XAWK
Language Reference Manual and Project Report

John Cieslewicz [1]
johnc@cs.columbia.edu

Gabriela Creţu [1]
gcretu@cs.columbia.edu

Shi Tak Man [2]
sm2173@columbia.edu

Prashant Puri [3]
pp2119@columbia.edu

December 21, 2004

[1]First-year Ph.D student in the Department of Computer Science.
[2]Junior in the School of Engineering and Applied Science.
[3]Masters student in the Department of Computer Science.

Contents

Chapter 1

Welcome to *XAWK*!

1.1 Introduction

XAWK is a programming language for the intuitive processing of XML files. As its name implies, *XAWK* is based on the AWK language and the design goal is to make *XAWK* as easy to use for processing XML files as AWK is for processing space-separated text files.

1.2 Related Work

AWK was created by Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan. It is an interpreted programming language that processes text files by searching for regular expressions. When a pattern is matched, AWK executes the user-specified instuctions associated with that pattern.

The idea of creating a language similar to AWK that operates on XML files is not new. Peter von der Ahé and Jakob Justsen built XAWK which does the same actions as AWK, but on XML files "in a way native to xml," that is, node-based as opposed to line-based processing [4].

Another approach was made by Stefan Tramm, who built XML GAWK. He replaced AWK's line-by-line file processing with an XML parser, thus processing the XML file token-by-token. XML GAWK matches the regular expression against any element name, element value, attribute name or attribute value [3].

Finally, this work is based, in part, on unfinished work to develop XAWK at Columbia University from November 2003 [1].

1.3 Language Features

1.3.1 Robust

Most importantly, *XAWK* will be a robust language with well-defined behavior. The implementation will provide thorough feedback for errors in user programs and XML input files.

1.3.2 Ease of Use

XAWK's program structure is very similar to AWK's. A programmer familiar with AWK will be able to write an *XAWK* program with ease.

In *XAWK*, the familar AWK pattern expression is replaced by a path expression whose form consists of a subset of the XPath syntax. Expressions refer, therefore, to either a node (an element or text) in the XML parse tree or the attribute of an element [1, page 3].

XAWK supports a subset of the rich features found in AWK. The language inclues standard operators for manipulating numerical and string variables. It also provides associative arrays. Additionally, *XAWK* includes built-in variables, such as an associative array of attributes on the current element, the name and value of the current element, and the fully qualified path to the current element.

Please see Section 1.4 for an example of *XAWK* in action.

1.3.3 Portable Implementation

Written in JAVA, the *XAWK* interpreter is portable to all systems with a supported JAVA Virtual Machine implementation.

From the command line, the user supplies a *XAWK* program file and one or more XML files to be processed. Similar to AWK, the user may also specify the *XAWK* program commands directly from the command line.

1.4 Example

XAWK can be used to compute and evaluate information from an XML file with ease. For example, if we have an XML file[1] containing a class roster and grade information, we can calculate the students' grades and print a class report. This can be implemented using the following sample code:

```
#Similar to XAWK, there is a special section for initialization
BEGIN{
  numStudents = 0;
  teacher = "unknown";
  teacher_email = "none";
  studentNames;
  studentGrades;
}

#This path matches a "class" tag in the root document with a child "teacher" tag.
#This small function stores teacher information
/class/teacher/{
  #CA is a built-in associative array of the attributes on the current element
  teacher = CA[name];
  teacher_email = CA[email];
}

#Record the student's name and average
/class/student/{
  studentNames[numStudents] = CA[name];
  problemSetScores;
  numProblemSets;
  projectScores;
  numProjects;
```

[1]Please see Appendix A for an example of this file.

```
  # This nested path matches tag "assignment" within the /class/student/ path.
  # XAWK matches all "assignment" tags within the context of its parent, "student".
  /assignment/{
    if(CA[type] == "problem set"){
      numProblemSets ++;
      grade = 0;
      # text() matches a text value, i.e. <grade>value</grade>
      /grade/text(){
        grade = CV;
      }
      problemSetScores[numProblemSets-1] = grade;
    }else if(CA[type] == "project"){
      numProjects ++;
      grade = 0;
      /grade/text(){
        grade = CV;
      }
      projectScores[numProjects-1] = grade;
    }
  }

  # Now compute the averages
  psAvg = 0;
  for(score in problemSetScores){
    psAvg += score;
  }
  psAvg = psAvg/numProblemSets;

  projAvg = 0;
  for(score in projectScores){
    projAvg += score;
  }
  projAvg = 0;

  studentGrades[numStudents] = psAvg*.40 + projAvg+.60;
  numStudents ++;
}

# The END block contains actions to be executed after the entire
# XML file has been processed.
END{
  print("----------------");
  print("Teacher: " + teacher);
  print("Email: " + teacher_email;
  print("----------------");

  for(n = 0; n < numStudents; n++){
    print((n+1) +"]\t" + studentNames[n]);
    print("\tGrade: " + studentGrades[n]);
  }
  print("----------------");
}
```

1.5 Implementation Issues

A key decision in the implementation of *XAWK* is the choice of an application programming interface (API) to use for processing the XML files. There are two standard APIs for processing XML documents with Java, the Simple API for XML (SAX) and the Document Object Model (DOM). For each API there exist different versions and different parsers that support it.

SAX, the Simple API for XML, is "event driven" [2]. SAX parses the XML document and provides content to the client application via callback functions. Because SAX uses call backs, the client application is responsible for maintaining any information about the XML document content. The call back design also makes SAX fast and memory efficient. Since the entire XML file is not stored in the memory, it is possible to process large files. SAX makes one pass over the XML file and, therefore, is not the right API choice when the client application needs access to the whole document at once or random access in the file. In that case, DOM is a better solution.

DOM, the Document Object Model, builds a model of the XML file as a tree in memory [2]. This makes DOM more flexible, but at the same time it makes it depend on the parser classes and, consequently, on the performance of the parser that is used. DOM is not as fast as SAX and needs a lot of memory for the document trees.

Because *XAWK* processes the XML file in a single pass, the SAX API offers a number of advantages. The better speed and memory efficiency of SAX allows *XAWK* to process larger XML files more quickly than a DOM API. The single pass nature of the SAX API is also appropriate for *XAWK* since *XAWK* processes a file from top to bottom. The main challenge of using the SAX API is the callback interface. The *XAWK* interpreter must control the advance through the XML document, because, as each path is matched, the interpreter must be able to advance according to the user program. Unfortunately, the SAX API's callback interface places the XML parser in control of advancing through the XML file.

To overcome this challenge, we propose that the *XAWK* interpreter be composed of two threads. One thread handles the SAX callbacks, assembling the parsed XML file into useable objects that are then placed in a thread-safe queue. The other thread would execute the *XAWK* user program, extracting XML objects from the queue as needed.

1.6 Summary

XAWK is a powerful tool for processing XML files. Its familiar AWK-like syntax enhances the language's accessibility, while also providing the programmer with ample flexibillity to process diverse XML files.

Chapter 2

Language Reference Manual

2.1 Lexical Aspects

XAWK programs use the Unicode character set. Anywhere in *XAWK* source code, an escape sequence of the form `uxxxx`, where `xxxx` is a four-digit hex number represented using the characters 0-9, a-f, and A-F, encodes an arbitrary 16-bit Unicode character.

Comments start at a pound character (#) and continue to and end-of-line character.

Identifiers (e.g., variable names, label) start with a lettter followed by letters, digits, or the underscore character (_). The definitions of letter and digit follow the Unicode standard. *XAWK* is case sensitive.

String literals begin and end with double quotes(̈). In additon to the globally usable Unicode escapte sequence `uxxxx`, the following escape sequences are also accepted:

`n`	newline (`u000A`)
`t`	tab (`u0009`)
`b`	backspace (`u0008`)
`r`	return (`u000D`)
`f`	form feed (`000C`)
`X`	backslash (`005C`)
`'`	single quote (`0027`)
`"`	double quote (`0022`)
`xxx`	octal character, each digit is 0-7, cannot exceed (`377`)

AWK accepts any backslash-escaped character; XAWK *uses the Java escape sequences.*

2.1.1 Keywords

The identifiers listed below are reserved as keywords:

`break`	`continue`	`if`	`else`
`do`	`while`	`for`	`in`
`print`	`printf`	`true`	`false`

`;`
`break`[*label*]`;`
`continue`[*label*]`;`
`do` *statement* `while(`*expression*`);`
expression`;`
`if(`*expression*`)` *statement* [`else` *statement*]`;`
`for(`*expression*`;`*expression*`;`*expression*`)` *statement*`;`
I/O statement`;`
`for(`*variable* `in` *array*`)` *statement*`;`
`while(`*expression*`)` *statement*`;`
pattern {*statements*}
{*statements*}

2.2 Statements

Any statement may be proceeded by a label:
label:

These labels are used by the `break` and `continue` statements to terminate and immediately restart the named block. The scope of each label is limited to its statement.

A lone semicolon `;` denotes the empty statement.

Sequences of pattern statements have special meaning: they are considered to search as a block. Specifically, all patterns are first checked against the current node, which must be an element, then checked against every child, grandchild, etc. of the current element in a preorder depth-first search, i.e., the order in which information appears in an XML file. To be considered a pattern statement block, the pattern statements must appear consecutively in the *XAWK* program file. A non pattern statement will

There is an implicit starting path whenever an action is executed that referes to the element on which the patter that invoked the action. A pattern block terminates once the cursor has advanced to the next sibling of its starting context, or if it explicitly terminates itself with a `break` statement.

