified relation is true. The operators are < (less than), > (greater than), ≤ (less than or equal to), ≥ (greater than or equal to).

\[
\langle \text{relational-expression} \rangle ::= \langle \text{additive-expression} \rangle \\
| \langle \text{relational-expression} \rangle < \langle \text{additive-expression} \rangle \\
| \langle \text{relational-expression} \rangle > \langle \text{additive-expression} \rangle \\
| \langle \text{relational-expression} \rangle \leq \langle \text{additive-expression} \rangle \\
| \langle \text{relational-expression} \rangle \geq \langle \text{additive-expression} \rangle
\]

3.4.6 Equality Operators

The == (equal to) and != (not equal to) operators compare operands for equality. Like the relational operators, the equality operators produce 0 if the relationship is false and 1 if the relationship is true. The equality operators have a lower precedence than the relational operators, and are also left associative.

\[
\langle \text{equality-expression} \rangle ::= \langle \text{relational-expression} \rangle \\
| \langle \text{equality-expression} \rangle == \langle \text{relational-expression} \rangle \\
| \langle \text{equality-expression} \rangle != \langle \text{relational-expression} \rangle
\]

3.4.7 Assignment Expressions

All assignment expressions require a modifiable identifier on the left hand side. The only type of modifiable identifier is a variable (see §3.2.2). The assignment operator, =, is right-associative and the return value of assignment is the value being assigned.

\[
\langle \text{assignment-expression} \rangle ::= \langle \text{equality-expression} \rangle \\
| \langle \text{identifier} \rangle = \langle \text{assignment-expression} \rangle
\]

3.5 Pattern Expressions

A pattern expression matches content in the binary data file. Pattern expressions are distinct in syntax from the expressions described in §3.4.
Pattern expressions can contain constants or typed variable bindings. Each constant or binding is known as a pattern term, and pattern terms are whitespace delimited. Already bound pattern variables can also be used in pattern statements.

### 3.5.1 Constants

Inside pattern expressions, constants work differently than those described in §3.1.4. All constants in pattern terms are implicitly expressed in hexadecimal form, and the 0x prefix required to express hexadecimal constants in §3.1.4 must be omitted. Constants are read in hexadecimal as bytes. For this reason leading zeros on a value have semantic meaning. For example /0000abcd/ matches the four byte pattern 00 00 ab cd in the binary file, while /abcd/ matches the two byte pattern ab cd.

The byte order expected in the file is dependent on the run-time value of the LE special variable (see §3.2.4). If this variable is non-zero, then constants are interpreted in little endian order. If the variable is zero, then the interpretation is big endian. The default value is zero. As an example, if LE != 0 the pattern /aabb/ matches the file with two bytes in the order bb aa, and if LE == 0 then the pattern matches bytes aa bb in that order.

Whitespace does make a difference in the interpretation of pattern constants. When LE != 0, the patterns /aabb/ and /aa bb/ are distinctly different. If LE == 0, these two patterns are the same. Whitespace added between byte boundaries also changes the semantic meaning; /01ab/ is not identical to /0 1 a b/. If a constant has an odd number of hexadecimal digits, an implied 0 is added to the left-most side of the value. That is /123/ is the same as /0123/ (is the same as /01 23/ if LE == 0). For this reason the example of /0 1 a b/ is identical to /00 01 0a 0b/ (and /00010a0b/ if LE == 0).

### 3.5.2 Bindings

A bind pattern term consists of an identifier, a colon (:), and a bind type. Valid bind types are: int1, int2, int4, uint2, uint4. The int1 type matches a one-byte integer, the uint4 type matches an unsigned 4-byte integer. Identifiers have the same naming rules as §3.1.2.

\[
\langle \text{pattern-binding} \rangle := \langle \text{identifier} \rangle \ ': ' \langle \text{bind-type} \rangle
\]

\[
\langle \text{bind-type} \rangle := \text{int1} \\
| \text{int2} \\
| \text{int4} \\
| \text{uint1}
\]
3.5.3 Pattern Variables

Previously bound pattern variables can be used in a pattern expression to match the previously bound value.

\[
\langle \text{pattern-variable} \rangle ::= \langle \text{identifier} \rangle
\]

3.5.4 String Literals

String literals are also supported in pattern expressions. The syntax is exactly the same as strings in §3.1.5. The string literal is converted to and interpreted as a sequence of ASCII-encoded bytes, exactly how constants are (see §3.5.1).
Chapter 4

Project plan

The plan for this project was to execute the following basic steps.

1. **Determine objectives of the language** – the language was designed to be useful for extracting data quickly from binary files. It was modelled after an awk-style syntax, but was constrained to be simple enough to implement in a single semester.

2. **Design the language syntax** – the language syntax was modelled after awk and the microc language, and the concept was tested in parallel with writing a parser for it.
   
   (a) **Write the scanner/parser** – This was critical to working through details of the language in parallel with designing it.
   
   (b) **Design the AST types** – The abstract syntax tree (AST) types were designed to be as simple as possible, so that the translation to bytecode could be done easily.

3. **Write the logic to output the AST to GraphViz** – This was done mostly to help debug the AST structure, but also to make sure the AST was easy enough to traverse before attempting the bytecode translation. Translating the AST to GraphViz code enabled the creation of a visualization that could be generated for different inputs to make sure very early on that things like operator precedence were working correctly.

4. **Design the bytecode** – The plan was to take the bytecode from microc and add some basic instructions specific to the reading of binary files, such as an instruction to read a given number of bytes, branch conditionally on reaching end-of-file, etc. The stack-style design of the microc bytecode made it easier to implement than a register-based bytecode. One change I did intend to make was switching from relative branching to absolute branching.

5. **Implement the translation logic** – The translation logic would hopefully be a straightforward step once the previous were completed. The plan was to implement translation from AST structures into basic blocks, and implement the necessary bytecode instructions in parallel, while testing along the way.

6. **Project report** – The project report would be completed for required intermediate deadlines, with the remainder written after finishing the bulk of the coding.
7. **Test** – Finally, a conclusive round of testing would be done to ensure the language has no glaring bugs. Inefficiency and minor non-fatal flaws would be tolerated.

Development tools used included git and GitHub for version control, Make for generating both the bawk executable as well as class documents, and a few decent text editors. GraphViz was used for visualizing the AST, and \LaTeX{} was used to generate all the documents.

### 4.1 Project log

The project log, listed as git commits, is shown below. The full git repository is included in the project archive file.

<table>
<thead>
<tr>
<th>Listing 4.1: Git log for project development</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 commit 9b2726855909c940c129ce02e3e81611238e5ed0</td>
</tr>
<tr>
<td>2 Author: Kevin Graney <a href="mailto:nanonet@gmail.com">nanonet@gmail.com</a></td>
</tr>
<tr>
<td>3 Date: Mon Aug 12 21:06:52 2013 -0400</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5 Report work</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7 Added the git log as a project log and finished up a few sections.</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9 Makefile</td>
</tr>
<tr>
<td>10 plt_docs/gitlog.tex</td>
</tr>
<tr>
<td>11 plt_docs/report-lessons.tex</td>
</tr>
<tr>
<td>12 plt_docs/report-plan.tex</td>
</tr>
<tr>
<td>13 plt_docs/report-test.tex</td>
</tr>
<tr>
<td>14 5 files changed, 1155 insertions(+), 4 deletions(-)</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16 commit 057a96571e6b99e2cfbf81126564acb0eb4095bd8</td>
</tr>
<tr>
<td>17 Author: Kevin Graney <a href="mailto:nanonet@gmail.com">nanonet@gmail.com</a></td>
</tr>
<tr>
<td>18 Date: Mon Aug 12 20:17:55 2013 -0400</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20 Report work</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22 plt_docs/report-architecture.tex</td>
</tr>
<tr>
<td>23 plt_docs/report-introduction.tex</td>
</tr>
<tr>
<td>24 plt_docs/report-lrm.tex</td>
</tr>
<tr>
<td>25 plt_docs/report-plan.tex</td>
</tr>
<tr>
<td>26 plt_docs/report-tutorial.tex</td>
</tr>
<tr>
<td>27 plt_docs/report.tex</td>
</tr>
<tr>
<td>28 6 files changed, 224 insertions(+), 7 deletions(-)</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>30 commit 5d13044f761abc93542db2260c05d2c4087c3950</td>
</tr>
<tr>
<td>31 Author: Kevin Graney <a href="mailto:nanonet@gmail.com">nanonet@gmail.com</a></td>
</tr>
<tr>
<td>32 Date: Sun Aug 11 17:38:11 2013 -0400</td>
</tr>
<tr>
<td>33</td>
</tr>
<tr>
<td>34 Remove old test case</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>36 tests/simple.bawk</td>
</tr>
<tr>
<td>37 1 file changed, 6 deletions(-)</td>
</tr>
<tr>
<td>38</td>
</tr>
<tr>
<td>39 commit 6e9314853eef20ed8d3a856b0f786f1ba68fa1e1</td>
</tr>
<tr>
<td>40 Author: Kevin Graney <a href="mailto:nanonet@gmail.com">nanonet@gmail.com</a></td>
</tr>
</tbody>
</table>
Date: Sun Aug 11 17:37:50 2013 -0400

Add test of int vs. uint types

tests/test-pat11.bawk | 8 ++++++++
tests/test-pat11.bawk.out | 2 ++
2 files changed, 10 insertions(+)  

commit 5c58672c9c786b60cc336fe221d89d72318847f1
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Aug 11 17:30:25 2013 -0400

Report work

plt_docs/report-lrm.tex | 5 +++--
plt_docs/report-plan.tex | 27 +++++++++++++++++++++++++---
2 files changed, 21 insertions(+), 11 deletions(-)

commit 820fc3560c2251d4421636ad13389da4666b1df3
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Aug 11 15:47:49 2013 -0400

Report work

plt_docs/report-architecture.tex | 5 +++--
plt_docs/report-lessons.tex | 8 +++++++--
2 files changed, 11 insertions(+), 2 deletions(-)

commit 9c4ec989cb27188b53c679b21e53b36b137b222
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Aug 11 15:47:38 2013 -0400

Line length enforcement for report

compile.ml | 3 ++--
reader.ml | 16 +++++++--
2 files changed, 11 insertions(+), 8 deletions(-)

commit 42d780a9e61394a6d229e5371b498a3c4bed9189
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Aug 11 14:56:07 2013 -0400

Work on final report

plt_docs/report-architecture.tex | 3 ++--
plt_docs/report-plan.tex | 10 +++++++--
plt_docs/report-test.tex | 27 +++++++++++++++++++++++++---
plt_docs/report.tex | 28 +-------------------
4 files changed, 44 insertions(+), 24 deletions(-)

commit 03f47bf08e889638f51df5ae5e62798dc5e715f1
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Aug 11 13:42:53 2013 -0400

Work on final report
Fail on bad use of string literals

String literals can only be used in pattern expressions.

Add Beq instruction to bytecode interpreter

Add if/else construct to AST GraphViz output

Add test case for LE and a string constant

Add test case for LE
commit eaa573f996315216274479668d5267b01e38eed
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Aug 10 13:02:29 2013 -0400

Add support for little endian constants

Pattern constants (of hexadecimal integer form) are now supported
when LE!=0 to be interpreted as a little endian value.

compile.ml | 13 ++++++++++++ 1 file changed, 12 insertions(+), 1 deletion(-)

commit 1f92e256bc1632bc000b2a509f8bab3169808a90
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Aug 10 13:01:38 2013 -0400

Add support for previously bound pattern variables

Previously bound pattern variables can be used in future pattern
matches.

compile.ml | 18 +++++++++++++++++ 4 files changed, 39 insertions(+), 1 deletion(-)

commit b03fc3b1ad8c2603175085bfbd8f14b7ec42b6d3
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Aug 10 12:12:40 2013 -0400

Test writing of RP

tests/test-pat7.bawk | 16 ++++++++++++ 2 files changed, 21 insertions(+)

tests/test-pat7.bawk.out | 5 ++++

commit 8e005718abc113fdd2105bddd9ffbc70016482ba5
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Aug 10 10:55:02 2013 -0400

Add LE as a default variable

compile.ml | 12 +++++++++++ 1 file changed, 8 insertions(+), 4 deletions(-)

commit d2c07179ed76266c6aea84b8ed313ed31d9cddd4
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Aug 10 10:43:14 2013 -0400

Support assignment to RP

bytecode.ml | 3 +++ 1 file changed, 3 insertions(+)

22
commit dd0b8d75c79828e90bb557b78738ae31e4b3f487
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Aug 10 10:38:10 2013 -0400

Add bind type names to reserved words

plt_docs/report-1rm.tex | 4 ++--
1 file changed, 2 insertions(+), 2 deletions(-)

commit b0f748857c35b232b3c0f7e6f3f1cd0021ba1c45
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Aug 10 10:37:01 2013 -0400

Change dynamic scoping of assignment

Changed the scoping of the assignment operator to be dynamic
and match the scoping of how variables are read.

compile.ml | 8 +++++---
tests/test-scopel.bawk | 2 +--
tests/test-scope2.bawk | 16 +++++++++++++++++
tests/test-scope2.bawk.out | 5 ++++
4 files changed, 27 insertions(+), 4 deletions(-)

commit 22c9279c64a1d417a2fcee8b6fc8086b65de1e49
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Aug 10 10:36:07 2013 -0400

Add test case of if/else constructs

tests/test-cond1.bawk | 15 +++++++++++++
tests/test-cond1.bawk.out | 2 ++
2 files changed, 17 insertions(+)

commit e6bd2027d36914828584a5d4cca14479d2d1397c
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Aug 8 19:47:44 2013 -0400

Change sample program to use uint types

1 file changed, 3 insertions(+), 5 deletions(-)

commit 49af898a754e376a4856817d9420b1e290345e1f
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Aug 8 19:51:14 2013 -0400

Create final report skeleton

Created a skeleton of the final report based on the syllabus.

plt_docs/report-introduction.tex | 8 ++
plt_docs/report-1rm.tex | 220 ++++++++++++++++++++++++++++++++++++++
plt_docs/report.tex | 52 ++++++
3 files changed, 280 insertions(+)

23
commit 87e3d252b4ed368753869e6c7ea161b4728ef23d
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Aug 8 19:42:39 2013 -0400

Add if/else statement support

This will be required for supporting runtime resolution of RP
easily, so it seems trivial to just implement the if/else construct
in the language too.

ast_types.mli |  1 +
compile.ml    | 18 +++++++++++++++
parser.mly    |  8 +++++
scanner.ml    |  3 ++
4 files changed, 23 insertions(+), 7 deletions(-)

commit 3cf57fd37d723fb04bd7235ee61fa58989cdd3a
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Aug 4 15:57:47 2013 -0400

Remove unused variable

scanner.ml |  2 +- 1 file changed, 1 insertion(+), 1 deletion(-)

commit 4eea7d725f07304ab6c4c5f231632b55769f0b72
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Aug 4 15:50:55 2013 -0400

Add test case for bind types

tests/test-pat6.bawk |  23 +++++++++++++++++++++++
tests/test-pat6.bawk.out |  12 +++++++
2 files changed, 35 insertions(+)

commit 905b922d5078080bd0575690d148f68b501cb70f
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Aug 4 15:49:14 2013 -0400

Add uint1 type

ast.ml |  3 ++
ast_types.mli |  1 +
scanner.ml |  1 +
3 files changed, 5 insertions(+)

commit 24dd85b1b31fe60fb58003cb96b62ce33edc87
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Aug 4 15:27:56 2013 -0400

Add support for differentiating int/uint types

Added compiler support for treating integer types as signed. This
assumes that the interpretation stored in the file is twos
complement.
Add instruction to convert to two's complement

Added a bytecode instruction to reinterpret the top of the stack as a two's complement value with the given bit length.

bytecode.ml |  10 +++++++-
bytecode_types.mli |  1 +
utile.ml |  6 +++++
utile.mli |  4 +++++
4 files changed, 20 insertions(+), 1 deletion(-)

commits

Add function to map binding types to their sizes

ast.ml |  7 +++++++
ast.mli |  4 +++++
2 files changed, 11 insertions(+)

Add pattern strings to AST GraphViz output

ast.ml |  7 +++++--
1 file changed, 5 insertions(+), 2 deletions(-)

Add LaTeX file for final report

.gitignore |  1 +
Makefile |  2 +-plt_docs/report.tex |  114 +++++++++++++++++++++++++++++++++++++++++++++++++++)
3 files changed, 116 insertions(+), 1 deletion(-)

Add test case for variable scoping

Show that variable scoping works inside of function calls
Add support for function calls with arguments

Added support for function call arguments. Values are placed on the stack at an offset from the frame pointer. The symbol table contains values $\leq -100$ to represent these values differently from global variables.

Add support for starting enumeration at a value

Added an optional argument to start the enumeration at a specific value. The default remains 0.

Make pattern matching work across entire file

Pattern matching now searches forward over the entire file instead of expecting a match at the current location of RP. This is implemented in an incredibly inefficient way, but this should be good enough for class.
Writing to this variable is not yet supported.

bytecode.ml | 3 +++
compile.ml | 2 +-
2 files changed, 4 insertions(+), 1 deletion(-)

commit b6edc2975e4ee271ba7d8e0622f830c61ffefd59
Author: Kevin Graney <nanonet@gmail.com>
Date: Fri Aug 2 20:51:32 2013 -0400

Added the Beo and Drp instructions to bytecode
Beo branches if the next byte reaches EOF and Drp drops a value from the stack.

bytecode.ml | 7 +++++
bytecode_types.mli | 1 +
compile.ml | 1 +
reader.ml | 17 ++++++-------
reader.mli | 4 +++-
5 files changed, 19 insertions(+), 11 deletions(-)

commit e794ccf368d43475081411e6b7b063aaa6274f1a
Author: Kevin Graney <nanonet@gmail.com>
Date: Fri Aug 2 19:41:42 2013 -0400

Add test for scoping of variables

tests/test-scope1.bawk | 15 ++++++++++++++++++++++
tests/test-scope1.bawk.out | 5 ++++
2 files changed, 20 insertions(+)

commit f5652c7d9d9ff21400cc8243a805da048308af8b7
Author: Kevin Graney <nanonet@gmail.com>
Date: Fri Aug 2 19:37:59 2013 -0400

Merge symbol map into a single StringMap

Removed the function_map and variable_map objects and created a single symbol_map StringMap to hold all the symbols. There still exists a bindings map to contain the size of the binding variables.

bawk.ml | 4 +++
compile.ml | 60 +++++++++++++++++++++++++++++-------------------------
compile.mli | 13 ++++++------
3 files changed, 36 insertions(+), 41 deletions(-)

commit d238b3e53cb35d6285251e98d202ff6fafa6450b
Author: Kevin Graney <nanonet@gmail.com>
Date: Fri Aug 2 18:34:12 2013 -0400

Add parent pointers to env

Added parent pointers to the environment records to support linked lists. Create a simple resolution function for function calls so things like print() (a built-in function) work from inside functions.
Add Beq to label resolution

commit 2fa4bd4f81bbe1e9f4980029be7f7d03bd3a5ae0
Author: Kevin Graney <nanonet@gmail.com>
Date: Fri Aug 2 18:57:03 2013 -0400

Add support for strings in pattern expressions

commit dd63a8c9d9a101245eadef753d7812bf3fcf67c7
Author: Kevin Graney <nanonet@gmail.com>
Date: Fri Aug 2 13:48:32 2013 -0400

Added support for string literals inside of pattern expressions.
This allows the first useful test case to be added for finding
the width and height of a PNG file.

commit fdad37b643f96444bb64620717901e649dd514d1
Author: Kevin Graney <nanonet@gmail.com>
Date: Fri Aug 2 13:30:19 2013 -0400

Added explode and implode functions to Utile

commit 34e7d8295fc9af5f18103305609d5f583f95d3b38
Author: Kevin Graney <nanonet@gmail.com>
Date: Fri Aug 2 13:29:14 2013 -0400

Add boolean operators to bytecode interpreter.

These may not be of limited use since we don't have conditionals,
but they should work.
Remove error-prone branch

No need to have Bne instruction for binding pattern match, since
this will cause it to reject cases where the variable binds
to a value of zero.

No need to have Bne instruction for binding pattern match, since
this will cause it to reject cases where the variable binds
to a value of zero.

Add test case for int4 bind type

Add test case for int1 pattern binding

Added pattern variables. The implementation is limited. Only int1
type works at the moment, the namespace for variables is shared
with the globals, and the storage of the data is still messy.
Added simple global variables. These are implemented as an array in the bytecode interpreter. The compiler assigns each name a unique entry in the array. Currently all variables are considered global. Variables are not being set at an offset of FP on the stack because they are dynamically created and not required to be declared.

```
bytecode.ml | 9 ++++++++  
compile.ml  | 33 +++++++++++++++++++++++++++++++++
compile.ml  | 7 +++++++
tests/test-var1.bawk | 5 ++++
tests/test-var1.bawk.out | 2 ++
5 files changed, 56 insertions(+)

commit 2ee86a40da8b5aed8e0e7844c1b4cbb7a3ef9d39
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jul 27 21:02:36 2013 -0400

Add basic function call test cases

```
tests/test-func1.bawk | 10 ++++++++  
tests/test-func1.bawk.out | 5 ++++
tests/test-func2.bawk | 7 +++++++
tests/test-func2.bawk.out | 2 ++
4 files changed, 24 insertions(+)  

commit d79330e9a40633c38a61a7760254a9a3756689f0
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jul 27 20:51:53 2013 -0400

Support function calls by name

```
bytecode.ml | 6 ++++
bytecode_types.mli | 2 --
compile.ml    | 15 ++++++++----
3 files changed, 18 insertions(+), 5 deletions(-)

commit bd94bb1eb0546ff2633e152df2e61759144d8ab
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jul 27 19:12:03 2013 -0400

Implement compilation of function declarations