Unlike AWK, XAWK *does not accept newlines as statement terminators*
XAWK*'s* `continue` *subsumes the behavior of AWK's* `next` *statement.*

2.2.1 I/O Statements

The `print` statement is a special I/O statement whose output may be redirected into files or pipes.

identifier	variable reference
identifier `[`*expression*`]`	associative array reference
`@`*expression*	attribute reference, shorthand for `CA[`*expression*`]`
string literal	
numeric literal	

2.3 Expressions

2.3.1 Operators

XAWK's operators in order of decreasing precedence:

`++ --`	increment, decrement (pre- and postfix)
`+ - !`	unary plus, minus, and logical NOT
`* / %`	multiplication, division, modulus
`+ -`	addition, subtraction
`< <= > >= != ==`	relationals
	string concatenation (juxtaposition)
`in`	array membership
`&&`	logical AND
`||`	logical OR
`?:`	conditional (ternary)
`= += -= *= /= %=`	assignment

Operators are left associative except for assignment and `?:`. Any expression may be parenthesized.

The logical AND `&&`and OR `||` operators are evaluated from left to right using *short-circuit evaluation.*

Concatenation by juxtaposition poses an ambiguity problem in the grammer. For example, a statement such as `x ++ y` which could be parsed as `(x++) y` or `x (++y)`. To resolve this ambiguity, our grammar is greedy and will parse the previous example using the first parse solution.

XAWK*does not include AWK's regular expression matching operators or its little-used exponentiation operators.*

2.4 Pattern Experssions

XAWK's pattern expressions are a subset of the XPath syntax. Each path expression refers to either a node (element or text) in the XML document tree or an attribute of an element.

Text and attribute constraints may only appear at the end of a path expression, since they have no sub-nodes. The `doc` directive may only appear at the beginning of a pattern expression in the outermost pattern block.

When the expression of a `doc` directive is a string, it is treated as a filename. Like AWK, *XAWK* uses a filename as a sort of cache for a file descriptor. The file is opened when the name is first encountered. Later appearances simply reference the open file. After the end of a file is

`doc(`*expression*`)`	matches a file
identifier	matches an element with a given name
`*`	matches any element (not attributes or text)
`text()`	matches a text node
`@`*name*	matches attribute *name*
`@*`	matches all attributes
$path_1$`/`$path_2$	matches children of $path_1$ that match $path_2$
$path_1$`//`$path_2$	matches any descendants of $path_1$ that match $path_2$

reached, the file is automatically closed and it it reopened at the beginning if the filename appears again.

In the future, a `doc` expression may be a URL or refer to an in-memory tree.

The order in which patters are executed varies depending on the object being referenced. For elements and text blocks, actions are executed in the order in which these objects appear int he XML file or tree. Because the order in which attributes appear in XML files is irrelevant, patters that refer to attributes are executed in the order in which they appear in the XML file.

Underlying this is an operational model in which the search steps over objects (i.e., elements and text blocks), but not attributes.

2.5 Built-in Variables

`PATH`	Fully-qualified path to current object
`CA`	Associative array of attributes on current element
`CE`	Name of current element
`CV`	Value of current object (e.g., text, attribute value, element name)

Chapter 3

XAWK Examples

In this chapter the authors present examples of useful *XAWK* programs.

3.1 Grading XML File

The xml file used in this example has the following format:

```
<students>
<group name="XAWK" description="XML Processing language" ta="SE" lrm="received">
<student lname="Cieslewicz" fname="John" email="" hw1="88" hw2="80" final="85" project="90" mt="60" />
<student lname="Cretu" fname="Gabriela" email="" hw1="86" hw2="81" final="80" project="90" mt="68" />
<student lname="Puri" fname="Prashant" email="" hw1="87" hw2="80" final="85" project="90" mt="61" />
<student lname="Man" fname="Shi Tak" email="" hw1="87" hw2="80" final="85" project="90" mt="69" />
</group>

<!-- More groups here -->

</students>
```

3.2 Grading Program

This program processes the grading xml file and outputs student grades. It uses pattern matching, control statements, associative arrays, and mathematical operations.

```
 #setting these variables to 0 is optional since they will be initialized to 0 upon their first use.
 numStudents = 0;
 numGroups = 0;

 Hm1Average = 0;
 Hm2Ayerage = 0;
 midtermAverage = 0;
 projectAverage = 0;
 finalAverage = 0;

#matches the outermost tag "students" with "group"
/doc("roster.xml")/students/group/{
  print "Group Name:", CA["name"];
  print "Group Description is:", CA["description"];
  numGroups++;
  print "Number of Groups", numGroups;
  #inner tag "student"
  /student/{
    StudentFullName = @fname " " @lname;
    studentNames[numStudents] = StudentFullName;
    HomeworkScore;
    numHomework = 0;
    #Find out Hm1, Hm2, Mt, Final, Project and calculate grade

    grade = @hw1;
    numHomework++;
    HomeworkScore[numHomework-1] = grade;
    print "Homework1 Grade: ", HomeworkScore[numHomework-1];
    Hm1Average += grade;
    numHomework++;
    grade = @hw2;
    HomeworkScore[numHomework-1] = grade;
    print "Homework2 Grade: ", HomeworkScore[numHomework-1];
    Hm2Average += grade;
    grade = @mt;
    midtermScores = grade;
    print "midterm Grade: ", midtermScores;
    midtermAverage += grade;
    grade = @final;
    finalAverage += grade;
    finalScore = grade;
    print "Final Grade: ", finalScore;
    grade = @project;
    ProjectScore = grade;
    projectAverage += grade;
    print "ProjectScore ", ProjectScore;

    HmAvg = 0;
    for(i = 0; i< numHomework ; i++) {
      HmAvg += HomeworkScore[i];
    }
    HmAvg = HmAvg/numHomework;

    studentGrades[StudentFullName] = HmAvg*.10 + ProjectScore*.40 + midtermScores*.20 + finalScore*.30;
    print "\nStudent Name :" , CA["fname"]," ",  CA["lname"];
    print "\tGrade: " , studentGrades[StudentFullName] ;
    numStudents++;
  } # end of student
} # end of group

total = numStudents ;
Hm1Average = Hm1Average/total;
Hm2Average = Hm2Average/total;
HmAverage  = Hm1Average + Hm2Average;
HmAverage /= 2;
midtermAverage = midtermAverage/total;
finalAverage = finalAverage/total;
projectAverage = projectAverage/total;
classTotal = HmAverage*.10 + projectAverage*.40 + midtermAverage*.20 + finalAverage*.30;

print "Number of Students : " , total;
print "Number of Groups " , " " , numGroups;
print "Class Average for Homework 1 is :" , Hm1Average;
print "Class Average for Homework 2 is :" , Hm2Average;
print "Class Average for Midterm is :" , midtermAverage;
print "Class Average for Project is :" , projectAverage;
print "Class Average for Final is :" , finalAverage;
print "Class Total Average is : ", classTotal;
print "-------------------------------------------------------------------------------------";
# Note the use of associative arrays and the for(x in array) control statement!
for(x in studentNames) {
 print "Name of Student : " , x;
 print "Grade is " , studentGrades[x];
 print "------------------------------------------------------------------------------------";
}
```

Chapter 4

Project Plan

In this chapter we will describe our project plan and implementation.

4.1 Coding Style

The following is an example of the coding style used in this project:

```
/*
 * File: <FileName>
 * XAWK (c) 2004 - 2005
 * Date
 *
 * Authors: Name of Author [email address]
 *          Name of Author [email address]
 */

package edu.columbia.xawk;

import import.packages.go.here;

/**
 * JavaDoc style description of class.
 *
 */
public class ClassName{
  //static variables
  static varName;

  //instance variables
  private varName;

  //constructors
  /**
   * JavaDoc style description of constructor.
   */
  public ClassName(){

  }

  //public methods
  /**
   * JavaDoc style description of method.
   */
  public getName(){

  }
  //protected methods - same as public
  //private methods - same as public
  //static methods - same as public
}
```

4.2 Project Design

Figure 4.1 shows the basic interaction between the components of *XAWK*. The runtime infrastructure includes the parser, walker, and interpreter. Whenever the runtime infrastructure encounters a pattern statement, it first finds or creates the appropriate XML Processing Engine for the xml document. It then asks the XML Processing Engine for the current element and path. The pattern statement is then queried to see if the path satisfies the current pattern, if so the current XML element is processed by the user's supplied *XAWK* program. Otherwise, different patterns may attempt to match against this path and, failing that, the runtime infrastructure will ask the XML Processing Engine to advance within the xml document so that it can attempt to match against a new path.

Because the SAX parser works on callbacks, it assumes that it *drives* xml file processing. *XAWK*, however, requires control over how the xml file is processed. To gain control over xml processing we run the sax parser in a separate thread and connect that thread to our main processing thread via a *thread safe queue* containing the elements returned by the sax parser. As our XML Processing Engine *advances* through a file it dequeues elements from the fixed length queue. The queue serves to throttle the SAX parser and gives *XAWK* control over xml parsing.

One important implementation detail of this queue is that the xml elements it contains are copies of the elements returned by the SAX parser. This is because the SAX parser reuses the data structures it passes to its callback handler. Placing these data structures in the queue would allow their contents to be changed as the SAX parser continues processing the XML file.