```
bytecode.ml | 2 ++
bytecode_types.mli | 9 +++++----
```
Don't compile code when printing out the AST

The AST is useful for debugging, often of code that doesn't compile. Therefore, it is silly to compile the code (and abort) before printing the AST.

Adding function declarations to parser

To declare functions the syntax is changed so a "def" keyword must precede the function identifier. This eliminates reduce/reduce conflicts between expressions and the function declarations.

Fix bug in pattern match compilation

The file wasn't seeking back to the pre-pattern match location when the pattern did not match.

Add test cases for pattern matching
Fix interpreter bug in Bne

Top of stack (compare-to value) was not being popped.

Add instructions to store file position on stack

Added instructions to store the file position on the stack and read a file position to seek to from the stack. This enables the bytecode interpreter to return to the position in the file at the conclusion of a pattern statement.

Fix error in built-in print function

The built-in print function didn't pop it's argument from the stack.

Add branch rewriting on second pass
Add Reader module and support for branching

Added the Reader module and support for resolving branch addresses correctly using the 'Label' pseudo instruction and two passes.

<table>
<thead>
<tr>
<th>File</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>1 +</td>
</tr>
<tr>
<td>bytecode.ml</td>
<td>4 ++++</td>
</tr>
<tr>
<td>reader.ml</td>
<td>50 ++++++++++++++++++++++++++++++++++++++++++++++++++</td>
</tr>
<tr>
<td>reader.mli</td>
<td>26 +++++++++++++++++++++++++++++++++++++</td>
</tr>
</tbody>
</table>

4 files changed, 81 insertions(+)

commit df48722be79dd90290f9ff2bf6e394c62dac47cc
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jul 27 15:16:20 2013 -0400

Don't compile inside Bytecode module

This removes the dependency of the Bytecode module on the Compile module. OCaml appears somewhat picky about enforcing non-cyclic dependencies.

<table>
<thead>
<tr>
<th>File</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bawk.ml</td>
<td>5 +++--</td>
</tr>
<tr>
<td>bytecode.ml</td>
<td>8 +++----</td>
</tr>
<tr>
<td>bytecode.mli</td>
<td>4 +++--</td>
</tr>
</tbody>
</table>

3 files changed, 8 insertions(+), 9 deletions(-)

commit 6092ae879e5ed00e52f74100f2435643a21f66ae
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jul 27 14:56:06 2013 -0400

Use ocamldep to find linking order

Use 'ocamldep -sort' to determine the order the .cmo files get linked.

<table>
<thead>
<tr>
<th>File</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>2 +-</td>
</tr>
</tbody>
</table>

1 file changed, 1 insertion(+), 1 deletion(-)

commit ad4836c8a8e890289a5d77a2998242ff560f05e0
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jul 27 14:45:42 2013 -0400

Allow enumerate to do some primitive introspection

Changed the step function for enumerate to allow it to behave differently based on the value of the head of the content list.

This is useful for enumerating bytecode with pseudo instructions, where we want to no give the pseudo instructions their own address.

<table>
<thead>
<tr>
<th>File</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>compile.ml</td>
<td>2 +-</td>
</tr>
<tr>
<td>utile.ml</td>
<td>4 +++--</td>
</tr>
</tbody>
</table>

3 files changed, 4 insertions(+), 4 deletions(-)
commit 3e153a4adf5f5be93ffa7a127487573a38b51a3b
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jul 27 14:19:55 2013 -0400

Add bytecode support for branches

Added the Label pseudo-instruction to the bytecode to allow the
branching addresses to be resolved. Basically labels are entered
by the one stage of the compilation and then resolved to actual
addresses by a later stage.

<table>
<thead>
<tr>
<th>file</th>
<th>changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytecode.ml</td>
<td>5 +++++</td>
</tr>
<tr>
<td>bytecode_types.ml</td>
<td>4 +++-</td>
</tr>
<tr>
<td>compile.ml</td>
<td>15 +++++++-</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3 files changed, 19 insertions(+), 5 deletions(-)</td>
<td></td>
</tr>
</tbody>
</table>

commit 41b603c93a75d06b673989533ed4541735910d63
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jul 27 13:45:43 2013 -0400

Support passing a binary filename in argv

Added support for reading argv[2] as the filename of a binary file
to process. Argument parsing is still very primitive, but this
is enough to build something that works.

<table>
<thead>
<tr>
<th>file</th>
<th>changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bawk.ml</td>
<td>10 ++++++++</td>
</tr>
<tr>
<td>bytecode.ml</td>
<td>6 +++-</td>
</tr>
<tr>
<td>bytecode.mli</td>
<td>4 ++-</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3 files changed, 14 insertions(+), 6 deletions(-)</td>
<td></td>
</tr>
</tbody>
</table>

commit 747889f4963cf7c90a2d1f6f219c5b8ccf59b7663
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jul 25 22:00:04 2013 -0400

Begin implementing pattern matching compilation

A start on the implementation of compiling pattern expressions into
bytecode.

<table>
<thead>
<tr>
<th>file</th>
<th>changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>compile.ml</td>
<td>19 ++++++++</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1 file changed, 18 insertions(+), 1 deletion(-)</td>
<td></td>
</tr>
</tbody>
</table>

commit a74c6bd36d904a37206f7e2b5dc6bff9c74ca91
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jul 25 21:59:37 2013 -0400

Add more instructions to string_of_instruction

<table>
<thead>
<tr>
<th>file</th>
<th>changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytecode.ml</td>
<td>4 +++-</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1 file changed, 3 insertions(+), 1 deletion(-)</td>
<td></td>
</tr>
</tbody>
</table>

commit 368e05b6506f7ea33dfe16c5fb91694195c0ca92
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jul 25 21:56:25 2013 -0400
Add size_of_bind_type function

Added a function to return the size in bytes of a given bind_type.

ast.ml | 7 ++++++
ast.mli | 4 ++++
2 files changed, 11 insertions(+)

commit 3057ec54d1e6ecc46d4fcf6a549c92c05fb0350a0
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jul 25 21:33:15 2013 -0400

Reverse parsing order of pattern tokens

Pattern tokens were stored in the AST in the reverse of the desired order. This switches the order.

parser.mly | 5 +++++
1 file changed, 4 insertions(+), 1 deletion(-)

commit bfc73199e41bf37ca708d6c2e47258b4b3af02d
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jul 25 21:29:51 2013 -0400

Add PC values to bytecode output

Added value of program counter address to instructions when printing the bytecode output.

bytecode.ml | 4 ++++
1 file changed, 3 insertions(+), 1 deletion(-)

commit ede71a47f4d8fa8e1f2b408930444358bd9046007
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jul 25 21:18:37 2013 -0400

Add parsing of hex constants in pattern expression

Added parsing of hexadecimal constants in the pattern expression, as defined in the LRM. Big/little endian conversion should probably be handled by the bytecode at runtime.

parser_help.ml | 18 ++++++++++++++++++
1 file changed, 17 insertions(+), 1 deletion(-)

commit f8dbc43c280dc1264938e567664e38994e95368
Author: Kevin Graney <nanonet@gmail.com>
Date: Wed Jul 24 20:03:51 2013 -0400

Add regression tests

Started the regression test suite with a very simple test case.
Halt at conclusion of program

Added the Hlt instruction to the end of compiled programs, and implemented halt functionality in the interpreter. Previously we just kept reading through to the next instruction in memory.

bytecode.ml | 2 ++
compile.ml | 2 +−
2 files changed, 3 insertions(+), 1 deletion(−)

Add basic arithmetic operators to interpreter

Added basic arithmetic instructions to bytecode interpreter. It's still missing a lot of things, but this is a start.

bytecode.ml | 23 ++++++++++++++++++++++
bytecode.ml | 6 ++++
2 files changed, 27 insertions(+), 2 deletions(−)

Add stub for bytecode execution flag

Added the −e flag to compile a program into bytecode and then execute that bytecode.

bawk.ml | 7 +++++−
bytecode.ml | 4 +++
bytecode.ml | 3 +++
3 files changed, 11 insertions(+), 3 deletions(−)

Fix bug where built-in function addresses increase

Fixed a bug where built-in function addresses increased as the name list is read instead of decreased. (Negative addresses are used to identify built-in functions.)

compile.ml | 3 +−
1 file changed, 2 insertions(+), 1 deletion(−)
commit 51109abddc9345b4147b6771c0f2c3c0eacff4b0
Author: Kevin Graney <nanonet@gmail.com>
Date: Mon Jul 22 23:02:41 2013 -0400

    Add step function to enumerate
    Added an optional argument to enumerate to allow a step function
to be specified.
    
    utile.ml |  4 +++--
    utile.mli |  2 --
    2 files changed, 3 insertions(+), 3 deletions(-)

commit c5f9a558bbec35da21b91546ebc287af0b4b510
Author: Kevin Graney <nanonet@gmail.com>
Date: Mon Jul 22 22:53:57 2013 -0400

    Populate starting environment with built-ins
    Populated the default environment with built-in functions. They
are read from an array, enumerated.

    bawk.ml | 13 +++++++++++
    compile.ml | 14 ++++++++--
    compile.mli | 38 ++++++++++++++++++++++++++++++++++++++
    3 files changed, 60 insertions(+), 5 deletions(-)

commit b970704fb337247656f959c79fc90415f68566f5
Author: Kevin Graney <nanonet@gmail.com>
Date: Mon Jul 22 22:53:13 2013 -0400

    Add enumerate function
    Added enumerate function to Utile library.

    utile.ml |  5 ++++
    utile.mli |  6 +++++--
    2 files changed, 10 insertions(+), 1 deletion(-)

commit 1bbdd68335ca43bafa27186611dc21a3edca399f
Author: Kevin Graney <nanonet@gmail.com>
Date: Mon Jul 22 21:42:09 2013 -0400

    Add function call stubs
    Added preliminary logic for generating bytecode for function
calls. Still stub code. Also added basic environment to support
things like function names (and eventually scoping).

    bytecode.ml |  3 ++--
    compile.ml | 35 ++++++++++++++++++++++++++++++-
    compile.mli |  4 ++--
    3 files changed, 33 insertions(+), 9 deletions(-)

commit 19dc1c220040c645ab88bc8bc946062360e2a53e0
Ignore autogenerated makefiles

Add more stub code to the translator

Add bytecode creation support for binary operators.

Change string_of_operator to produce bytecode

Change string_of_operator from producing human readable strings
to bytecode operator strings. This benefits the bytecode generation
and makes the AST graph more concise.
Add primitive bytecode creation

Added Bytecode_types module to mimic pattern used for Ast_types.
Also got simple translation working, so the -c flag now produces
the bytecode output. Currently only works for primitive programs.

Makefile | 2 +--
bytecode.ml | 24 +++++++++++++++++
bytecode.mli | 10 +++++++++
bytecode_types.mli | 16 +++++++++++
compile.ml | 18 +++++++++++++++++
compile.mli | 5 ++++
tests/simple.bawk | 6 ++++
7 files changed, 55 insertions(+), 26 deletions(-)

commit 3277f84748d4b697c71bc01b8fee79b6a9a91200
Author: Kevin Graney <nanonet@gmail.com>
Date: Wed Jul 17 21:50:48 2013 -0400

Produce ocaml2doc by default

Changed Makefile to produce design_docs target by default. (This
is the ocaml2doc output.)

Makefile | 2 +--
1 file changed, 1 insertion(+), 1 deletion(-)

commit c269dc76c098f6f880e395abf334a6262a7cae20
Author: Kevin Graney <nanonet@gmail.com>
Date: Wed Jul 17 19:34:53 2013 -0400

Automatic dependency generation

Makefile now generates dependencies from ocamldep automatically. It's
a little ugly in that it makes the dependencies regardless of what the
target is (including clean) but this is still better than manually
updating the Makefile all the time.

Makefile | 47 ++++++++++++++++++++++-------------------------------
1 file changed, 19 insertions(+), 28 deletions(-)

commit 21e268ef44da666feaeaabcd77926a3884f25f87f
Author: Kevin Graney <nanonet@gmail.com>
Date: Wed Jul 17 19:15:14 2013 -0400

Add bytecode and compile modules

Add primitive bytecode and compile modules.

Makefile | 11 +++++++++++++
bawk.ml | 2 +--
bytecode.ml | 21 ++++++++++++++++++++++
bytecode.mli | 2 ++
Add support for command line arguments

Added primitive support for command line arguments to distinguish between actions such as compile, output the AST, output the Bytecode, etc.

bawk.ml | 35 +++++++++++++++++++++++++++-
1 file changed, 23 insertions(+), 12 deletions(-)

commit 7c73eafb8a0d38620081c90e417c3110561c7302
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Jul 14 14:01:17 2013 -0400

Expand definition of bind patterns

Work on LRM to expand the definition of bind patterns.

plt_docs/lrm.tex | 20 +++++++++++++++++---
1 file changed, 18 insertions(+), 2 deletions(-)

commit 244a3faae0a96b850562e183fc94103800e6598
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jun 27 14:39:18 2013 -0400

LRM work

plt_docs/lrm.tex | 149 +++++++++++++++++++++++++++++++++++++++-
1 file changed, 108 insertions(+), 41 deletions(-)

commit c52dd031082c4562d9030a5784bc19ec1994fa94
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jun 27 10:43:05 2013 -0400

Add assignment operator

Add right-associative assignment operator.

ast.ml | 4 ++++
ast_types.mli | 1 +
parser.mly | 2 ++
tests/prototype.bawk | 6 ++++
4 files changed, 13 insertions(+)

commit 0dc571931abd863381854157cd1a7c67d17efa98
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jun 27 10:08:00 2013 -0400

Add preliminary pattern expr constant parsing
Added preliminary code to differentiate between integer constants in pattern expressions and those in plain expressions so they can be treated differently. Pattern expression constants will always be interpreted in hexadecimal as a series of bytes, and plain expression constants will be more C-like in their interpretation.

Makefile | 16 +++++++++++++++++
ast.ml | 11 ++++++++++
ast_types.mli | 3 ++
parser.mly | 10 ++++++++ 
parser_help.ml | 5 ++++
parser_help.mli | 6 +++++
scanner.ml | 6 ++++
7 files changed, 38 insertions(+), 19 deletions(-)

commit 03c361378ae507fae887147f1b22f85e51776cff
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jun 27 09:31:13 2013 -0400

Add Sublime Text 2 project file
Also update .gitignore to ignore other Sublime Text files.

.gitignore | 2 ++
bawk.sublime-project | 21 +++++++++++++++++++++
2 files changed, 23 insertions(*)

commit a345dd50a587d0b70d2709cf64a0908521ad2c4a
Author: Kevin Graney <nanonet@gmail.com>
Date: Thu Jun 27 09:16:11 2013 -0400

Remove recursive makefiles
Changing to a flat structure for simplicity.

Makefile | 9 +++++++++++++++++++++++++++++++++
plt_docs/Makefile | 9 ++
2 files changed, 8 insertions(+), 10 deletions(-)

commit b950f353c6c05e0c8400cf63c0114da0b9d0142a
Author: Kevin Graney <nanonet@gmail.com>
Date: Tue Jun 25 22:34:34 2013 -0400

Work on LRM

plt_docs/lrm.tex | 16 ++++++++++++++++++
1 file changed, 13 insertions(+), 3 deletions(-)

commit 0f3edccfc005cc678f3e12d4a20f2f926a86ead
Author: Kevin Graney <nanonet@gmail.com>
Date: Tue Jun 25 20:17:15 2013 -0400

Work on LRM

plt_docs/lrm.tex | 70 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
1 file changed, 42 insertions(+), 28 deletions(-)
commit a05a96b731bd0492b2efe2cbe709b7fcd72786e
Author: Kevin Graney <nanonet@gmail.com>
Date: Mon Jun 24 22:46:32 2013 -0400

    Added to prototype.bawk test cases

tests/prototype.bawk | 6 ++++
1 file changed, 6 insertions(+)

commit 9f52f5177a5b24b421b5e63fab59532a772e5278
Author: Kevin Graney <nanonet@gmail.com>
Date: Mon Jun 24 22:46:09 2013 -0400

    LRM work

plt_docs/lrm.tex | 45 ++++++++++++++++++++++++++++++++++++++++++++++++---
1 file changed, 39 insertions(+), 6 deletions(-)

commit 01cbe8ea67897d83343bc32614160a99d8abe291
Author: Kevin Graney <nanonet@gmail.com>
Date: Mon Jun 24 22:45:16 2013 -0400

    Add string literals

    Added lexer/parser support for string literals. Parser support
may need to be firmed up still (assignment to a string literal
should not be allowed, so we need some kind of lvalue/rvalue
distinction).

ast.ml | 3 +++
ast_types.mli | 1 +
parser.mly | 2 ++
scanner.mll | 72 +++++++++++++++++++++++++++++++++…………………………
4 files changed, 77 insertions(+), 1 deletion(-)

commit 840aa273da4bf579668a4cfdfe7e7f5c40e0d825
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Jun 23 09:47:36 2013 -0400

    Fix warning about non-unit type

    Changed to use ignore in the print out of the GraphViz DOT file
where a function with a side effect is called that returns a value
we don't care about.

ast.ml | 2 +--
1 file changed, 1 insertion(+), 1 deletion(-)

commit 16b81c4478d499ff5d68e9baf0fb2a79762df0af
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Jun 23 09:46:17 2013 -0400

    Add precedence and associativity for comparators

    Resolves a bunch of shift/reduce conflicts in the parser generator.
The precedence ordering is taken from the microc example.

The precedence ordering is taken from the microc example.

A simple test case to test several different parts of the parser.

Useful for looking at the AST GraphViz DOT output of.

A simple test case to test several different parts of the parser.

Useful for looking at the AST GraphViz DOT output of.

Let identifiers start with underscore

Let identifiers start with underscore

Add comparator operators

Added equality comparison along with >, <, >=, <=, !=.

Add comparator operators

Added equality comparison along with >, <, >=, <=, !=.
commit 42d1ff15e1ed4a5223c6bcf03c95d51077b5c1a5
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 23:27:02 2013 -0400

Make edge insertion consistent

Some of the edge insertion for the GraphViz DOT generation of the
AST was inconsistent. Some functions assumed the caller made the
dege and some assumed the callee, resulting in a few double or
missing edges. Now the caller is always required to create a
node and a link to its parent.

ast.ml | 16 ++++---------
1 file changed, 7 insertions(+), 9 deletions(-)

commit 5ce7f0fc576ff7b9a272c2363237ae9d49a4c0c3
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 23:11:21 2013 -0400

Add function calls to AST

Added function calls to the AST. Identified an
inconsistency in how parent edges are drawn in the GraphViz output,
so that bug needs to be fixed in the future.

ast.ml | 7 ++++
ast_types.mli | 1 +
parser.mly | 10 ++++++
scanner.mll | 1 +
4 files changed, 18 insertions(+), 1 deletion(-)

commit 386c4d95c38f1975579fcef7e6152e568960da326
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 21:14:55 2013 -0400

Fix bug with reverse ordered statement_list

Fixed a bug in the parser where the statement_list was in reverse
order, and the GraphViz DOT generating function was reversing it
to hide the problem. Should be correct now, with a Yacc pattern
that reverses it in the parser.

ast.ml | 5 +++
parser.mly | 5 ++++
2 files changed, 6 insertions(+), 4 deletions(-)

commit 0f627538861edd0c9eccaf3f42fda3e3426a521d27
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 20:16:03 2013 -0400

Add support for decimal literals

Added support for base 10 decimal literals.

scanner.mll | 2 ++
Add arithmetic binary operators

Added addition, subtraction, multiplication, and division. Had to define division in terms of the existing FSLASH token. This makes the code a little unclear, and might be changed for clarity reasons in the future.

ast.ml | 11 ++++++++++
ast_types.mli | 3 +++
parser.mly | 10 +++++++++
scanner.mll | 2 +-  
4 files changed, 24 insertions(+), 2 deletions(-)

Make output graph read left-to-right with input

The output GraphViz DOT tree for the AST now reads left-to-right in order with the input lines of code. Before it read right-to-left because the lists were not reversed before being passed to List.fold_left.

ast.ml | 9 +++++---  
1 file changed, 5 insertions(+), 4 deletions(-)
Add pattern expressions to DOT output.

Added pattern expression to the GraphViz DOT output of the AST.

ast.ml | 28 ++++++++++++++++++++++++++-
1 file changed, 27 insertions(+), 1 deletion(-)

commit 16fe0e54bb34ac41e52c74382a6f8ed32fd4ff3
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 18:58:24 2013 -0400

Add expressions to DOT output

Added the expressions to the GraphViz DOT output of the AST.

ast.ml | 13 +++++++++++++
1 file changed, 13 insertions(+)

commit 569b897a0d9b42356f72fc51603f2bd1d86c1d3f
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 18:23:39 2013 -0400

Add support for non-terminals to DOT output

Added support for non-terminal nodes to the GraphViz dot output.

ast.ml | 11 +++++++++--
1 file changed, 9 insertions(+), 2 deletions(-)

commit b4b6df9eff32b4111f25569960620560b82487c0
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 17:10:09 2013 -0400

Add additional punctuation to lexer

Added additional punctuation symbols to lexer for binary operators
as well as parenthesis. Changed strings of length 1 to characters
in pattern matching.

parser.mly | 9 +++++++--
scanner.mll | 23 +++++++++++++++++++--
2 files changed, 26 insertions(+), 6 deletions(-)

commit 466ff9a815d9c8b79df2216b9eae1345664259cc
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 16:18:24 2013 -0400

Allow underscore in literal names

Allow an underscore as the second and subsequent character in
literal names. In the future it may also be allowed as the first
character.

scanner.mll | 2 +-
Add Literals to the AST

ast_types.mli | 1 +
parser.mly | 1 +
2 files changed, 2 insertions(+)

commit 4434b3157a6093affd36ffed17364181749eff84
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 16:18:03 2013 -0400

Remove extraneous make targets in plt_docs

Removed the lrm and proposal targets since it's just as easy
to type "make lrm.pdf" and "make proposal.pdf".

plt_docs/Makefile | 5 +++-
1 file changed, 1 insertion(+), 4 deletions(-)

commit 30fe72378962e036fbff55887fec066fd286b21a
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 15 16:14:31 2013 -0400

Add PLT documents

Added documents for PLT (COMS W4115) to the repository.

.gitignore | 7 ++++
plt_docs/Makefile | 12 ++++
plt_docs/lrm.tex | 27 +++++++++++
plt_docs/proposal.tex | 109 ++++++++++++++++++++++++++++++++++++++++++++++++++
4 files changed, 155 insertions(+)

commit 704d83a3797db42f08b46b00dd6cb2a3c31a8907
Author: Kevin Graney <nanonet@gmail.com>
Date: Wed Jun 12 21:15:26 2013 -0400

Renamed docs folder

Renamed the 'docs' folder since it's a little too generic, and for
this project several types of documentation are expected. The
ocamldoc documentation is now in a folder named 'design_docs'.

.gitignore | 4 +++-
design_docs/Makefile | 2 ++
docs/Makefile | 2 --
3 files changed, 4 insertions(+), 4 deletions(-)

commit e500cbd1e47b791227ff13233cb0c38bf9940674
Author: Kevin Graney <nanonet@gmail.com>
Date: Wed Jun 12 21:11:38 2013 -0400

Renamed docs folder

Renamed the 'docs' folder since it's a little too generic, and for
this project several types of documentation are expected. The
ocamldoc documentation is now in a folder named 'design_docs'.

docs/Makefile | 2 --
3 files changed, 4 insertions(+), 4 deletions(-)

commit 0158e187d22268255c67a6683321535ca3cb
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Jun 9 23:15:40 2013 -0400
Clean up the parse tree dot code generation

Cleaned up the print_tree function to abstract away some of the
dot code generation into a couple routines. Still need to expand
the function to more types in the AST.

ast.ml  |  47 ++++++++++++++++++++++++++++
1 file changed, 28 insertions(+), 19 deletions(-)

cmmitt 691c33954f98e7ba711216915f3833f686eb98e852
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Jun 9 21:28:57 2013 –0400

Change print_tree to print AST in dot form

Changed the print_tree method to print out graphviz dot code to
draw the tree. This is still very crude, but it's much easier to
visualize how the tree is being formed.

To support this change the Ast_types.program type was removed, and
now the root program is represented as a Block statement instead of
a list of statements. This should be a nicer overall
representation anyway. The parser handles the type conversion anyway.

ast.ml  |  37 +++++++++++++++++++++++++++++++++
ast.mli  |  2 +
ast_types.mli  |  4 ++
parser.mly  |  4 ++
4 files changed, 38 insertions(+), 9 deletions(-)

commit d86ec95db986452cda52cc51e728b06e1e25a1a2c858
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Jun 9 19:32:45 2013 –0400

Add variable bindings to pattern

Added support for variable bindings to the pattern matching.

ast_types.mli  |  14 ++++++++------
parser.mly  |  10 +++++------
scanner.mll  |  13 ++++++++------
3 files changed, 28 insertions(+), 9 deletions(-)

commit 39e981dd9607dd6354adec46ceef31885dfd7024
Author: Kevin Graney <nanonet@gmail.com>
Date: Sun Jun 9 19:28:19 2013 –0400

Change hex literals to require 0x prefix

Changed hex literals to require 0x prefix. This avoids ambiguity
with variable name literals, which are required to start with a
letter. (e.g. is 'ff' a variable or 255?)

parser.mly  |  4 ++
speaker.mll  |  4 ++
Add pattern syntax to parser

First cut at getting the pattern matching syntax into the parser.

Ignore auto-generated documentation

Updated ocamldep dependencies. These remain static for now.

Add block statements to grammar

Added support for block statements to the grammar, and began adding some scanner ability for the patterns. The bawk main loop is just dummy logic that prints 'item' for each statement it encounters.
Add .gitignore

Ignore OCaml build files

.gitignore | 11 ++++++++++++
1 file changed, 11 insertions(+)

commit 56c8597a3730aeb0a4d92c21379ce09974b9f418
Author: Kevin Graney <nanonet@gmail.com>
Date: Sat Jun 8 22:45:30 2013 -0400

Initial commit

Basic parser for reading in a hexadecimal string and printing the value in hex.

Makefile | 32 +++++++++++++++++++++++++++++++++++++++
at.mli | 4 ++++
bawk.ml | 9 +++++++
docs/Makefile | 2 ++
parser.mly | 18 ++++++++++++++++
scanner.mll | 30 ++++++++++++++++++++++++
utile.mi | 5 +++++
utile.mli | 5 +++++
8 files changed, 105 insertions(+)

50
Chapter 5

Architectural design

5.1 Design decisions

The design decisions for this project were not made with considerations towards performance or scalability. Design decisions were based around ease of implementation and demonstration of compiler concepts. For this reason some features of a practical language were excluded because they cost significant effort for minimal benefit to the experience of writing a compiler.

5.2 Execution phases

The bawk interpreter consists of a bytecode compiler and a bytecode runtime environment. The two are seamlessly integrated into a single executable program, bawk. By calling this program with a -c option the results of the bytecode compiler are output, and by calling it with no option (or an explicit -e) the given program is compiled internally to bytecode and then immediately executed in the bytecode interpreter.

5.2.1 Bytecode compiler

The bytecode compiler consists of a scanner (generated by OCamllex), a parser (generated by OCamlyacc), and the compiler itself. The scanner parses a sequence of input characters that form a given bawk language program and returns a list of tokens. The token types include punctuation that has meaning in the language, the C-style comments (/*...*/), identifiers, keywords, constants, and string literals. Whitespace is consumed by the scanner, but serves no purpose other than to separate tokens.

There is at least one instance of ambiguity in the scanner. Identifier names, as defined in §3.1.2, consist of a letter or underscore followed by letters, underscores, and/or numerals. However, constants in pattern expressions, as defined in §3.5.1, consist of a sequence of hexadecimal digits, possibly including the letters ’a’-’f’ in uppercase or lowercase. This leads to possible confusion in the code chunk listed below. On line 2 it is not known if cafe123 is a variable name or a hexadecimal number. This ambiguity is actually resolved by the parser, which treats ambiguous constants as constants if they are valid hexadecimal.

Listing 5.1: Possibly ambiguous code
The parser converts the sequence of tokens into an abstract syntax tree (AST). The types used in the AST are defined in `ast_types.mli`, and rely on the OCaml tagged union types to represent the tree. The OCamlyacc specification for the parser is defined in the `parser.mly` file. The parser definition is fairly straightforward. There is some conversion done at this stage from strings to integers, but primarily it simply builds the AST using the tagged union types defined in the `Ast_types` module. The root of the generated AST is a statement, specifically a `Ast_types.Block` type, which contains a list of statements.

The `bawk` compiler can output the AST in a GraphViz format, which is useful for visualizing and debugging the AST. The `-ast` flag to the compiler will output GraphViz code. This can be turned into a PDF by piping it to GraphViz using the command

```
```

The logic to generate the GraphViz code is the majority of the `ast.ml` file. Other functions in that file serve to provide a few helper functions used when parsing the AST.

In the translation phase of the compiler, most of the heavy lifting is done by translating AST structures into basic blocks. In addition, the compiler keeps track of a symbol table in the `env` type in `compile.ml`. The structure forms a linked list, with a new node added each time a new, nested scope, is entered. Each node has a pointer to the parent scope, and a `StringMap` from symbols to either (1) the jump address for a subroutine, or (2) the index into globals. The environment also contains additional information for binding variables, which includes the size of the binding.

### 5.2.2 Bytecode operations

The compiler takes the AST created by the parser portion and translates it to a custom stack-based bytecode. The instructions for the bytecode are defined in `bytecode_types.mli`. Most of the instructions are related directly to the stack, but several are designed specifically for the application of this language: reading binary files.

The translation phase from the AST to bytecode is really a two phase translation process. The first does the basic translation from the AST to basic blocks of bytecode. In this initial translation the various branch instructions are given a unique value referencing a `Label` instruction. The `Label` instruction is a special pseudo instruction that is removed in the second phase of the translation. When the `Label` instructions are removed, their address, or position in the instruction list, is noted. These addresses then replace the values in all the branch instructions, converting them from the value referencing a label to the address of that label.

All the branch instructions in the bytecode operate on absolute addresses. There is no relative addressing in the bytecode by design, simply for simplicity. After the second phase of the translation is complete, the `Label` instructions removed and branch instructions pointing to addresses, the bytecode is ready for the runtime interpreter.
5.2.3 Bytecode runtime

The runtime interpreter is fairly simple, and defined primarily in bytecode.ml. The runtime executes the stack-based bytecode in a virtual machine environment. For simplicity the bytecode instructions composing the program, the execution stack, and global variable values are stored in three different arrays: instructions, stack, and globals respectively. Execution is performed by the recursive exec function, which takes a frame pointer, a stack pointer, and a program counter.

The program counter is simply an index into the instructions array for the instruction being executed in the current cycle. When the recursive call to exec is made, to execute the next instruction, the program counter is either incremented by one to run the next sequential instruction, or, in the case of a branch or function return instruction, called with some absolute address from the stack or an instruction argument. Since all branching is to absolute addresses the bytecode interpreter never computes relative offsets from the current program counter.

The stack grows from an index of zero in the stack array upwards, with items pushed and popped as individual instructions are executed. The argument to exec is a pointer to where the next value can be pushed on the stack. The stack only holds integer values, which is the primary motivation for not supporting functions, such as a printf-like function, that take string arguments.

Function arguments are placed on the stack, and the frame pointer references the location of these arguments. The previous frame pointer is also placed onto the stack and reset when the function returns, allowing nested function calls with arguments to work properly.

Global variables are stored in the globals array, with each symbol, other than function arguments, assigned an offset into this array. The name 'globals' is a bit of a misnomer, since variables local to blocks or functions are also stored in this same array. The slots in the globals array are not re-used; they last the entire duration of the program and are statically assigned to symbols by the compiler.

The bytecode interpreter stack is limited to 1024 values, and the globals array is limited to this same size of 1024 values. These limits are not enforced by the compiler, but the bytecode interpreter will crash if they are exceeded. The length of the instructions array is not explicitly limited.

5.3 Limitations

One major limitation of the language design is the lack of any type of return statement or return value from a function. This prevents the use of many types of useful recursion, and really limits the utility of the language. See §7.3.1 for more details on this limitation.
Chapter 6

Test plan

The testing for this project was performed using a variety of very tiny bawk language programs designed to exercise various features of the language. These programs are provided with the source code in the tests directory. Each program is named test-<name>.bawk and its output is expected to be identical to the test-<name>.bawk.out file. Both the source and expected output files are listed in §8.2.

The tests are all designed to operate on the binary file tests/lichtenstein.png, which serves as a modestly sized sample file. The beginning of the lichtenstein.png file is shown below for convenience, and the file itself can be found packaged with the bawk source.