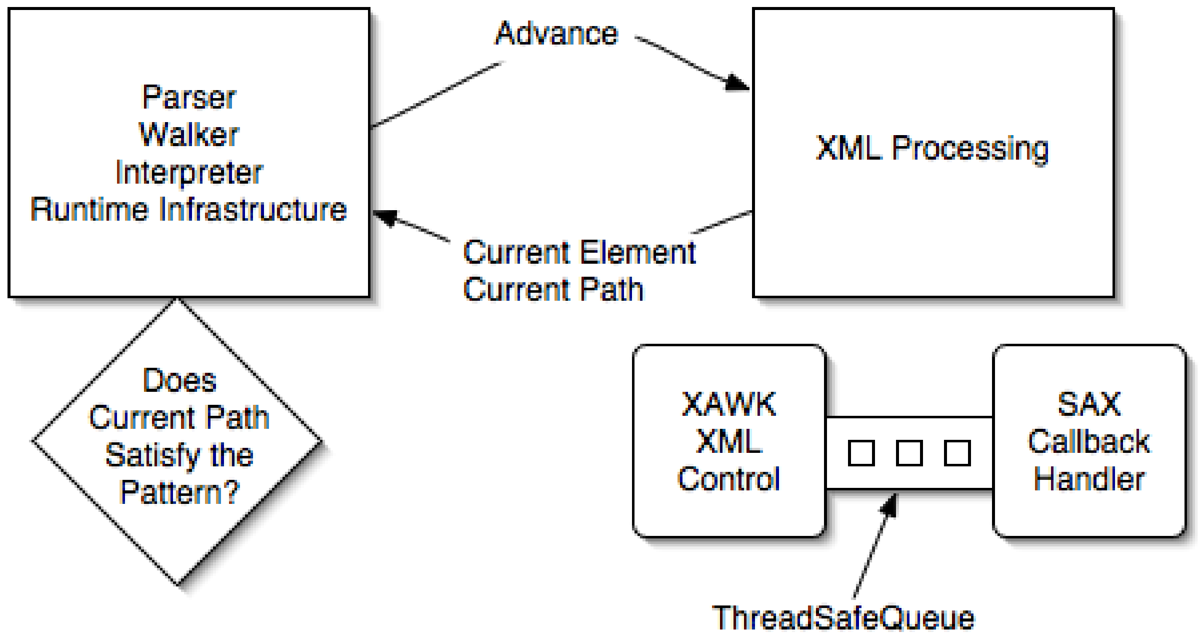

Figure 4.1: XAWK Component Interaction

Figure 4.2 shows the design of our variables and symbol table. This design is a good example of employing an additional level of indirection to solve a problem. In this case, our challenge was allowing the same variable to be assigned both strings and numbers during its lifetime. We solve this problem by having each variable contain a *XawkDataType* that can be replaced without altering the variable's name or place in the symbol table. Also, making the *XawkArray* contain variables, though seemingly circular, allows our array entries to contain arbitrary data types.

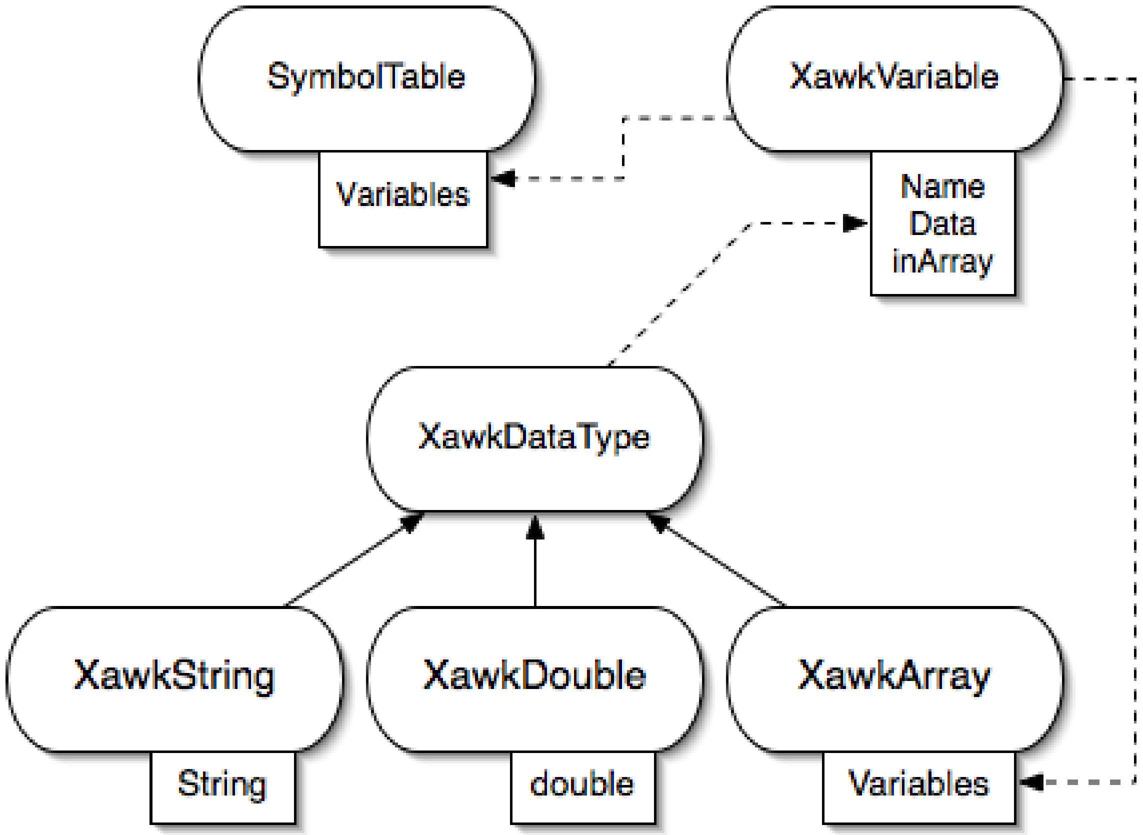

Figure 4.2: XAWK Symbol Table and Variables

4.3 Implementation Plan

We initially divided the work into XML processing and Runtime Infrastructure. After building the basic XML processing engine, we turned all of our attention to finishing the Runtime Infrastructure so that we could integrate the XML component.

The grammar and walker for our language proved challenging because the AWK language, which was our model, presents some significant nondeterminism problems. Further, introducing pattern statements as well as the control constructs to manage xml parsing were non trivial.

4.4 Testing Plan

Since *XAWK* simulates the AWK language in the ways how variables are initialized and promoted and how strings are concatenated, excessive amount of tests are required to ascertain the flexibility of our language while maintain certain restrictions on the syntax and logic. Indeed, our testing suite is divided into three stages, which are Logic / Syntax Testing, Errors Handling, and finally Compiling and Program Testing.

4.4.1 Logic / Syntax Testing

Recall how the AWK language deals with variable initialization, type promotion, and string concatenation:

Variables are initialized and declared when they are first used; Variable types are promoted by performing arithmetic operations with numeric; Strings are concatenated by juxtaposition.

Regarding the flexibility of AWK and our *XAWK* language, we have created several big test files to test as many inputs as possible in addition to the special scenarios described above. The following is a section of a large test file `bigtest.xawk` located at `/xawk/programs/`

```
print "-------------- ASSIGNMENT OPERATIONS --------------";
print "";

assign1 = 10;
assign2 = a + 10;
assign3 = "5a" + 5;
assign4 = "4.5b" + 5.5;
assign5 = "e" + 10;

print "Assign statements 1-5 should print 10 and :";
print "assign1 = ", assign1;
print "assign2 = ", assign2;
print "assign3 = ", assign3;
print "assign4 = ", assign4;
print "assign5 = ", assign5;
print "";

......

// assignplus, assignminus, assignmult, assigndiv not shown here

print "-------------- ARITHMETIC OPERATIONS --------------";
print "";

plus1 = 5e0 + 5;
plus2 = 10 + a;
plus3 = "10a" + 0.0;
plus4 = "" + "10e0john";
plus5 = "5?" + 5;

print "Plus statements 1-5 should print 10 and :";
print "plus1 = ", plus1;
print "plus2 = ", plus2;
print "plus3 = ", plus3;
print "plus4 = ", plus4;
print "plus5 = ", plus5;
print "";

...

// minus, mult, div, mod not shown here

print "-------------- INCREMENT/DECREMENT OPERATIONS --------------";
print "";

x = 10;
inc1 = x++;
x = 10;
inc2 = ++x;
inc3 = 10++;
inc4 = ++10;
inc5 = "10abc"++;
inc6 = ++"10e0";
inc7 = ""++;
inc8 = ++"";

print "Increment statement 1 should be 10 and inc1 = ", inc1;
print "Increment statement 2 should be 11 and inc2 = ", inc2;
print "Increment statement 3 should be 10 and inc3 = ", inc3;
print "Increment statement 4 should be 11 and inc4 = ", inc4;
print "Increment statement 5 should be 10 and inc5 = ", inc5;
print "Increment statement 6 should be 11 and inc6 = ", inc6;
print "Increment statement 7 should be 0  and inc7 = ", inc7;
print "Increment statement 8 should be 1  and inc8 = ", inc8;
print "";

...

// Decrement not shown here

print "-------------- UNARY OPERATIONS ----------------";

...

print "-------------- BOOLEAN OPERATIONS --------------";

...
print "-------------- ARRAY OPERATIONS --------------";

print "Testing indexing with different data types...";
print "";
# Indexing with different types of constants:
```

```
x[1] = "Mary";
x["two"] = "had";
x[5.5] = "a";

# Indexing with different types of variables
a = 169;
b = "three";
c = 3.14;

x[a] = "little";
x[b] = "lamb";
x[c] = ".";

# Test recall
# prints  Mary had a little lamb
print x[1], " ", x["two"], " ", x[5.5], " ", x[a], " ",x[b], x[c];
print "";

print "Testing different types in the array";

# Placing different value types into an array

x[1] = "one";
x[2] = 2;
x[3] = 3.0;

# Plalcing different variable types into an array
d = "four";
e = 5;
f = 6.0;
x[7] = "good";

x[4] = d;
x[5] = e;
x[6] = f;
# Placing Array in Array
x[8] = x[7];

print x[1]; # Prints one
print x[2]; # Prints 2
print x[3]; # Prints 3
print x[4]; # Prints four
print x[5]; # Prints 5
print x[6]; # Prints 6
print x[7]; # Prints good
print x[8] #  Prints good

# Testing with Array comparison
# Prints 5
```

```
x[10] = 5;
if(x[10] < 10) {
 print x[10] ;
}

print "-------------- STRING CONCATENATION --------------";

x = "hello" "world";
print "should be helloworld: ", x;
y = 5 "hi";
print "should be 5hi:", y;
z = 5.5 "prashant";
print "should be 5.5prashant:", z;
a = "prashant" 5;
print "should be prashant5:", a;
b = "john" 5.5;
print "should be john5.5:", b;
c = 5 5;
print "should be 55:", c;
d = 5.5 5.6;
print "should be 5.55.6:", d;
print "Should be 2far";
x = 2 "far";
print x;

print "Should be 25";
print 2 5;

print "Should be foo4";
print "foo" 4;
print "";

print "Should be foo4.5";
print "foo" 4.5;

print "mary" " " "had" " " "a" " " "little" " " "lamb";

print "should be foo2.6bar";
x = "foo" 2.6 "bar";
print x;

y = "foo";
x = "bar";
w = 4;
z = 5.5;

print "Should be foobar";
print y x;

print "Should be 45.5";
print w z;

a = w x y z;

print "Should be 4barfoo5.5";
print a;

...
print "------------- CONTROL STATEMENTS -------------";

x[1] = 1;
x[2] = 2;
x[3] = 3;

for(a in x){
print a;
}

a = 0;
if(a == 1){
print "you fail";
}else{
print "you win";
}

a = 1;
if(a == 0){
print "you fail";
}else{
print "you win";
}

if(a){
print "you win!";
}
```