```
0000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452 .PNG........IHDR
0000010: 0000 0200 0000 0200 0802 0000 007b 1a43 ...........{.C
0000020: ad00 0000 0970 4859 7300 000a f000 000a .....pHYs.......
0000030: f001 42ac 3498 0000 0007 7449 4d45 07d7 ..B.4......TIME..
0000040: 0511 0921 0919 38af 7500 0020 0049 4441 ...!..8.u....IDA
0000050: 5478 dab4 bd5b ac6d 5b76 1dd4 5ae5 63ae Tx...[.m[v..Z.c.
0000060: bdcf bdb7 1ed7 a9b2 2b71 d995 38b1 63f2 ..........+q..8.c.
0000070: b021 4fe7 0181 8404 a280 1421 10e2 2522 .!O........!..""
0000080: 7ef8 46fc 874f 7ef9 e40f 81f8 43e1 2bca ~.F..O~.....C.
0000090: 4f82 2118 0e0a 5809 2838 ca8b d871 1c95 0!H...X.(8...q.
00000a0: 53b6 eb75 5be7 ec35 476f 7cf4 3ec6 1c73 S..u..5Go|>..s
00000b0: edb5 cfb7 65c8 aeab 5be7 eeb3 f75a 73cd .....e[....Zs.
00000c0: 3946 1fbd b7d6 7aeb fcbd fff6 7f2b 7a17 9F....z....+z.
00000d0: 4220 626b 0c0b c075 8af6 604f 2638 f317 ed72 "bkl.*H.9.....
00000e0: 22f6 d09b c075 8af6 604f 2638 f317 ed72 "u..",&8...r
00000f0: d1e6 7cd8 daa5 b953 4e38 e194 2481 8289 ..|....SN8..$...
... 0059270: 3122 4a04 2150 a6c8 fff1 00c1 c91a 2379 1"J.!P........#y
0059280: 1a27 0000 0000 4945 4e44 ae42 6082 .']....IEND.B'.
```

The test programs are executed by a shell script, run_tests.sh, which runs each program and compares its output to the expected values.
Chapter 7

Lessons learned

This project provided a number of valuable software engineering and language design lessons. The language design lessons were invaluable, as this was my first attempt at writing a compiler from scratch that had any level of sophistication beyond regular expression matching on text files.

7.1 Language design

The design of the language was critical to being successful with this project. It was crucial to really think through the specifics of the language before even thinking about writing the compiler’s translation logic. It was critical to write the scanner and parser in parallel with hashing out the language details. Writing these helped force the resolution of many shift/reduce ambiguities and couple reduce/reduce ambiguities in the original, pre-parser, design.

One visible addition to the language that came about from writing the parser is the def keyword before function definitions. This was added to resolve a conflict in the parser. Similarly, when adding if statements I realized that parenthesis were required around the conditional expression when a parser conflict was resolved after adding them.

7.2 Scanner and parser generators

As described in §5.2.1 there is some ambiguity in the scanner distinguishing between hexadecimal values and identifier names. I didn’t realize this ambiguity in the initial language design until testing the compiler with a hexadecimal constant beginning with a letter. This ambiguity was obviously not detected as a shift/reduce or reduce/reduce conflict by the parser generator because it was a collision of definitions in the scanner. Luckily it was possible to change the language definition slightly and add some logic in the parser to resolve the ambiguity.

7.2.1 GraphViz output

The GraphViz output was very useful in the early stages of writing the scanner and parser, but as it became clear that the basic design was solid the utility of the visualization decreased. Despite the decreasing returns on invested time, it wasn’t a whole lot more difficult than implementing an ASCII format of AST output.
7.3 Compiler and bytecode design

7.3.1 Return statement problem

Late in the project I realized that I did not implement any type of return statement in the language. This was an oversight, but unfortunately the design of the bytecode did not make it easy to add at the last minute. The bytecode instruction Rts returns from a subroutine by popping the arguments to the function off the stack. Unfortunately this instruction requires a hard-coded constant for the number of instructions to pop. Without changing the bytecode and its interpreter to support reading this value from the stack it is difficult to return from an arbitrary point in the function. It could likely still be accomplished without a bytecode interpreter change, but it would require some redesign of the translate function in compile.ml to pass around additional information, namely a Label number for the end of the function. This change was just too large to risk making in the last week of the semester, so the language stands without a return statement.

7.3.2 Lack of full support for string types

One way to improve the bawk language would be to add support for strings inside of expressions. This would facilitate adding support for functions such as printf and also the reading of a fixed-length or null-terminated string from a file. Support from strings in this way was one of the initial conceptual ideas behind the language, but late in the semester it became apparent that the bytecode interpreter would have to be overhauled to support both integers and strings on the stack and in the globals array.
Chapter 8

Source code listing

8.1 Compiler and bytecode interpreter

8.1.1 Scanner & parser definitions

scanner.mll

The scanner.mll file contains Ocamllex specifications for the bawk scanner.

```ocaml
let initial_string_buffer = String.create 256
let string_buff = ref initial_string_buffer
let string_index = ref 0

let is_in_string = ref false
let in_string () = !is_in_string

let reset_string_buffer () =
  string_buff := initial_string_buffer;
  string_index := 0

let store_string_char c =
  if !string_index >= String.length (!string_buff) then
    begin
      let new_buff = String.create (String.length (!string_buff) * 2) in
      String.blit (!string_buff) 0 new_buff 0 (String.length (!string_buff));
      string_buff := new_buff
    end;
  String.unsafe_set (!string_buff) (!string_index) c;
  incr string_index

let store_lexeme lexbuf =
  let s = Lexing.lexeme lexbuf in
  for i = 0 to String.length s - 1 do
    store_string_char s.[i];
  done

let get_stored_string () =
```

57
let s = String.sub (!string_buff) 0 (!string_index) in
string_buff := initial_string_buffer;
s
let char_for_backslash = function
| 'n' -> '\010'
| 'r' -> '\013'
| 'b' -> '\008'
| 't' -> '\009'
| c -> c

let hex_character = [ '0'−'9' 'A'−'F' 'a'−'f' ]
let hex_sequence = "0x" hex_character+
let hexdigit_sequence = hex_character+
let literal = [ 'A'−'Z' 'a'−'z' ' ' ''] ['0'−'9' 'A'−'Z' 'a'−'z' ' ' ']

let newline = ( '\010' | "\013\010")

rule token = parse
| [ ' ' ' ' ' ' ''] { token lexbuf } | /* */ { comment lexbuf }

(* language semantics *)
| ':' { COLON } | ';' { SEMICOLON } | ',' { COMMA } |
| '/' { FSLASH } | '{' { LBRACE } | '}' { RBRACE } |
| '(' { LPAREN } | ')' { RPAREN } | '+' { PLUS } |
| '-' { MINUS } | '*' { TIMES } |

(* DIVIDE is the same as FSLASH *)
| '=' { ASSIGN } | "==" { EQ } |
| "!=" { NEQ } | '<' { LT } |
| "<=" { LEQ } | '>=' { GEQ } |

(* type definitions *)
| "int1" { BIND_TYPE(Ast_types.Int_1_byte) } | "int2" { BIND_TYPE(Ast_types.Int_2_bytes) } |
| "int4" { BIND_TYPE(Ast_types.Int_4_bytes) } | "uint1" { BIND_TYPE(Ast_types.UInt_1_byte) } |
| "uint2" { BIND_TYPE(Ast_types.UInt_2_bytes) } | "uint4" { BIND_TYPE(Ast_types.UInt_4_bytes) } |
| "if" { IF } | "else" { ELSE } |
The parser.mly file contains Ocamlyacc specifications for the bawk parser.

Listing 8.2: parser.mly

```ml
open Parser_help

%token <Ast_types.bind_type> BIND_TYPE

/* punctuation */
%token COLON SEMICOLON COMMA FSLASH RBRACE LBRACE RPAREN LPAREN DEF IF ELSE

/* operators */
%token PLUS MINUS TIMES ASSIGN

//... (remaining code omitted for brevity)...
```
/* comparators */
%token EQ NEQ LT LEQ GT GEQ

%token <string> STRING_LITERAL
%token <string> LITERAL
%token <string> INT_LITERAL
%token EOF

/* associativity and precedence */
%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES FSLASH

%start program
%type <Ast_types.statement> program
%
pat_token:
| INT_LITERAL { parse_pattern_const $1 } |
| LITERAL COLON BIND_TYPE { Ast_types.Binding($1, $3) } |
| LITERAL { (* Resolve a little ambiguity from the scanning phase, basically 
  binding variables can’t be valid hex numbers or we don’t know if it’s 
  a hex number or a variable. Here it’s resolved to be a number. *) 
  try parse_pattern_const $1 
  with Failure(_) -> Ast_types.Literal($1) 
  } |
| STRING_LITERAL { Ast_types.PatString($1) } |

pat_expr:
rev_pat_expr ( List.rev $1 )

rev_pat_expr:
| [] |
| rev_pat_expr pat_token { $2 :: $1 } |

statement:
| expr SEMICOLON { Ast_types.Expr($1) } |
| LBRACE statement_list RBRACE { Ast_types.Block($2) } |
| FSLASH pat_expr FSLASH statement { Ast_types.Pattern($2,$4) } |
| DEF LITERAL LPAREN lit_list RPAREN LBRACE statement_list_list RBRACE |
| Ast_types.FunctionDec1({ |
  Ast_types.fname = $2; |
  Ast_types.arguments = $4; |
  Ast_types.body = Ast_types.Block($7)}) |
| IF LPAREN expr RPAREN statement %prec NOELSE |
| Ast_types.If($3, $5, Ast_types.Block([])) |
| IF LPAREN expr RPAREN statement ELSE statement |
| Ast_types.If($3, $5, $7) |
**8.1.2 Interfaces**

**Listing 8.3: parser_help.mli**

```ml
(** [parse_pattern_const] converts a pattern constant into a [PatternBytes] list of integers for the AST. This method interprets leading zeros as significant among other things. *)