More sections in the file include Output Statements, Concatenation Statements, Control Statements, Array Operations, and Pattern Searching. Notice that we are only checking the correctness of the outputs here, while exceptions and errors are handled in the second stage. An example of a logic and syntax testing would be the Short-Circuit Evaluation under the Boolean Operations

```
bool2 = 0;
if (10 > 9.9 || 5/0 == 5)
    bool2 = 1;  # Short-Circuit Evaluation
```

We were not aware of this issue until the second stage, Error Handling, throws out an exception. Now it works fine in our language without generating an error.

4.4.2 Error Handling

The purpose of this stage is to catch as many errors as possible and throw a corresponding error to each error. While we take care of everything that should work in our language in the first stage, we want to make sure that our language has a lower and upper limit in this stage. Anything that is out of the limit generates an error.

Since the program terminates once an error is found, it would not be efficient for us to test each possible error by modifying the source code every time. The way we do this is to write a shell script that runs a list of files that throw corresponding exceptions under the `/xawk/programs/exceptions/` directory. The body of the script test.sh is given below

```
echo ------------------- Exceptions testing -----------------------
for i in ‘grep -l ’ ’ $HOME/xawk/programs/exceptions/*.xawk‘
do
     echo
     echo Executing $i ....
     java edu.columbia.xawk.Xawk $i
done
```

This script, located at `/xawk/`, runs each `.xawk` file in the exceptions directory. Since each test file contains one exact error, running this script allows us to see all the exceptions generated. Test files include files that generate division by zero error, invalid assignment, invalid if statement, etc.

4.4.3 Compiling and Program Testing

In the final stage, we handle compiling errors and large program testing. We have combined all the necessary compilation into an ANT Build process, where everything is compiled correspondingly as described in the `build.xml` file, located at `/xawk/` directory

```
  <?xml version="1.0" ?>
- <project name="xawk" default="compile" basedir=".">
- <!--
 set global properties for this build
  -->
  <property name="build" value="./classes" />
  <property name="src" value="./src" />
  <property name="lib" value="./lib" />
- <!--
- Must be changed to reflect individual antlr location
  -->
  <property name="antlr" value="/Users/johnc/Library/antlr-2.7.4" />
  <property name="xawk" value="${src}/edu/columbia/xawk" />
  <property name="programs" value="./programs" />
- <!--
 compiler
  -->
- <!--
 <property name="build.compiler" value="jikes" />
```

```
  -->
- <target name="init">
- <!--
 Create the time stamp
  -->
  <tstamp />
  <mkdir dir="${build}" />
  </target>
- <target name="compile" depends="init" description="Builds XAWK">
  <echo message="Building XAWK" />
- <!--
 Compile the java code from ${src} into ${build}
  -->
- <java classname="antlr.Tool" dir="${xawk}" fork="yes">
  <arg value="xawk.g" />
- <classpath>
  <pathelement path=".:${antlr}/antlr.jar" />
  </classpath>
  </java>
- <java classname="antlr.Tool" dir="${xawk}" fork="yes">
  <arg value="walker.g" />
- <classpath>
  <pathelement path="${antlr}/antlr.jar" />
  </classpath>
  </java>
  <javac srcdir="${src}" destdir="${build}" classpath="${antlr}/antlr.jar" />
  </target>
```

The XML code above takes care of all the compilation, while the code segments below handles separation execution of the test file, depending on which test file we want to run test on

```
- <target name="run" depends="compile">
  <echo message="Running test.xawk" />
- <java classname="edu.columbia.xawk.Xawk" fork="yes">
  <arg value="${programs}/test.xawk" />
- <classpath>
  <pathelement path="${build}:${antlr}/antlr.jar" />
  </classpath>
  </java>
  </target>

- <target name="controltest" depends="compile">
  <echo message="Running control_test.xawk" />
- <java classname="edu.columbia.xawk.Xawk" fork="yes">
  <arg value="${programs}/control_test.xawk" />
- <classpath>
  <pathelement path="${build}:${antlr}/antlr.jar" />
  </classpath>
  </java>
  </target>

...

// Other tests not shown
```

Besides the normal compilation procedures, our ANT Build also includes convenient tools such as clean and archive, which allows developers to remove unnecessary class files and archive old source code —

```
- <target name="clean">
- <!--
 Clean compiled crap
  -->
  <delete dir="${build}" />
  <delete file="xawk_archive.tar.gz" />
  </target>
- <target name="archive" depends="clean">
- <!--
 Archive the Clean Code
  -->
  <tar tarfile="xawk_archive.tar.gz" compression="gzip" basedir="." />
  </target>
  </project>
```

All the errors that correspond to compilation will be caught during the ANT build.

In addition to compilation, we have also run tests on large programs such as the one described in the Tutorial section, regarding the xml file with sample grades. Please refer to the Tutorial section for the source code and description of these examples.

Chapter 5

Lessons Learned

In this chapter we present the lessons learned from this project.

5.1 John's Lessons

Undertaking any large project requires significant planning. In the case of this project, careful planning paid off in a relatively smooth implementation. The code was appropriately modularized and interfaces defined in advance. In addition to planning, it is also important to recognize situations that call for significant refactoring or even a complete rewriting of the code. During this project we encountered such a situation, but by rewriting the interpreter code we created a cleaner implementation that the made interactions between variables as well as between the interpreter and the xml parsing module much cleaner.

The most important lesson from this project is the importance steady work throughout the semester. Because we started early, the implementation was carefully thought out and executed rather than hurried. This gave us an opportunity to try different approaches to problems without the stress of a soon approaching deadline.

5.2 Gabriela's Lessons

This project is supposed to be a class project, so it starts out small. Even if you want to keep it small, it is impossible, because as long you start giving it functionality, somehow it grows by itself. So here it is a pretty consistent project I might say.

But everything had to start with the simplest thing: the print statement and then on top of it start building the language. As long as you can do the assignment everything comes so smooth. We might say that to get rid of the comments in the lexer is trivial, but not when the comment eats the EOF.

So basic functionality from the beginning and a lot of testing, then the control statements in the walker, got us to the phase when we could implement the most important part of the project: the pattern matching. This was the most challenging stage of the project, and gave us a lot of satisfaction.

About working in a group, I might say that for a project like this it is impossible to split completely the tasks that each member has to do. This is because there is a strong relationship between the grammar, the walker and the interpreter and any change in any of these might lead to changes in all of them.

5.3 Shi Tak's Lessons

Since this is a team project, team atmosphere is essential in completing this project. While the division of work may not always be equal, it is always important for team member to proceed to other tasks after he or she finishes his / her assigned task. Sitting here waiting for the team leader to assign another task decelerates the progress of the entire project, while work undone will need to be picked up by another team member who may already have other assigned tasks. Relying on other team members does not help one from learning from the project, as the ultimate goal of a team project is for each one to learn something new besides the accomplishment of the project. A project will only be successful if everyone dedicates to it with timse and enthusiasm.

5.4 Prashant's Lessons

I learned that the key requirement for successfully designing a compiler is planning. This was understood very well by the team members and executed under the guidance of our team lead. I learnt the importance of testing when developing a compiler, after developing the compiler XAWK, it was testing that underlined some bugs which we were able to solve in time. The final paper as well as development of the compiler was made easier by including the whitepaper and LRM in the beginning.

Chapter 6

Code Listing

This chapter includes a listing of all source code used in the final version of *XAWK* submitted for COMS4115, *Programming Languages and Translators* at Columbia University, Fall 2004. The entire *XAWK* project is contained in the `edu.columbia.xawk` package.

6.1 ANTLR Input Files

6.1.1 xawk.g

```
// *******************************************************
// File: xawk.g
// XAWK (c) 2004-2005
// Authors:  Gabriela Cretu [gcretu@cs.columbia.edu]
// John Cieslewicz  [johnc@cs.columbia.edu]
//
// Description:
// This file contains the antlr grammar for the XAWK lexer
// and parser.
// *******************************************************

header{
package edu.columbia.xawk;
}

{
import java.util.*;
}

////////////////////////
// *Define the Parser* //
////////////////////////
class XawkParser extends Parser;
options {
    //lookahead
    k=2;
    //construct an AST during parsing
    buildAST = true;
    //Export the token types for the tree
    exportVocab = Xawk;
    //Dont automatically catch every exception
    defaultErrorHandler = false;
}

tokens{
UPL;
UMN;
ULNOT;
LINC;
LDEC;
RINC;
RDEC;
ARRAY;
STATEMENT;
PATTERN_STATEMENT;
PATTERN_STATEMENT_DES;
PATTERN_STATEMENT_BLOCK;
MATCHNEXT;
ENDOFPATTERN;
MATCHALLATTR;
MATCHATTR;
DOC;
DOC_DESC;
CONCAT;
TEXT;
LABEL;
INIT;
REL;
INC;
INDEX;
ATREF;
RINC_STATEMENT;
LINC_STATEMENT;
RDEC_STATEMENT;
LDEC_STATEMENT;
FOR_COND;
FOR_EXP;
FOR_IN;
TEXT_S;
}

//defining a program
program:
(statement)* EOF! {#program = #([STATEMENT, "PROGRAM"],program);}
;

//defining a statement
statement:
        label_statement
        | break_statement SEMICOLON!
        | cont_statement SEMICOLON!
        | do_statement SEMICOLON!
        | if_statement
        | for_statement
        | io_statement SEMICOLON!
        | while_statement
        | statement_block
        | pattern_statement_block
        | exp_statement
        ;

exp_statement:
exp SEMICOLON!;

// a statement block is separated because it is used in isolation with respect
// to a pattern statement.
statement_block:
LCUR! (statement)* RCUR!
{#statement_block = #([STATEMENT,"STATEMENT"],statement_block); }
;

pattern_statement_block:
(
( options {greedy=true;}:
          pattern_statement_descendent | pattern_statement)+
{#pattern_statement_block = #([PATTERN_STATEMENT_BLOCK,"PATTERN_STATEMENT_BLOCK"],pattern_statement_block); }
);

//defining the label statement
label_statement:
        ID COLON! statement
        {#label_statement = #([LABEL, "LABEL"], label_statement);}
;

//definig the break statement
break_statement:
        "break"^ LBRACKET! ID RBRACKET!;

//defining the continue statement
cont_statement:
        "continue"^ LBRACKET! ID RBRACKET!;

//defining the do statement
do_statement:
        "do"^ statement "while"! LPAREN! exp RPAREN!;

//definig the if statement
if_statement:
        "if"^ LPAREN! exp RPAREN! statement
        (options {greedy = true;}: "else"! statement)?
        ;

//defining the for statement
for_statement:
        "for"^ LPAREN! for_condition RPAREN! statement
        ;

for_condition:
        ((exp)? SEMICOLON!) => (for_exp) | (exp {#for_condition = #([FOR_IN,"FOR_IN"],for_condition); })
        ;

for_exp:
init (SEMICOLON! rel SEMICOLON! inc)? {#for_exp = #([FOR_COND,"FOR_COND"],for_exp); }
;

init:
(exp)? {#init = #([INIT,"INIT"],init); }
;

rel:
(exp)? {#rel = #([REL,"REL"],rel); }
;

inc:
(exp)? {#inc = #([INC,"INC"],inc); }
;

//defining the IO statement
io_statement:
        "print"^ io;


io:
exp (COMMA^ exp)*;

//defining while statement
while_statement:
        "while"^ LPAREN! exp RPAREN! statement;

//defining a pattern statement that begins with "match descendent"
pattern_statement_descendent:
(MATCHDESCENDENT!)
pattern_tail
statement_block
{#pattern_statement_descendent = #([PATTERN_STATEMENT_DES,"PATTERN_STATEMENT_DES"],pattern_statement_descendent); };

//defining a pattern statement that begins with "match child"
pattern_statement:
(FORWARDSLASH!)
pattern_tail
statement_block
{#pattern_statement = #([PATTERN_STATEMENT,"PATTERN_STATEMENT"],pattern_statement); };

//defines the pattern tail rule used to construct the pattern
pattern_tail:
("doc"! LPAREN! STRINGLITERAL RPAREN! ((FORWARDSLASH! {#pattern_tail = #([DOC,"DOC"],pattern_tail); }) | (MATCHDESCENDENT! {#pattern_tail = #([DOC_DESC,"DOC_DESC"],pattern_tail); })) pattern_tail)
| ( (ID (FORWARDSLASH!{#pattern_tail = #([MATCHNEXT,"MATCHNEXT"],pattern_tail); }|MATCHDESCENDENT^) pattern_tail )
| (ID_PATTERN (FORWARDSLASH! {#pattern_tail = #([MATCHNEXT,"MATCHNEXT"],pattern_tail); }|MATCHDESCENDENT^) pattern_tail)
| (STAR
((FORWARDSLASH!{#pattern_tail = #([MATCHNEXT,"MATCHNEXT"],pattern_tail); }|MATCHDESCENDENT^) pattern_tail)?)
| pattern_termination
)
;

//defines the conditions which may terminate a pattern
pattern_termination
:
(AT! ID (FORWARDSLASH!{#pattern_termination = #([MATCHATTR,"MATCHATTR"],pattern_termination); }))
| (AT! STAR! (FORWARDSLASH!{#pattern_termination = #([MATCHALLATTR,"MATCHALLATTR"],pattern_termination); }))
| ("text"! LPAREN! RPAREN! (FORWARDSLASH!{#pattern_termination = #([TEXT,"TEXT"],pattern_termination); }))
| /*nothing*/ {#pattern_termination = #([ENDOFPATTERN, "ENDOFPATTERN"], pattern_termination);}
;

//defining the expressions
exp:
  exp_next ((ASSIGN^ | ASSIGNPLUS^ | ASSIGNMINUS^ | ASSIGNMULT^ | ASSIGNDIV^) exp_next)*;
exp_next:
cond_exp (QMARK^ exp_next COLON! exp_next)? ;
cond_exp :
     logic_term (OR^ logic_term)*;
logic_term:
     logic_factor (AND^ logic_factor)*;
logic_factor:
     rel_term ("in"^ rel_term)*;
rel_term:
        concat_term ((GT^ | GE^ | LS^ | LE^ | E^ | NOTE^) concat_term)*;
concat_term:
aritm_term ( {#concat_term = #([CONCAT, "CONCAT"], concat_term);} aritm_term)*;
aritm_term :
        mult_term (options {greedy=true;}: (PLUS^ | DASH^ ) mult_term)*;
mult_term :
        unary_term ((STAR^ | FORWARDSLASH^ | MODULUS^) unary_term)*;
unary_term :
        PLUS! postpos_term { #unary_term = #([UPL,"UPL"], unary_term); }
        | DASH! postpos_term { #unary_term = #([UMN,"UMN"], unary_term); }
        | LOGICALNOT! postpos_term { #unary_term = #([ULNOT,"ULNOT"], unary_term); }
        | AT! postpos_term {#unary_term = #([ATREF,"ATREF"], unary_term);}
        | postpos_term;
postpos_term:
        prepos_term (options {greedy=true;}: INCREMENT! {#postpos_term = #([RINC,"RINC"],postpos_term);} | DECREMENT! {#postpos_term = #([RDEC,"RDEC"],postpos_term);})?;
prepos_term:
        INCREMENT! atom { #prepos_term = #([LINC,"LINC"], prepos_term);}
        | DECREMENT! atom { #prepos_term = #([LDEC,"LDEC"], prepos_term); }
        | atom;
atom:
        STRINGLITERAL | ID | DoubleConst | Integer | (LPAREN! exp RPAREN!) | (ID LBRACKET! exp RBRACKET!{ #atom = #([ARRAY,"ARRAY"], atom);});

///////////////////////////////////////////////
// *define the Lexical part of the grammer* //
//////////////////////////////////////////////
class XawkLexer extends Lexer;

options {
    charVocabulary = '\u0003'..'\uFFFF';
    exportVocab = Xawk;
    k=2;
    testLiterals = false;
}

//defining all the operators

INCREMENT:              "++";
DECREMENT:              "--";
PLUS:                   '+';
DASH:                   '-';
LOGICALNOT:             '!';
STAR:                    '*';
FORWARDSLASH:           '/';
MODULUS:                '%';
GT:                     '>';
GE:                     ">=";
LS:                     '<';
LE:                     "<=";
NOTE:                   "!=";
E:                      "==";
AND:                    "&&";
OR:                     "||";
QMARK:                  '?';
COLON:                  ':';
ASSIGN:                 '=';
ASSIGNPLUS:             "+=";
ASSIGNMINUS:            "-=";
ASSIGNMULT:             "*=";
ASSIGNDIV:              "/=";
SEMICOLON:              ';';
MATCHDESCENDENT:        "//";
protected
PERIOD:                 '.';
WS:                     (' ' | '\t' | '\n' {newline();} | '\r')
                        {$setType(Token.SKIP);};
AT:                     "@";
COMMA: ",";

//a "pattern id" may be found in a xml path, but cannot be a standard identifier
ID_PATTERN
       :
       '\''!
        ((LETTER)(LETTER|DIGIT|COLON|PERIOD|DASH|'_')*)
        '\''!
       ;

//defining ID
ID options{ testLiterals = true; }
    : LETTER
      (LETTER | '_' | (DIGIT))*
    ;

//defining string literals and numeric literals
STRINGLITERAL
    : '"'!
      ( (ESCAPE |~('"' | '\n' | '\r'|'\\' )) | ('"'! '"'))*
      '"'!
    ;

//technique found at http://www.doc.ic.ac.uk/lab/secondyear/Antlr/lexer.html
ESCAPE
    :'\\'
        ( 'n' { $setText('\n'); }
        | 'r' { $setText('\r'); }
        | 't' { $setText('\t'); }
        | 'b' { $setText('\b'); }
        | '"' { $setText('"'); }
        | 'f' { $setText('\f'); }
        | '\\'{ $setText('\\'); }
        | '\''{ $setText('\''); }
        | {char unicode;} 'u' unicode = UNICODE {$setText(unicode);}
        );

LBRACKET                    :'[';
RBRACKET                    :']';
LPAREN                      :'(';
RPAREN                      :')';
LCUR                        :'{';
```

```
RCUR                        :'}';

//note the check for \uFFFF, 2's compliment -1 to check for EOF. Resolves bug where
//comment on last line of file caused program to hang since EOF token was never
//generated.
COMMENT
    : '#' (options {greedy=false;}:
            ~('\n'|'\r'|'\uFFFF')
        )*(('\n'{newline();}|'\r') { $setType(Token.SKIP);} | ('\uFFFF' {$setType(Token.EOF_TYPE);}))
    ;

NUMBER
    :(DIGIT)+
        (((PERIOD)(DIGIT)*(EXPONENT)?){ $setType(DoubleConst); }
        |(EXPONENT) {$setType(DoubleConst); }
        |/*nothing*/ {$setType(Integer); }
        )
    ;

FRACTIONALNUMBER
    :((PERIOD)(DIGIT)*(EXPONENT)?){ $setType(DoubleConst); }
    ;

protected
UNICODE returns [char unicodeChar] { String unicode = new String();}
       :(h1:HEXDIGIT!){unicode += h1.getText();  }(h2:HEXDIGIT!){unicode += h2.getText();}(h3:HEXDIGIT!){unicode += h3.getText();}(h4:HEXDIGIT!){unicode += h3.getText(); unicodeChar = (char)(java.l

protected
DIGIT:
        '0'..'9';

protected
LETTER:
        'A'..'Z' | 'a'..'z';

protected
HEXDIGIT:
        'a'..'f' |'A'..'F'| DIGIT;

protected
EXPONENT
    : ('e'|'E')('+'|'-')? (DIGIT)+
    ;
```