val parse_pattern_const: string -> Ast_types.pat_token
```

```ocaml
8.1.2 Interfaces

Listing 8.3: parser_help.mli

```
Listing 8.4: ast_types.mli

(** Types used by the AST *)

type operators = Add | Subtract | Multiply | Divide
| Equal | Neq | Less | Leq | Greater | Geq

type bind_type =
| Int_1_byte
| Int_2_bytes
| Int_4_bytes
| UInt_1_byte
| UInt_2_bytes
| UInt_4_bytes

type literal = string

type expr =
| LitInt of int
| ExprLiteral of string
| Binopt of expr * operators * expr
| Call of string * expr list
| LitString of string
| Assign of string * expr


type pat_token =
| PatternByte of int
| PatternBytes of pat_token list
| Binding of literal * bind_type
| Literal of literal
| PatString of string


type pat_expr = pat_token list

type func_decl = {
  fname: string;
  arguments: string list;
  body: statement;
}

and statement =
| Pattern of pat_expr * statement
| Block of statement list
| Expr of expr
| FunctionDecl of func_decl
| If of expr * statement * statement

Listing 8.5: bytecode_types.mli

type instruction =
| Lit of int (* Push a literal *)
| Drp (* Discard a value *)
| Bin of Ast_types.operators (* Perform arithmetic on top of stack *)
| Two of int (* Convert top of stack to two's complement *)
| Lod of int (* Fetch global variable *)
| Str of int (* Store global variable *)
| Lfp (* Load frame pointer from the stack *)
Sfp (* Store frame pointer to the stack *)
Jsr of int (* Call function by absolute address *)
Ent (* Push FP, FP -> SP, SP += i *)
Rts of int (* Restore FP, SP, consume formals, push result *)
Beq of int (* Branch absolute if top-of-stack is zero *)
Bne of int (* Branch absolute if top-of-stack is non-zero *)
Bra of int (* Branch absolute *)
Beo of int (* Branch if end of file *)
Hlt (* Terminate *)
Rdb of int (* Read a number of bytes from the file *)
Ldp (* Load the position in the file onto the stack *)
Skp (* Seek the file to the position on the stack *)
Label of int (* Used for branching jumps *)

Listing 8.6: ast.mli

(val print_tree: Ast_types.statement -> unit)
(val string_of_bind_type: Ast_types.bind_type -> string)
(val size_of_bind_type: Ast_types.bind_type -> int)
(val is_signed_type: Ast_types.bind_type -> bool)
(val make_terminal: this:int -> ?parent:int -> string -> unit)
(val make_nonterminal: this:int -> ?parent:int -> string -> unit)
(val make_link: int -> int -> unit)
(val folded_printer: ('a -> int -> int -> 'b) -> int -> int -> 'a -> 'b)
(val string_of_operator:)

(*** [print_tree] outputs the AST in GraphViz DOT format. This is useful for
visualizing how the parser is constructing the tree ***)

(*** [string_of_bind_type] converts an [Ast_types.bind_type] to a [string] ***)

(*** [size_of_bind_type] returns the size, in bytes, of an [Ast_types.bind_type] ***)

(*** [is_signed_type] returns true if the bind type is signed and false
otherwise ***)

(*** [make_terminal] creates a terminal node for the node with an id of [this],
and labels it with the string argument. If the optional [parent] argument
is provided a link is also created between this node and the parent. ***)

(*** [make_nonterminal] creates a non-terminal node for the node with an id of
[this], and labels it with the string argument. If the optional [parent]
argument is provided a link is also created between this node and the
parent. ***)

(*** [make_link] creates a link between two of the GraphViz nodes ***)

(*** [folded_printer] returns a function that can be used with [List.fold_left]
to traverse a list of objects that should be children of a common parent.
The parent’s ID is the second argument and the print function is the
first argument. ***)

(*** [string_of_operator] returns a string of the bytecode instruction for
the given operator. *)

val string_of_operator : Ast_types.operators -> string

Listing 8.7: compile.mli

(** Compilation of bawk code *)

module StringMap : (* TODO: figure out how to get rid of this monstrosity! *)
sig
  type key = String.t
  type 'a t = 'a Map.Make(String).t
  val empty : 'a t
  val is_empty : 'a t -> bool
  val mem : key -> 'a t -> bool
  val add : key -> 'a -> 'a t
  val singleton : key -> 'a -> 'a t
  val remove : key -> 'a t -> 'a t
  val merge :
    (key -> 'a option -> 'b option -> 'c option) -> 'a t -> 'b t -> 'c t
  val compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int
  val equal : ('a -> bool) -> 'a t -> 'a t -> bool
  val iter : (key -> 'a -> unit) -> 'a t -> unit
  val fold : (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
  val for_all : (key -> 'a -> bool) -> 'a t -> bool
  val exists : (key -> 'a -> bool) -> 'a t -> bool
  val filter : (key -> 'a -> bool) -> 'a t -> 'a t
  val partition : (key -> 'a -> bool) -> 'a t -> 'a t * 'a t
  val cardinal : 'a t -> int
  val bindings : 'a t -> (key * 'a) list
  val min_binding : 'a t -> key * 'a
  val max_binding : 'a t -> key * 'a
  val choose : 'a t -> key * 'a
  val split : key -> 'a t -> 'a t * 'a option * 'a t
  val find : key -> 'a t -> 'a
  val map : ('a -> 'b) -> 'a t -> 'b t
  val mapi : (key -> 'a -> 'b) -> 'a t -> 'b t
  val merge :
    (key -> 'a option -> 'b option -> 'c option) -> 'a t -> 'b t -> 'c t
  val compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int
  val equal : ('a -> bool) -> 'a t -> 'a t -> bool
  val iter : (key -> 'a -> unit) -> 'a t -> unit
  val fold : (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
  val for_all : (key -> 'a -> bool) -> 'a t -> bool
  val exists : (key -> 'a -> bool) -> 'a t -> bool
  val filter : (key -> 'a -> bool) -> 'a t -> 'a t
  val partition : (key -> 'a -> bool) -> 'a t -> 'a t * 'a t
  val cardinal : 'a t -> int
  val bindings : 'a t -> (key * 'a) list
  val min_binding : 'a t -> key * 'a
  val max_binding : 'a t -> key * 'a
  val choose : 'a t -> key * 'a
  val split : key -> 'a t -> 'a t * 'a option * 'a t
  val find : key -> 'a t -> 'a
  val map : ('a -> 'b) -> 'a t -> 'b t
  val mapi : (key -> 'a -> 'b) -> 'a t -> 'b t
end

val cardinal : 'a t -> int
val bindings : 'a t -> (key * 'a) list
val min_binding : 'a t -> key * 'a
val max_binding : 'a t -> key * 'a
val choose : 'a t -> key * 'a
val split : key -> 'a t -> 'a t * 'a option * 'a t
val find : key -> 'a t -> 'a
val map : ('a -> 'b) -> 'a t -> 'b t
val mapi : (key -> 'a -> 'b) -> 'a t -> 'b t

(** [translate_program] takes an AST statement and returns a list of bytecode
instructions implementing the statement *)

val translate_program : Ast_types.statement -> Bytecode_types.instruction list

val clean_environment : env

64
Listing 8.8: bytecode.mli

1 (** Bytecode generation *)
2 (** [execute_bytecode] is a function called from the main argument parser. It generates and then executes the bytecode instructions. *)
3 val execute_bytecode: Bytecode_types.instruction list -> in_channel -> unit
4 (** [print_bytecode] is a function called from the main argument parser. It generates and then prints out the bytecode instructions. *)
5 val print_bytecode: Bytecode_types.instruction list -> unit
6 (** [string_of_instruction] converts a bytecode instruction into a string representation *)
7 val string_of_instruction: Bytecode_types.instruction -> string
8 (** [execute_instructions] interprets an array of bytecode instructions as a program, executing said instructions. *)
9 val execute_instructions: Bytecode_types.instruction array -> in_channel -> unit
10 (** [is_pseudo] returns true if the given bytecode instruction is a pseudo instruction and false otherwise *)
11 val is_pseudo: Bytecode_types.instruction -> bool
12 val enumerate_instructions: Bytecode_types.instruction list -> (int * Bytecode_types.instruction) list

Listing 8.9: reader.mli

1 (** The Reader module contains functions for extracting and decoding bits of information from an [in_channel] for a given binary file. *)
2 (** [read_byte] reads a single byte from the input channel and returns an integer representing that byte *)
3 val read_byte: in_channel -> int
4 (** [read_bytes] reads the next [n] bytes from the input channel into a list of integers *)
5 val read_bytes: in_channel -> int -> int list
6 (** [read_string_null] reads a null-terminated string from the input channel *)
7 val read_string_null: in_channel -> string
8 (** [read_string_fixed] reads a fixed-length string from the input channel *)
9 val read_string_fixed: in_channel -> int -> string
10 (** [read_unsigned] reads the next [n] bytes as an unsigned integer
11  The algorithm is: read bytes into a list with [read_bytes], traverse the list in reverse order, and accumulate a sum of every byte shifted by it's position in the list. *)
12 val read_unsigned: in_channel -> int -> int
13 (** [get_pos] returns the position in the file *)
14 val get_pos: in_channel -> int
(** [set_pos] set the position in the file *)
val set_pos : in_channel → int → unit

val advance : in_channel → int → unit

(** [is_eof] returns true if the next byte of the file reaches EOF and false otherwise *)
val is_eof : in_channel → bool

---

**Listing 8.10: utile.mli**

1 (** General purpose utility functions *)
2
3 (** [int_of_hex] converts a string of hexadecimal digits into an integer *)
4 val int_of_hex : string → int

val enumerate : ?step : ('a → int → int) → ?start : (int) → 'a list → (int * 'a) list

(** [explode] converts a string into a list of characters *)
val explode : string → char list

(** [implode] converts a list of characters into a string *)
val implode : char list → string

(** [bytes_of_string] converts a string into a list of integers representing the byte values for each character *)
val bytes_of_string : string → int list

(** [signed_of_unsigned] converts an integer into its twos' complement interpretation *)
val signed_of_unsigned : int → int

---

8.1.3 Source Files

**Listing 8.11: bawk.ml**

1 (** The main function of the bawk compiler *)
2 open Compile
3
4 module StringMap = Map.Make(String) (* TODO: can we use the other def? *)
5
6 (** Possible actions for the compiler to take *)
7 type action = Ast | Compile | Execute
8     | D_print_clean_env
9
10 (** [decode_action] reads the [argv] array of command line arguments and returns the action that the compiler is being asked to take. This is fairly primitive, and only the first argument is used to make the decision. *)
11 let decode_action argv =
12     if Array.length argv > 1 then
List.assoc argv.(1) [
  (*-ast*, Ast);
  (*-c*, Compile);
  (*-e*, Execute);
  (*-D-print-clean-env*, D_print_clean_env)
] else Execute;;

let decode_in_file argv =
  if Array.length argv > 2 then
    open_in_bin argv.(2)
  else
    open_in_bin "/dev/random";;

(** [parse_channel] runs an input channel through the lexer and parser phases of the compiler returning an AST. *)

let parse_channel channel =
  let lexbuf = Lexing.from_channel channel in
  let program = Parser.program Scanner.token lexbuf in
  program;;

let _ =
  let action = decode_action Sys.argv in
  let in_file = decode_in_file Sys.argv in
  match action with
    Ast -> Ast.print_tree (parse_channel stdin)
  | Compile -> Bytecode.print_bytecode
    (Compile.translate_program (parse_channel stdin))
  | Execute -> Bytecode.execute_bytecode
    (Compile.translate_program (parse_channel stdin)) in_file

  (* Debug actions *)
  | D_print_clean_env ->
    let env = Compile.clean_environment in
    Printf.printf "Starting symbol table:\n";
    StringMap.iter (fun s i -> Printf.printf "\t%d %s\n" i s)
    env.symbol_map;;

let split_bytes s =
  let split s =
    let num_bytes = (String.length s) / 2 in
    let arr = Array.make num_bytes (String.create 2) in
    for i = 0 to num_bytes - 1 do
      arr.(i) <- String.create 2;
      String.blit s (2*i) arr.(i) 0 2
    done;
    Array.to_list arr
  in
  let pieces =
    if (String.length s) mod 2 == 0 then split s
  else split (Printf.sprintf "0%s" s)
in
List.map (fun x -> int_of_string ("0x" ^ x)) pieces;;

let parse_pattern_const s =
  let lst = split_bytes s in
  let bytes = List.map (fun x -> Ast_types.PatternByte x) lst in
  Ast_types.PatternBytes bytes

let conditional_link_with_parent this_id parent_id =
  if parent_id != -1 then make_link parent_id this_id
  else ();;

let conditional_link_with_parent this_id parent_id =
  if parent_id != -1 then make_link parent_id this_id
  else ();;

let folded_printer func parent_id =
  (fun id x -> let this_id = id + 1 in
    func x this_id parent_id );;

let is_signed_type = function
  | Int_1_byte -> true
  | Int_2_bytes -> true
  | Int_4_bytes -> true
  | UInt_1_byte -> false
  | UInt_2_bytes -> false
  | UInt_4_bytes -> false

let string_of_bind_type = function
  | Int_1_byte -> "int1"
  | Int_2_bytes -> "int2"
  | Int_4_bytes -> "int4"
  | UInt_1_byte -> "uint1"
  | UInt_2_bytes -> "uint2"
  | UInt_4_bytes -> "uint4"
let string_of_operator = function
  | Add -> "Add"
  | Subtract -> "Sub"
  | Multiply -> "Mul"
  | Divide -> "Div"
  | Equal -> "Eq"
  | Neq -> "Neq"
  | Less -> "Lt"
  | Leq -> "Leq"
  | Greater -> "Gt"
  | Geq -> "Geq"

(* each *_print function returns the next id available for use *)

let print_tree prog =
  let rec stmt_print root id parent =
    match root with
    | Pattern(pat_expr, stmt) ->
      let pattern_id = id + 1 in
      let stmt_id = pat_expr_print pat_expr pattern_id id in
      let consumed_ids = stmt_print stmt (stmt_id + 1) stmt_id in
      make_nonterminal "pattern" ~this:id ~parent:parent;
      make_nonterminal "statement" ~this:stmt_id ~parent:parent;
      consumed_ids
    | Block(statement_lst) ->
      make_nonterminal "block" ~this:id ~parent:parent;
      List.fold_left (folded_printer stmt_print id) id statement_lst
    | Expr(expr) ->
      make_nonterminal "expression" ~this:id ~parent:parent;
      expr_print expr (id + 1) id
    | FunctionDecl(decl) ->
      let stmt_id = stmt_print decl.body (id + 1) id in
      let consumed_ids = List.fold_left (folded_printer print_plain stmt_print id) (stmt_id + 1) decl.arguments
      in
      make_nonterminal "arguments" ~this:stmt_id ~parent:id;
      make_nonterminal (sprintf "fdef:%s" decl.fname) ~this:id ~parent:parent;
      consumed_ids
    | If(cond, stmt1, stmt2) ->
      let if_id = expr_print cond (id + 1) id in
      let ifstmt = stmt_print stmt1 if_id id in
      let consumed_ids = stmt_print stmt2 ifstmt id in
      make_nonterminal "if" ~this:id ~parent:parent;
      consumed_ids
    (* | _ ->
      make_nonterminal "other_stmt" ~this:id ~parent:parent;
      id + 1 *)