6.1.2 walker.g

```
 // ***************************************************************
 // File: walker.g
 // XAWK (c) 2004-2005
 // Authors: Gabriela Cretu [gcretu@cs.columbia.edu]
 // John Cieslewicz [johnc@cs.columbia.edu]
 //
 // Description:
 // This file contains the antlr rules to create the AST walker
 // for XAWK.
 // ***************************************************************

header{
package edu.columbia.xawk; //place the walker in the edu.columbia.xawk package
}

{
import java.io.*;
import java.util.*;
}

// Defines the AST Walker
class XawkWalker extends TreeParser;
options
{
importVocab = Xawk;  //use the XAWK vocab
//ASTLabelType = "LineNumberAST";
}

//instance variables of the XawkWalker class
{
private boolean makingPattern = false;
private XawkInterpreter xint = new XawkInterpreter();
private XawkVariable null_data = new XawkVariable("");
private Element null_element = new Element(Element.NULL_ELEMENT);
private boolean first = true;
private boolean no_expr = false;
private boolean next_step =false;
private boolean if_cond = true;
private boolean in_in = false;
private boolean run = false;
private boolean infinite_loop = false;
private boolean first_time_in_for = false;
private boolean attribute = false;
private boolean file = false;
private boolean all_null = false;
private LinkedList path = new LinkedList();
private LinkedList leaves = new LinkedList();
private LinkedList roots = new LinkedList();
private boolean in_if = false;
private boolean exception = false;
private String in_alias = null;
private XawkVariable current_array_element = null;
private Iterator array_iterator = null;
private boolean no_inc = false;
}

//The expr rule is the main rule for evaluating expressions and statements.
expr returns [ XawkVariable x ]
{
    XawkVariable a=null,b=null,c=null;
    Element d,e,f;
    x = null_data;
String out = "";
}

:#(OR first_term1:. second_term1:.)
{
a = expr(#first_term1);
XawkDataType xdt = a.getData();
if (xdt instanceof XawkDouble)
{
double val = ((XawkDouble)xdt).getValue();
if (val==0.0)
{
b = expr(#second_term1);
x = xint.or(a,b);
}
if(val == 1.0)
{
x = a;
}
}
else
throw new XawkException("Not a correct OR expression.");
}
| #(AND first_term:. second_term:.)
{
a = expr(#first_term);
XawkDataType xdt = a.getData();
if (xdt instanceof XawkDouble)
{
double val = ((XawkDouble)xdt).getValue();
if (val == 1.0)
{
b = expr(#second_term);
x = xint.or(a,b);
}
if(val == 0.0)
{
x = a;
}
}
else
throw new XawkException("Not a correct AND expression.");
}
| #(ULNOT a=expr)   { x = xint.logicalnot(a);}
  | #(GT a=expr b=expr) { x = xint.gt(a, b); }
| #(GE a=expr b=expr)    { x = xint.ge(a, b); }
| #(LS a=expr b=expr)    { x = xint.ls(a, b); }
| #(LE a=expr b=expr)    { x = xint.le(a, b); }
    | #(NOTE a=expr b=expr)  { x = xint.note(a, b); }
    | #(CONCAT a=expr b=expr) {x = xint.concat(a,b);}
| #(E a=expr b=expr)     { x = xint.equals(a, b); }
| #(PLUS a=expr b=expr)  { x = xint.plus(a, b); }
  | #(DASH a=expr b=expr)  { x = xint.minus(a, b); }
| #(STAR a=expr b=expr)  { x = xint.star(a, b); }
   | #(FORWARDSLASH a=expr b=expr)  { x = xint.div(a, b); }
| #(MODULUS a=expr b=expr)       { x = xint.modulus(a, b); }
| #(UPL a=expr) { x = xint.unaryPlus(a);}
| #(UMN a=expr)    { x = xint.unaryMinus(a);}
| #(ASSIGN a=expr b=expr)        { x = xint.assign(a,b);}
| #(ASSIGNPLUS a=expr b=expr)    { x = xint.assignplus(a, b); }
| #(ASSIGNMINUS a=expr b=expr)   { x = xint.assignminus(a, b); }
| #(ASSIGNMULT a=expr b=expr)    { x = xint.assignmult(a, b); }
   | #(ASSIGNDIV a=expr b=expr)     { x = xint.assigndiv(a, b); }
| #(RINC a=expr)  { x = xint.rincrement(a); }
| #(LINC a=expr )            { x = xint.lincrement(a); }
| #(RDEC a=expr ) { x = xint.rdecrement(a); }
| #(LDEC a=expr ) { x = xint.ldecrement(a); }
| #(INCREMENT a=expr) { x = xint.inc_stat(a); }
| #(DECREMENT a=expr)  { x = xint.dec_stat(a); }
| #(NOTHING a=expr b=expr) { x = xint.concat(a,b);}
| #(QMARK cond_exp:. stat1:. stat2:.)
{
if (xint.getForCondition(expr(#cond_exp)))
```

```
x = expr(#stat1);
else
x = expr(#stat2);
}

| #(ATREF a = expr)
{
String index = a.getVarName();
XawkVariable temp = xint.getCA();
XawkArray xa = (XawkArray)temp.getData();
if(xa.isPresent(index))
x = xa.getVariable(index);
else
{
XawkVariable xv = new XawkVariable(index);
xv.setArrayElementFlag(true);
xv.setData(new XawkString(""));
xa.addVariable(xv);
x = xv;
}
}

| intnum:Integer { x = xint.getNumber( intnum.getText() );}
| doublenum:DoubleConst { x = xint.getNumber( doublenum.getText() ); }
| str:STRINGLITERAL      { x = xint.getString( str.getText() ); }
| id:ID
{
if(in_in)
{
if(in_alias == null)
{
in_alias = id.getText();
x = null_data;
}
else
{
String idText = id.getText();
if(in_alias.equals(idText))
{
x = current_array_element;
}
else
{
a = xint.getVariable(idText);
x = a;
}
}
}
else
{
a = xint.getVariable( id.getText() );
x = a;
}
}

| #(ARRAY a=expr b=expr)
{  //a should be the variable returned from "getVariable()"
x = xint.getArrayVariable(a, b);
}
| #(STATEMENT (statement:.  { x = expr(#statement);})*)

| #("while" wexp:. whilestat:. )
{
while(xint.getForCondition(expr(#wexp)))
{
x=expr(#whilestat);
}
}

| #("do" dostat:. exp:.)
{
while(xint.getForCondition(expr(#exp)))
{
x=expr(#dostat);
}
}
| #("in" a=expr b=expr)
{
XawkDataType xdt = b.getData();
if(! (xdt instanceof XawkArray))
throw new XawkException("'in' expression can only be applied to arrays.");

XawkArray array = (XawkArray)xdt;
array_iterator = array.iterator();
}

| #(FOR_IN first_part:.)
     {
     in_in = true;
     expr(#first_part);
      }
```

```
    | #(FOR_COND init:. rel:. inc:.)
     {
     if (first_time_in_for)
     {
     expr(#init);
     first_time_in_for = false;
     no_inc = true;
     }
     if (no_inc) no_inc = false;
     else expr(#inc);
     expr(#rel);

     }
    | #(INIT (initial:.)?)   { if (initial != null) x = expr(#initial); else x = null;}
    | #(REL (relational:.)?)
    {
    if (relational != null)
    {
    b = expr(#relational);
    XawkDataType xdt = b.getData();
if (xdt instanceof XawkDouble)
{
double val = ((XawkDouble)xdt).getValue();
if (val==0.0)
{
next_step = false;
}
if(val==1.0)
{
next_step = true;
}
}
else throw new XawkException("Not a correct for condition");
    }
    else
    {
    next_step = true;
    x = null;
    }
    }
    | #(INC (increment:.)?) { if (increment != null) x= expr(#increment); else x = null;}

| #("for" for_cond:. forstat:.)
{
first_time_in_for = true;
expr(#for_cond);
if (!in_in)
{
while(next_step)
{
x=expr(#forstat);
expr(#for_cond);
}
}
else
{
while(array_iterator.hasNext())
{
current_array_element = (XawkVariable)array_iterator.next();
x = expr(#forstat);
}
array_iterator = null;
in_in = false;
in_alias= null;
current_array_element = null;
}
}


| #("if" expression:. ifstat:. (elsestat:.)?)
{
a = expr(#expression);
XawkDataType xdt = a.getData();
if (xdt instanceof XawkDouble)
{
double val = ((XawkDouble)xdt).getValue();
if (val==0.0 && #elsestat!=null)
{
x=expr(#elsestat);
}
if(val!=0.0)
{
x=expr(#ifstat);
}
}
else if (xdt instanceof XawkString)
{
b = xint.evaluate(a);
XawkDataType xdt1 = b.getData();
double val = ((XawkDouble)xdt1).getValue();
if (val==0.0 && #elsestat!=null)
{
```

```
x=expr(#elsestat);
}
if(val!=0.0)
{
x=expr(#ifstat);
}
}
else
throw new XawkException("Not a correct if condition");
}

| #("print" a=expr)
    {xint.printOutput(a); xint.println();}

| #(COMMA a=expr x=expr)
{ xint.printOutput(a);}

| #(PATTERN_STATEMENT_BLOCK pattern_stat_block:.)
{
LinkedList oldPath = path;
LinkedList oldRoots = roots;
LinkedList oldLeaves = leaves;

path = new LinkedList();
roots = new LinkedList();
leaves = new LinkedList();

xint.pushPath();
while(xint.advanceXML())
expr(#pattern_stat_block);
xint.popPath();

path = oldPath;
roots = oldRoots;
leaves = oldLeaves;
}

| #(PATTERN_STATEMENT a=expr pat_stat:.)
{


f = new Element(Element.CHILD);
f.setName("/");
path.add(f);
int n = 0;
if (((Element)roots.get(roots.size()-1)).getType() == Element.DOC_ELEMENT)
{
path.add(roots.get(roots.size()-1));
path.add(roots.get(roots.size()-2));
n = roots.size()-3;
}
else
n=roots.size()-1;

for(int i=0; i<leaves.size();i++)
{
path.add(leaves.get(i));
path.add(roots.get(n-i));
}
path.removeLast(); //removes last slash.
Pattern p = new Pattern(path);

//System.out.println("[WALKER] " + p.toString());
Match m;
if ((m = xint.match(p)) != null) {
xint.updateBuiltInVariables(m);
expr(#pat_stat);
}
path = new LinkedList();
roots = new LinkedList();
leaves = new LinkedList();


}

| #(PATTERN_STATEMENT_DES a=expr pat_stat_des:.)
{
f = new Element(Element.DESCENDENT);
f.setName("//");
path.add(f);
int n = 0;
if (((Element)roots.get(roots.size()-1)).getType() == Element.DOC_ELEMENT)
{
path.add(roots.get(roots.size()-1));
path.add(roots.get(roots.size()-2));
n = roots.size()-3;
}
else
n=roots.size()-1;

for(int i=0; i<leaves.size();i++)
{
```

```
path.add(leaves.get(i));
path.add(roots.get(n-i));
}
path.removeLast(); //removes last slash.
Pattern p = new Pattern(path);

Match m;
if ((m = xint.match(p)) != null) {
xint.updateBuiltInVariables(m);
expr(#pat_stat);
}

path = new LinkedList();
roots = new LinkedList();
leaves = new LinkedList();
}

| #(MATCHNEXT d = pattern e = pattern)
{
f = new Element(Element.CHILD);
f.setName("/");
roots.add(f);
/*System.out.println("Matching Next");*/
}

| #(MATCHDESCENDENT d = pattern e = pattern)
{
f = new Element(Element.DESCENDENT);
f.setName("//");
roots.add(f);
/*System.out.println("Matching DES");*/
}

| #(DOC  filename:.  a=expr)
{
if(!first)
throw new XawkException("doc not first");
else
{
f = new Element(Element.CHILD);
f.setName("/");
roots.add(f);
f = new Element(Element.DOC_ELEMENT);
file = true;
f.setName(pattern(#filename).getName());
roots.add(f);
}
}

| #(DOC_DESC filename2:. a=expr)
{
if(!first)
throw new XawkException("doc_desc not first");
else
{
f = new Element(Element.DESCENDENT);
f.setName("//");
roots.add(f);
f = new Element(Element.DOC_ELEMENT);
file = true;
f.setName(pattern(#filename2).getName());
roots.add(f);
}
}
;


pattern returns [ Element x ]
{
    Element d,e;
    XawkVariable a;
    x = null_element;
}
:
str:STRINGLITERAL
{
x = new Element(Element.XML_ELEMENT);
x.setName(str.getText());
if (!file)
leaves.add(x);
else
file = false;
/*System.out.println("getting the file name");*/
}

|id:ID
{
x = new Element(Element.XML_ELEMENT);
x.setName(id.getText());
if (!attribute)
leaves.add(x);
else
```

```
attribute = false;
/*System.out.println("getting id text: " + x.name);*/

}


|patid:ID_PATTERN
{
x = new Element(Element.XML_ELEMENT);
x.setName(patid.getText());
if (!attribute)
leaves.add(x);
else
attribute = false;
/*System.out.println("getting pattern id text: " + patid.getText());*/
}

|(STAR)
{
x = new Element(Element.MATCH_ALL);
x.setName("*");
leaves.add(x);
/*System.out.println("*STAR*");*/
}

|#(MATCHDESCENDENT d = pattern e = pattern)
{
x = new Element(Element.DESCENDENT);
x.setName("//");
roots.add(x);
/*System.out.println("MATCH DESCENDENT");*/
}

|#(MATCHNEXT d = pattern e = pattern)
{
x = new Element(Element.CHILD);
x.setName("/");
roots.add(x);
/*System.out.println("MATCH CHILD"); */
}

|#(MATCHALLATTR e = pattern_end)
{
x = new Element(Element.MATCH_ANY_ATTRIBUTE);
x.setName("@*");
leaves.add(x);
d = new Element(Element.CHILD);
d.setName("/");
roots.add(d);
/*System.out.println("MATCH ALL ATTR"); */first = true;
}

|#(MATCHATTR attr:. e = pattern_end)
{
x = new Element(Element.MATCH_ATTRIBUTE);
attribute=true;
x.setName(pattern(#attr).getName());
leaves.add(x);
d = new Element(Element.CHILD);
d.setName("/");
roots.add(d);
/*System.out.println("MATCH ATTR"); */
first = true;
}

|#(TEXT e = pattern_end)
{
x = new Element(Element.TEXT_ELEMENT);
x.setName("text");
leaves.add(x);
d = new Element(Element.CHILD);
d.setName("child");
roots.add(d);
/*System.out.println("TEXT");*/
first = true;
}

|#(ENDOFPATTERN d = pattern_end)
{
first = true;
}

|#(DOC d = pattern a = expr)
{
if(true)
throw new XawkException("doc not first");
}

|#(DOC_DESC d = pattern a = expr)
{
if(true)
throw new XawkException("doc_desc not first");
```

```
}
;

//Pattern End rule, a hack to allow us to have the pattern terminate with a /
// that does not have children.
pattern_end returns [Element x]
{
x = null_element;
}
: DASH {System.err.println("Dash found in path. This is a problem."); }
| /*nothing*/
;
```