and expr_print expr id parent =
  match expr with
  | LitInt(value) ->
    make_terminal (string_of_int value) ~this:id ~parent:parent;
    id + 1
  | Binopt(e1, op, e2) ->
    let e2_id = expr_print e1 (id + 1) id in
    let consumed_ids = expr_print e2 e2_id id in
    make_nonterminal (string_of_operator op) ~this:id ~parent:parent;
    consumed_ids
  | Call(name, arg_list) ->
    let name_id = id + 1 in
    let arg_id = name_id + 1 in
    make_nonterminal "call" ~this:id ~parent:parent;
    make_terminal name ~this:name_id ~parent:id;
    make_nonterminal "arguments" ~this:arg_id ~parent:id;
    List.fold_left (folded_printer expr_print arg_id) arg_id arg_list
  | LitString(str) ->
    make_terminal str ~this:id ~parent:parent;
    id + 1
  | Assign(name, expr) ->
    make_nonterminal "assign" ~this:id ~parent:parent;
    make_nonterminal name ~this:(id + 1) ~parent:id;
    expr_print expr (id + 2) id
  | _ ->
    make_nonterminal "other_expr" ~this:id ~parent:parent;
    id + 1

and pat_expr_print lst id parent =
  make_nonterminal "pat_expr" ~this:id ~parent:parent;
  List.fold_left (folded_printer pat_token_print id) id lst

and pat_token_print token id parent =
  match token with
  | Binding(literal, bind_type) ->
    let literal_id = id + 1 in
    let type_id = id + 2 in
    make_nonterminal "binding" ~this:id ~parent:parent;
    make_terminal literal ~this:literal_id ~parent:id;
    make_terminal (string_of_bind_type bind_type)
      ~this:type_id ~parent:id;
    id + 3
  | Literal(literal) ->
    make_nonterminal "literal" ~this:id ~parent:parent;
    make_terminal literal ~this:(id + 1) ~parent:id;
    id + 2
  | PatternBytes(values) ->
    make_nonterminal "bytes" ~this:id ~parent:parent;
    List.fold_left (folded_printer pat_token_print id) id values
  | PatternByte(value) ->
    make_terminal (string_of_int value) ~this:id ~parent:parent;
    id + 1
  | PatString(str) ->
    make_terminal (sprintf "%s" str) ~this:id ~parent:parent;
    id + 1
  | _ ->
make_nonterminal "foo" ~this:(id+1) ~parent:id;

in
print_string "digraph AST {\n";
print_string "ordering = out;\n";
ignore (stmt_print prog 0 0);
print_string "}"

Listing 8.14: compile.ml

open Ast_types
open Bytecode_types

module StringMap = Map.Make(String)
exception Compile_error of string;;

type pattern_binding = {
  loc: int;
  size: int;
}

type env = {
  symbol_map: int StringMap.t;
  bindings: pattern_binding StringMap.t;
  parent: env ref option;
}

let built_in_functions = ["print"; "RP"];
let special_vars = ["LE"];

let clean_environment = 
let preset_binding_list = 
  Utile.enumerate ~step:(fun x y -> y - 1) ~start:((-1)
built_in_functions)
  @ Utile.enumerate special_vars in
let built_ins = 
  List.fold_left (fun map item ->
    let (value, key) = item in
    StringMap.add key (value) map
  ) StringMap.empty preset_binding_list in
{
  symbol_map = built_ins;
  bindings = StringMap.empty;
  parent = None;
}
let new_environment parent_env =
  ref {
    symbol_map = StringMap.empty;
    bindings = StringMap.empty;
    parent = Some parent_env;
    };;

let rec resolve_symbol env name =
  try StringMap.find name !env.symbol_map
  with Not_found ->
match !env.parent with
None -> let error_msg = Printf.sprintf "No such symbol: %s" name in
raise (Compile_error error_msg)
| Some(env) -> resolve_symbol env name;;

let rec resolve_binding env name =
  try StringMap.find name !env.bindings
  with Not_found ->
    match !env.parent with
    None -> let error_msg = Printf.sprintf "No such symbol: %s" name in
             raise (Compile_error error_msg)
    | Some(env) -> resolve_binding env name;;

let add_function env fname addr =
  let symbol_map_new = StringMap.add fname addr env.symbol_map in
  {env with symbol_map = symbol_map_new}

let global_counter = ref (List.length special_vars);;
let get_next_global () =
  global_counter := !global_counter + 1;
  !global_counter;;

let add_variable_force env vname id =
  let symbol_map_new = StringMap.add vname id
  env.symbol_map in
  {env with symbol_map = symbol_map_new}

let add_variable env vname =
  let symbol_map_new = StringMap.add vname (get_next_global ())
  env.symbol_map in
  {env with symbol_map = symbol_map_new}

let rec get_var_address env vname =
  try StringMap.find vname !env.symbol_map;
  with Not_found ->
    match !env.parent with
    None -> env := add_variable !env vname;
             StringMap.find vname !env.symbol_map
    | Some(env) -> get_var_address env vname;;

let add_binding env bname size =
  let vaddr = get_var_address env bname in
  (* let symbol_map_new = StringMap.add bname vaddr env.symbol_map in *)
  let bindings_new = StringMap.add bname { loc = vaddr; size = size}
  !env.bindings in
  {!env with bindings = bindings_new}

exception Bad_binding of string;;

let get_binding_address env bname size =
  let bind_info =
    try StringMap.find bname !env.bindings;
    with Not_found ->
      env := add_binding env bname size;
      StringMap.find bname !env.bindings
    in


if bind_info.size != size then
  raise (Bad_binding "size mismatch")
else bind_info.loc;;

let label_counter = ref 0;;
let get_new_label () =
  label_counter := !label_counter + 1;
!label_counter;;

let form_label_map instructions =
  let arr = Array.make (!label_counter + 1) 0 in
  let enumerated = Bytecode.enumerate_instructions instructions in
  List.iter (fun x ->
    let (addr, instr) = x in
    match instr with
      | Label (id) -> arr.(id) <- addr; ()
      | _ -> ()
  ) enumerated;
arr;;

let resolve_labels instructions =
  let label_map = form_label_map instructions in
  let rec emit_resolution = function
    | [] -> []
    (* rewrite branches *)
    | Bne(id)::t -> Bne label_map.(id) :: emit_resolution t
    | Bra(id)::t -> Bra label_map.(id) :: emit_resolution t
    | Beq(id)::t -> Beq label_map.(id) :: emit_resolution t
    | Beo(id)::t -> Beo label_map.(id) :: emit_resolution t
    | Jsr(id)::t when id >= 0 -> Jsr label_map.(id) :: emit_resolution t
    (* remove the pseudo instructions *)
    | Label(id)::t -> emit_resolution t
    (* all other instructions *)
    | h::t -> h :: emit_resolution t
  in emit_resolution instructions;;

let rec translate_expr env expr =
  let recurse = translate_expr env in
  match expr with
    | LitInt(integer) -> [Bytecode_types.Lit(integer)]
    | ExprLiteral(var_name) ->
      let vaddr = resolve_symbol env var_name in
      [Lod vaddr ]
    | Binopt(e1, op, e2) ->
      let vaddr = resolve_symbol env var_name in
      [Lod vaddr ]
      recurse e1 @ recurse e2 @ [Bin op]
    | Call(func_name, args) ->
      let function_addr = resolve_symbol env func_name in
      (List.concat (List.map recurse args)) @ [Jsr function_addr]
    | Assign(var_name, expr) ->
      let vaddr = get_var_address env var_name in
      translate_expr env expr @ [
        Str vaddr
      ]
<table>
<thead>
<tr>
<th>LitString(str) -&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>let error_msg = Printf.sprintf &quot;Invalid use of string literal: %s&quot; str</td>
</tr>
<tr>
<td>in raise (Compile_error error_msg)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>in raise (Compile_error error_msg)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>and translated_pattern env expr fail_label =</th>
</tr>
</thead>
<tbody>
<tr>
<td>let rec check_item = function</td>
</tr>
<tr>
<td>PatternByte(value) -&gt; [</td>
</tr>
<tr>
<td>Rdb 1;</td>
</tr>
<tr>
<td>Lit value;</td>
</tr>
<tr>
<td>Bin Subtract;</td>
</tr>
<tr>
<td>Bne fail_label (* branch to failed match *)</td>
</tr>
<tr>
<td>]</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>let big_endian = List.flatten (List.map check_item bytes) in</td>
</tr>
<tr>
<td>let little_endian = List.flatten (List.map check_item (List.rev bytes)) in</td>
</tr>
<tr>
<td>let le_label = get_new_label () in</td>
</tr>
<tr>
<td>let end_label = get_new_label () in</td>
</tr>
<tr>
<td>[</td>
</tr>
<tr>
<td>Lod 0; (* Load LE *)</td>
</tr>
<tr>
<td>Bne le_label;</td>
</tr>
<tr>
<td>]</td>
</tr>
<tr>
<td>@ big_endian</td>
</tr>
<tr>
<td>@ [ Bra end_label; Label le_label; ]</td>
</tr>
<tr>
<td>@ little_endian</td>
</tr>
<tr>
<td>@ [ Label end_label; ]</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>let num_bytes = Ast.size_of_bind_type bind_type in</td>
</tr>
<tr>
<td>let vaddr = get_binding_address env literal num_bytes in</td>
</tr>
<tr>
<td>[</td>
</tr>
<tr>
<td>(* TODO: handle EOF here? *)</td>
</tr>
<tr>
<td>Rdb num_bytes;</td>
</tr>
<tr>
<td>]</td>
</tr>
<tr>
<td>@ [ Str vaddr; ]</td>
</tr>
<tr>
<td>]</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>let bytes = Utile.bytes_of_string str in</td>
</tr>
<tr>
<td>let ast_bytes = List.map (fun x -&gt; PatternByte x) bytes in</td>
</tr>
<tr>
<td>translated_pattern env ast_bytes fail_label @ []</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>let register = resolve_symbol env symbol in</td>
</tr>
<tr>
<td>let binding = resolve_binding env symbol in</td>
</tr>
<tr>
<td>let size = binding.size in</td>
</tr>
<tr>
<td>[</td>
</tr>
<tr>
<td>Lod register;</td>
</tr>
<tr>
<td>Rdb size;</td>
</tr>
<tr>
<td>Bin Subtract;</td>
</tr>
<tr>
<td>Bne fail_label</td>
</tr>
<tr>
<td>]</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>List.flatten (List.map check_item expr)</td>
</tr>
</tbody>
</table>
and translate env stmt =
    let recurse = translate env in
    match stmt with
    Block(stmt_list) ->
        List.flatten (List.map recurse stmt_list)
    | Expr(expr) ->
        translate_expr env expr
    | Pattern(pat_expr, stmt) ->
        let start_label = get_new_label () in
        let fail_label = get_new_label () in
        let end_label = get_new_label () in
        let new_env = new_environment env in
        let pattern_code = translated_pattern new_env pat_expr fail_label in
        [ Ldp; Label start_label; Ldp; Beo end_label; ] @ pattern_code @
        translate new_env stmt @
        [ Bra end_label; Label fail_label; Skp; Beo end_label; Rdb 1; Drp; Bra start_label; Label end_label; Skp; Skp;]
    | FunctionDecl(decl) ->
        let start_label = get_new_label () in
        let end_label = get_new_label () in
        let num_args = List.length decl.arguments in
        let new_env = new_environment env in
        List.iter (fun x ->
            let (id,name) = x in
            new_env := add_variable_force !new_env name id;
        ) (Utile.enumerate ~step:(fun x y -> y - 1) ~start:~-100 ~stop:List.rev decl.arguments);
        env := add_function !env decl.fname start_label in
        [ Bra end_label; Label start_label; Ent; ] @
        translate new_env decl.body @
        [ Rts num_args; Label end_label ]
    | If(condition, ifstmt, elsestmt) ->
        let if_label = get_new_label () in
        let end_label = get_new_label () in
        translate_expr env condition
        @ [ Bne if_label; ] @ translate env elsestmt
        @ [ Bra end_label; Label if_label; ]
        @ translate env ifstmt @ [ Label end_label; ];

let translate_program stmt =
    let env = ref clean_environment in
    resolve_labels (translate env stmt @ [Hlt]);;
open Printf
open Bytecode_types
open Ast_types

let string_of_instruction = function
  Lit(integer) -> sprintf "Lit %d" integer
| Bin(operator) -> "Bin " ^ Ast.string_of_operator operator
| Two(num) -> sprintf "Two %d" num
| Rdb(num) -> sprintf "Rdb %d" num
| Jsr(num) -> sprintf "Jsr %d" num
| Rts(num) -> sprintf "Rts %d" num
| Lod(num) -> sprintf "Lod %d" num
| Str(num) -> sprintf "Str %d" num
| Bra(addr) -> sprintf "Bra %d" addr
| Beq(addr) -> sprintf "Beq %d" addr
| Bne(addr) -> sprintf "Bne %d" addr
| Beo(addr) -> sprintf "Beo %d" addr
| Drp -> "Drp"
| Ldp -> "Ldp"
| Skp -> "Skp"
| Lfp -> "Lfp"
| Sfp -> "Sfp"
| Ent -> "Ent"
| Hlt -> "Hlt"
| Label(id) -> sprintf "Label %d (PSEUDO)" id

let is_pseudo = function
  Label(id) -> true
| _ -> false

let enumerate_instructions lst =
  Utile.enumerate ~step:(fun x y -> if is_pseudo x then y else y + 1) lst;;

let print_bytecode instructions =
  List.iter (fun x -> let (i, ins) = x in
    printf "%9d: %s
" i (string_of_instruction ins))
  (enumerate_instructions instructions);;

let execute_instructions instructions on_file =
  let stack = Array.make 1024 0 in
  let globals = Array.make 1024 0 in
  let rec exec fp sp pc =
    (* Printf.printf "(fp=%d sp=%d pc=%d) " fp sp pc;
      Array.iter (fun d -> Printf.printf "%d " d) stack;
      Printf.printf \"\n\"; *)
    match instructions.(pc) with
    | Lit i -> stack.(sp) <- i;
      exec fp (sp + 1) (pc + 1)
    | Bin op ->
      let op1 = stack.(sp - 2) and op2 = stack.(sp - 1) in
      stack.(sp - 2) <- (let boolean i = if i then 1 else 0 in
                        op1 op op2)
    | ...
match op with
  Add => op1 + op2
| Subtract => op1 - op2
| Multiply => op1 * op2
| Divide => op1 / op2
| Equal => boolean (op1 == op2)
| Neq => boolean (op1 != op2)
| Less => boolean (op1 < op2)
| Leq => boolean (op1 <= op2)
| Greater => boolean (op1 > op2)
| Geq => boolean (op1 >= op2)
};
exec fp (sp - 1) (pc + 1)
| Two size =>
let operand = stack.(sp - 1) in
stack.(sp - 1) <- (
  Utile.signed_of_unsigned operand size
);
exec fp sp (pc + 1)
| Jsr (-1) => (* print function *)
  print_endline (string_of_int stack.(sp - 1));
exec fp (sp - 1) (pc + 1)
| Rdb (1) =>
  stack.(sp) <- Reader.read_byte on_file;
  (* Printf.printf "Read byte: %x\n" stack.(sp); *)
exec fp (sp - 1) (pc + 1)
| Rdb (n) =>
  stack.(sp) <- Reader.read_unsigned on_file n;
exec fp (sp - 1) (pc + 1)
| Ldp =>
  stack.(sp) <- Reader.get_pos on_file;
  (* Printf.printf "Store position: %d at \n" stack.(sp) sp; *)
exec fp (sp - 1) (pc + 1)
| Skp =>
  Reader.set_pos on_file stack.(sp - 1);
  (* Printf.printf "Set position: %d
" stack.(sp) sp; *)
exec fp (sp - 1) (pc + 1)
| Bne addr => if stack.(sp - 1) == 0 then
  exec fp (sp - 1) (pc + 1) else
exec fp (sp - 1) addr
| Beq addr => if stack.(sp - 1) != 0 then
  exec fp (sp - 1) (pc + 1) else
exec fp (sp - 1) addr
| Bra addr => exec fp sp addr
| Bee addr =>
  if Reader.is_eof on_file then exec fp sp addr
  else exec fp sp (pc + 1)
| Jsr addr =>
  stack.(sp) <- pc + 1;
exec fp (sp + 1) addr
| Rts(num) =>
  (* Printf.printf "Return to %d from %d (fp=%d, num=%d, sp=%d)\n"*
  stack.(fp - 1) (fp - 1) stack.(sp - 1) num sp; *)
exec stack.(fp) (fp - num) stack.(fp - 1)
| Lod(-2) => (* RP variable *)
  stack.(sp) <- Reader.get_pos on_file;
Listing 8.16: reader.ml