6.2 Interpreter and Other Run Time Components

6.2.1 Xawk.java

```
/*
 * File: Xawk.java
 * XAWK (c) 2004- 2005
 * Created November 23, 2004
 *
 * Author: John Cieslewicz [johnc@cs.columbia.edu]
 */
package edu.columbia.xawk;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

import antlr.CommonAST;
import antlr.RecognitionException;
import antlr.TokenStreamException;

/**
 * The entry point for the XAWK program.
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @version November 23, 2004
 */
public class Xawk{
//static variable for debugging
public static boolean DEBUG = false;
//private static version number
private static String version = "0.3";

private static void executeFile(String filename){
try {
InputStream fin = new FileInputStream(filename);
XawkLexer xlex = new XawkLexer(fin);
XawkParser xparse = new XawkParser(xlex);
xparse.program();


CommonAST xtree = (CommonAST)xparse.getAST();

if(DEBUG)
System.out.println(xtree.toStringList());

XawkWalker xwalker = new XawkWalker();

XawkVariable xtype = xwalker.expr(xtree);

} catch (FileNotFoundException e) {
System.err.println("[XAWK IO Error] " + e.getLocalizedMessage());
if(DEBUG)
e.printStackTrace();
} catch (RecognitionException e) {
System.err.println("[XAWK Recognition Error] " + e);
if(DEBUG)
e.printStackTrace();
} catch (TokenStreamException e) {
System.err.println("[XAWK Token Error] " + e);
if(DEBUG)
e.printStackTrace();
}catch(Exception e){
System.err.println("[XAWK Error] " + e);
if(DEBUG)
e.printStackTrace();
}finally{
System.exit(1);
}
}
/*
 *
 */
private static void printInfo(){
```

```
System.err.println("XAWK Version " + version +  " \u00A9 2004-2005");
}
/*
 *
 */
private static void printUsage(){
System.err.println("Usage: xawk <program_file>");
}
/*
 *
 */
    public static void main (String[] args){
     if(args.length == 0){
     printUsage();
     System.exit(1);
     }

     printInfo();
     executeFile(args[0]);
     System.exit(0);
    }
}
```