```
let execute_bytecode instructions on_file =
    execute_instructions (Array.of_list instructions) on_file;;
```

```ml
let read_byte ic =
    let byte = input_char ic in
    int_of_char byte;;

let read_bytes ic n =
    let rec read c lst =
        if c == 0 then List.rev lst
        else read (c - 1) (read_byte ic :: lst)
    in read n [];;

let read_string_null ic =
    let rec scan ic str =
        match c with
            | '\000' -> Utile.implode (List.rev str)
            | _ -> scan ic (c :: str)
    in scan ic [];;

let read_string_fixed ic n =
    let rec scan ic str n =
        if n == 0 then Utile.implode (List.rev str)
        else
            let c = input_char ic in
            match c with
                | ',' -> ('\000' :: scan ic (List.rev str))
                | _ -> scan ic (c :: str)
        in scan ic n [];;
```
let read_unsigned ic n =
  let bytes = read_bytes ic n in
  let rbytes = List.rev bytes in
  let rec form_int n shift bytes =
    match bytes with
    | [] -> n
    | h :: t ->
      let next_n = n + (h lsl shift) in
      form_int next_n (shift + 8) t
  in form_int 0 0 rbytes;;

let advance ic n =
  let pos = pos_in ic in
  let new_pos = pos + n in
  seek_in ic new_pos;;

let get_pos ic =
  pos_in ic;;

let set_pos ic n =
  seek_in ic n;;

let is_eof ic =
  let pos = get_pos ic in
  try input_char ic; false with End_of_file -> true in
  set_pos ic pos; result;;

Listing 8.17: utile.ml

let int_of_hex str =
  int_of_string ("0x" ^ str);;

let enumerate ?step:(step=(fun x y -> y + 1)) ?start:(start=0) lst =
  let rec enum count = function
    | [] -> []
    | h :: t -> (count, h) :: (enum (step h count) t)
  in enum start lst;;

let explode s =
  let rec exp i l =
    if i < 0 then l else exp (i - 1) (s.[i] :: l) in
  exp (String.length s - 1) [];;

let implode l =
  let res = String.create (List.length l) in
  let rec imp i = function
    | [] -> res
    | c :: l -> res.[i] <- c; imp (i + 1) l in
  imp 0 l;;

let bytes_of_string str =
let chars = explode str in
List.map int_of_char chars;;

let signed_of_unsigned value num_bits =
let shift = num_bits - 1 in
let mask = 1 lsl shift in
if mask land value > 0 then (value land (lnot mask)) - mask
else value;;

8.2 Test files

All test files are designed to be run on the input file lichtenstein.png unless specified otherwise.

8.2.1 test-arith.bawk

Listing 8.18: Test program (test-arith.bawk)
1 print(0 + 10 - 9 + 3);

Listing 8.19: Expected output (test-arith.bawk.out)
1 4

8.2.2 test-cond1.bawk

Listing 8.20: Test program (test-cond1.bawk)
1 if (100 > 100*2) {
2 print(1);
3 } else {
4 print(0);
5 }

Listing 8.21: Expected output (test-cond1.bawk.out)
1 0
2 100
### 8.2.3 test-func1.bawk

Listing 8.22: Test program (test-func1.bawk)

```bash
1
2 def food () {
3     print(100);
4 }
5
6 print(1);
7 food();
8 print(2);
9 food();
10 print(3);
```

Listing 8.23: Expected output (test-func1.bawk.out)

```bash
1 1
2 100
3 2
4 100
5 3
```

### 8.2.4 test-func2.bawk

Listing 8.24: Test program (test-func2.bawk)

```bash
1 print(1);
2
3 def food () {
4     print(100);
5 }
6
7 print(2);
```

Listing 8.25: Expected output (test-func2.bawk.out)

```bash
1 1
2 2
```

### 8.2.5 test-func3.bawk

Listing 8.26: Test program (test-func3.bawk)

```bash
1 a = 100;
2
3 def bar(c) {
4     print(c);
5 }
6
7 def foo(a, b, c) {
```
print (a);
print (b);
a = 10;
bar(c * 2);
print (a);
print (c);
c = c * 2;
print (c);
}

foo(50+20, 10, a);

Listing 8.27: Expected output (test-func3.bawk.out)

70
10
100
200
10
100
200

8.2.6 test-pat1.bawk

Listing 8.28: Test program (test-pat1.bawk)

/89504e47/ { /* should match */
  print(10);
}

/89504347/ { /* should not match */
  print(20);
}

print(30);

Listing 8.29: Expected output (test-pat1.bawk.out)

10
30

8.2.7 test-pat2.bawk

Listing 8.30: Test program (test-pat2.bawk)

/89504e47/ { /* should match */
  print(10);
}

/89504e47/ { /* should match */
  print(20);
}
8
9 print(30);

Listing 8.31: Expected output (test-pat2.bawk.out)
1 10
2 20
3 30

8.2.8 test-pat3.bawk

Listing 8.32: Test program (test-pat3.bawk)
1 /8950/ { /* should match */
2     print(10);
3 }
4
5 /8951123456789/ { /* should not match */
6     print(20);
7 }
8
9 /8950/ { /* should match */
10     print(40);
11 }
12
13 print(30);

Listing 8.33: Expected output (test-pat3.bawk.out)
1 10
2 40
3 30

8.2.9 test-pat4.bawk

Listing 8.34: Test program (test-pat4.bawk)
1 a = 100;
2
3 /89 foo:int1 4e47/ { /* should match */
4     print(foo);
5 }
6
7 /89504e47/ { /* should match */
8     print(20);
9 }
10
11
12 print(30);

83
## 8.2.10 test-pat5.bawk

### Listing 8.36: Test program (test-pat5.bawk)

```bash
1 /89 foo2:int4/ { /* should match */
2   print(foo2); /* foo = 0x504e470d */
3 }
```

### Listing 8.37: Expected output (test-pat5.bawk.out)

```
1 1347307277
```

## 8.2.11 test-pat6.bawk

### Listing 8.38: Test program (test-pat6.b awk)

```bash
1 /* these values should be different */
2 /000a value:int4/ { print(value); }  
3 /000a value:uint4/ { print(value); } 
4 /000a value:int2/ { print (value); }  
5 /000a value:uint2/ { print (value); } 
6 /000a value:int1/ { print (value); } 
7 /000a value:uint1/ { print (value); } 
8 /* these values should be the same */
9 /000d value:int4/ { print(value); }  
10 /000d value:uint4/ { print(value); } 
11 /000d value:int2/ { print(value); } 
12 /000d value:uint2/ { print(value); } 
13 /000d value:int1/ { print(value); } 
14 /000d value:uint1/ { print(value); } 
```

### Listing 8.39: Expected output (test-pat6.bawk.out)

```
1 −268435446
2 4026531850
3 −4096
4 61440
```
8.2.12  test-pat7.bawk

Listing 8.40: Test program (test-pat7.bawk)

1
2 my_RP = RP;
3 print (my_RP);

4 / foo:int1 19 38af / { 
5   print (foo);
6   my_RP = RP;
7   print (my_RP);
8 }

9 print (my_RP);
10 RP = my_RP;
11
12 / foo:int1 00 / { 
13   print (foo);
14 }

15 }

Listing 8.41: Expected output (test-pat7.bawk.out)

1 0
2 9
3 72
4 72
5 117

8.2.13  test-pat8.bawk

Listing 8.42: Test program (test-pat8.bawk)

1
2 /4e47 value:int1/ { 
3   print (value);
4   print (RP);
5   /value value2:int2/ { 
6     print (value2);
7     print (RP);
8   }
9 }

10


### Listing 8.43: Expected output (test-pat8.bawk.out)

1. 13
2. 5
3. 18760
4. 14

### 8.2.14 test-pat9.bawk

### Listing 8.44: Test program (test-pat9.bawk)

```bash
/* LE defaults to false (0) */
LE = 1;

/ 091938af / {
    print(100);
    print(RP);
}

/ af381909 / {
    print(200);
    print(RP);
}

/ 09 19 38 af / {
    print(300);
    print(RP);
}

/ af 38 19 09 / {
    print(401);
    print(RP);
}

LE = 0;

/ 091938af / {
    print(500);
    print(RP);
}

/ 09 19 38 af / {
    print(600);
    print(RP);
}
```

### Listing 8.45: Expected output (test-pat9.bawk.out)

1. 100
2. 72
3. 200
4. 72
5. 300
8.2.15 test-pat10.bawk

Listing 8.46: Test program (test-pat10.bawk)

```awk
LE = 0;
/ "IHDR" / {
    print(100);
    print(RP);
}
LE = 1;
/ "IHDR" / {
    print(200);
    print(RP);
}
```

Listing 8.47: Expected output (test-pat10.bawk.out)

```bash
100
16
200
16
```

8.2.16 test-pat11.bawk

Listing 8.48: Test program (test-pat11.bawk)

```awk
/"IEND" end:uint2/ { print(end); }
/"IEND" end:int2/ { print(end); }
```

Listing 8.49: Expected output (test-pat11.bawk.out)

```bash
44610
−20926
```
### 8.2.17 test-scope1.bawk

**Listing 8.50: Test program (test-scope1.bawk)**

```bash
1
2 a = 1000;
3 b = 200;
4
5 def foo() {
6   a = 10 + 100;
7     print(a);
8     print(b);
9 }
10
11 print(a);
12 foo();
13 print(a);
14 print(b);
```

**Listing 8.51: Expected output (test-scope1.bawk.out)**

```
1 1000
2 110
3 200
4 110
5 200
```

### 8.2.18 test-scope2.bawk

**Listing 8.52: Test program (test-scope2.bawk)**

```bash
1
2 b = 200;
3
4 def foo() {
5   a = 10 + 100;
6     print(a);
7     print(b);
8 }
9
10 a = 1000;
11
12 print(a);
13 foo();
14 print(a);
15 print(b);
```

**Listing 8.53: Expected output (test-scope2.bawk.out)**

```
1 1000
2 110
3 200
4 110
5 200
```
8.2.19 test-useful1.bawk

Listing 8.54: Test program (test-useful1.bawk)
1 /* print the size of a PNG file */
2
3 /89 "PNG" 0d 0a 1a 0a/ {
4   /length:uint4 "IHDR"/ {
5     /width:uint4 height:uint4/ {
6       print(width);
7       print(height);
8     } } 
9 } 
10 }

Listing 8.55: Expected output (test-useful1.bawk.out)
1 512
2 512

8.2.20 test-var1.bawk

Listing 8.56: Test program (test-var1.bawk)
1
2 a = 10 + 20 * 20;
3 b = 20 * 20;
4 print(a);
5 print(b);

Listing 8.57: Expected output (test-var1.bawk.out)
1 410
2 400

8.3 Misc. Files

Listing 8.58: Makefile
1 DEP_FILENAME=.ocamldeps.mk
2
3 OBJS = utile.cmo \
4   reader.cmo \
5   scanner.cmo \
6   parser_help.cmo \
7   parser.cmo \
8   ast.cmo \
9   compile.cmo \
10  bytecode.cmo \
11  bawk.cmo \
12
13 default: bawk plt_docs/lrm.pdf plt_docs/proposal.pdf design_docs
Listing 8.59: run_tests.sh

#!/bin/sh

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0
Usage() {
    echo "Usage: testall.sh [options] [.mc files]"
    echo "-k   Keep intermediate files"
    echo "-h   Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
        echo "FAILED"
        error=1
        fi
        echo "$1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile. Differences, if any, written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 >>$3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
        SignalError "$1 differs"
        echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
        SignalError "$1 failed on $*"
        return 1
    }
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\///
    s/.mc//'`
    reffile=`echo $1 | sed 's/.mc$//'`
    basedir=`echo $1 | sed 's/\/[^\/]*/\/\/[^\/]*/g'`
    echo "-n "$basename"
    echo $1>&2
    echo "##### Testing $basename" 1>&2
    generatedfiles=""
    generatedfiles="$generatedfiles $({basename}.i.out)" &&
    Run "$BAWK" "-e tests/lichtenstein.png" "$1 "$2 $({basename}.i.out) &&
    Compare $({basename}.i.out) ${reffile}.out $({basename}.i.diff"
#generatedfiles="$generatedfiles $(basename).c.out" &&
#Run "$BAWK" "−c" "<$1 ""> $(basename).c.out &&
#Compare $(basename).c.out $(reffile).out $(basename).c.diff

# Report the status and clean up the generated files
if [ $error −eq 0 ]; then
  if [ $keep −eq 0 ]; then
    rm -f $generatedfiles
  fi
  echo "OK"
  echo "###### SUCCESS" 1>&2
else
  echo "###### FAILED" 1>&2
  globalerror=$error
  fi
}

while getopts kdpsh c; do
  case $c in
  k) # Keep intermediate files
      keep=1
      ;;
  h) # Help
      Usage
      ;;
  esac
  done

shift `expr $OPTIND − 1`

if [ $# −ge 1 ]
then
  files=$@
else
  files="tests/fail−*.bawk tests/test−*.bawk"
fi

for file in $files
do
  case $file in
  *.test−*)
    Check $file 2>> $globallog
    ;;
  *.fail−*)
    CheckFail $file 2>> $globallog
    ;;
  *)
    echo "unknown file type $file"
    globalerror=1
    ;;
  esac
done

exit $globalerror