6.2.2 XawkInterpreter.java

```
/*
 * File: XawkInterpreter.java
 * XAWK (c) 2004 - 2005
 * Authors: Gabriela Cretu  [gcretu@cs.columbia.edu]
 *  Shi Tak Man  [sm2173@columbia.edu]
 *  Prashant Puri  [pp2119@columbia.edu]
 *  John Cieslewicz  [johnc@cs.columbia.edu]
 */

package edu.columbia.xawk;
/**
 * @author Gabriela, Shi Tak, Prashant, and John
 * @version 2.0
 */
import java.io.File;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Stack;

public final class XawkInterpreter {

private LinkedList symbolTable;

private LinkedList xmlParsers;
private XawkXML currentParser;
private Stack pathStack;

private XawkVariable CA;
private XawkVariable CE;
private XawkVariable CV;
private XawkVariable PATH;

/** Default Constructor
 *
 */
public XawkInterpreter() {
symbolTable = new LinkedList();
xmlParsers = new LinkedList();
pathStack = new Stack();
// Built in var CA, set associative array of current attributes
CA = new XawkVariable("CA");
CA.setData(new XawkArray());
symbolTable.add(CA);

// Built in var CE, string representing the name of the current element
CE = new XawkVariable("CE");
CE.setData(new XawkString(""));
symbolTable.add(CE);

//Built in var CV, string representing the value of the current element
CV = new XawkVariable("CV");
CV.setData(new XawkString(""));
symbolTable.add(CV);

PATH = new XawkVariable("PATH");
PATH.setData(new XawkString(""));
symbolTable.add(PATH);
}

/**
 * @param a
 * @return
 */
```

```
public XawkVariable evaluate(XawkVariable a){
XawkVariable returnValue = new XawkVariable("");
if (logicalValue(a)) returnValue.setData(new XawkDouble(1));
else returnValue.setData(new XawkDouble(0));
return returnValue;
}
private boolean inSymbolTable(XawkVariable xv){
Iterator it = symbolTable.iterator();
while(it.hasNext()){
XawkVariable x = (XawkVariable)it.next();
if(xv.getVarName().equals(x.getVarName())){
return true;
}
}
return false;
}

/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable assign(XawkVariable a, XawkVariable b) {
if(!b.isDummy() && !b.isArrayElement()){
if(!inSymbolTable(b)){
b.setData(new XawkString(""));
this.symbolTable.add(b);
}
}

if(a.isDummy())
System.err.print("Assigning to dummy variable!");

if(!inSymbolTable(a) && !a.isArrayElement()){
this.symbolTable.add(a);
}
//printSymbolTable();
a.setData((XawkDataType)((XawkDataType)b.getData()).clone());
return a;
    }

private void printSymbolTable(){
Iterator it = symbolTable.iterator();
System.out.println("Symbol Table:");
while(it.hasNext()){
XawkVariable xv = (XawkVariable)it.next();
XawkDataType xdt = xv.getData();
if(xdt instanceof XawkString){
System.out.println(xv.getVarName()
+ "[STRING] = " + ((XawkString)xdt).getValue());
}else if(xdt instanceof XawkDouble){
System.out.println(xv.getVarName()
+ "[DOUBLE] = " + ((XawkDouble)xdt).getValue());
}else if(xdt instanceof XawkArray){
System.out.println(xv.getVarName() + "[ARRAY]");
}
}
System.out.println("");
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable assignplus(XawkVariable a, XawkVariable b) {
XawkVariable aux = null;
aux = plus(a,b);
return assign(a, aux);
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable assignmult(XawkVariable a, XawkVariable b) {
XawkVariable aux = null;
aux = star(a,b);
return assign(a, aux);
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable assigndiv(XawkVariable a, XawkVariable b) {
XawkVariable aux = null;
aux = div(a,b);
return assign(a, aux);
```

```
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable assignminus(XawkVariable a, XawkVariable b) {
XawkVariable aux = null;
aux = minus(a,b);
return assign(a, aux);
     }
/**
 *
 * @param xv
 * @return
 */
private double coerceToDouble(XawkVariable xv){
XawkDataType xdt = xv.getData();
double returnVal = 0;
if(xdt instanceof XawkDouble){
returnVal = ((XawkDouble)xdt).getValue();
}else if(xdt instanceof XawkString){
XawkDouble xd = XawkDataType.parse((XawkString)xdt);
returnVal = xd.getValue();
}else if(xdt instanceof XawkArray){
throw new XawkException("Arrays must be used with subscripts in expressions.");
}else{
System.err.println("Unknown data type. Cannot coerce to double.");
System.exit(1);
}
return returnVal;
}
/**
 *
 * @param a
 */
private void symbolTableCheck(XawkVariable a){
if(!a.isDummy()){
if(!inSymbolTable(a) && !a.isArrayElement()){
a.setData(new XawkString(""));
this.symbolTable.add(a);
}
}
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable plus(XawkVariable a, XawkVariable b){

XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);
symbolTableCheck(b);

double aa = coerceToDouble(a);
double bb = coerceToDouble(b);

returnValue.setData(new XawkDouble(aa+bb));
return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable concat(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);
symbolTableCheck(b);

String aa = coerceToString(a);
String bb = coerceToString(b);

returnValue.setData(new XawkString(aa+bb));
return returnValue;
}

private String coerceToString(XawkVariable xv){
String returnValue ="";
XawkDataType xdt = xv.getData();
if(xdt instanceof XawkString){
returnValue =  ((XawkString)xdt).getValue();
}else if(xdt instanceof XawkDouble){
returnValue = XawkDataType.toXawkString(xdt).getValue();
}else if(xdt instanceof XawkArray){
throw new XawkException("Arrays must be used with subscripts in expressions.");
}else{
System.err.println("Uknown Data Type - " + xdt.getClass().getName() +
". Cannot coerce to string");
System.exit(1);
}

return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable minus(XawkVariable a, XawkVariable b){

XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);
symbolTableCheck(b);

double aa = coerceToDouble(a);
double bb = coerceToDouble(b);

returnValue.setData(new XawkDouble(aa-bb));
return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable div(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);
symbolTableCheck(b);

double aa = coerceToDouble(a);
double bb = coerceToDouble(b);
if(bb == 0.0)
throw new XawkException("Division by zero error.");
returnValue.setData(new XawkDouble(aa/bb));
return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable star(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);
symbolTableCheck(b);

double aa = coerceToDouble(a);
double bb = coerceToDouble(b);

returnValue.setData(new XawkDouble(aa*bb));
return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable modulus(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);
symbolTableCheck(b);

double aa = coerceToDouble(a);
double bb = coerceToDouble(b);
int ai = (int)aa;
int bi = (int)bb;

if(aa != ai || bb != bi)
throw new XawkException("Modulus operator may only be applied to integeral numeric values.");

returnValue.setData(new XawkDouble(ai%bi));
return returnValue;
}
    /**
    *
    * @param a
    * @return
    */
```

```
public XawkVariable unaryPlus(XawkVariable a){
           XawkVariable returnValue = new XawkVariable("");

           symbolTableCheck(a);

           double aa = coerceToDouble(a);

           returnValue.setData(new XawkDouble(aa));
           return returnValue;
   }
   /**
    *
    * @param a
    * @return
    */
   public XawkVariable unaryMinus(XawkVariable a){
           XawkVariable returnValue = new XawkVariable("");

           symbolTableCheck(a);

           double aa = coerceToDouble(a);

           returnValue.setData(new XawkDouble(aa * -1.0));
           return returnValue;
   }
/**
 *
 * @param a
 * @return
 */
public XawkVariable rincrement(XawkVariable a){
//check if a is in the symbol table
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);

double aa = coerceToDouble(a);
a.setData(new XawkDouble(aa+1));

returnValue.setData(new XawkDouble(aa));
return returnValue;
}
/**
 *
 * @param a
 * @return
 */
public XawkVariable lincrement(XawkVariable a){
//check if a is in the symbol table
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);

double aa = coerceToDouble(a);
a.setData(new XawkDouble(aa+1));

returnValue.setData(new XawkDouble(aa+1));
return returnValue;
}
/**
 *
 * @param a
 * @return
 */
public XawkVariable inc_stat(XawkVariable a){
//check if a is in the symbol table
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);

double aa = coerceToDouble(a);
a.setData(new XawkDouble(aa+1));

returnValue.setData(new XawkDouble(aa+1));
return returnValue;
}
/**
 *
 * @param a
 * @return
 */
public XawkVariable ldecrement(XawkVariable a){
//check if a is in the symbol table
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);

double aa = coerceToDouble(a);
a.setData(new XawkDouble(aa-1));

returnValue.setData(new XawkDouble(aa-1));
return returnValue;
```

```
}
/**
 *
 * @param a
 * @return
 */
public XawkVariable rdecrement(XawkVariable a){
//check if a is in the symbol table
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);

double aa = coerceToDouble(a);
a.setData(new XawkDouble(aa-1));

returnValue.setData(new XawkDouble(aa));
return returnValue;
}
/**
 *
 * @param a
 * @return
 */
public XawkVariable dec_stat(XawkVariable a){
//check if a is in the symbol table
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);

double aa = coerceToDouble(a);
a.setData(new XawkDouble(aa-1));

returnValue.setData(new XawkDouble(aa-1));
return returnValue;
}

/**
 * Comparison is numeric, only if both operands are numeric. Otherwise, coerce to
 * strings, and compare lexically. Awk p. 45
 * @param a
 * @param b
 * @return
 */
public XawkVariable equals(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");
symbolTableCheck(a);
symbolTableCheck(b);

if(a.getData() instanceof XawkDouble && b.getData() instanceof XawkDouble){
XawkDouble aa = (XawkDouble)a.getData();
XawkDouble bb = (XawkDouble)b.getData();
returnValue.setData(new XawkDouble( aa.getValue() == bb.getValue() ? 1 : 0));
}else{
String aa = coerceToString(a);
String bb = coerceToString(b);
returnValue.setData(new XawkDouble( aa == bb ? 1 : 0));
}

return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable ge(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");
symbolTableCheck(a);
symbolTableCheck(b);

if(a.getData() instanceof XawkDouble && b.getData() instanceof XawkDouble){
XawkDouble aa = (XawkDouble)a.getData();
XawkDouble bb = (XawkDouble)b.getData();
returnValue.setData(new XawkDouble( aa.getValue() >= bb.getValue() ? 1 : 0));
}else{
String aa = coerceToString(a);
String bb = coerceToString(b);
returnValue.setData(new XawkDouble( aa.compareTo(bb) >= 0 ? 1 : 0));
}

return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable ls(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");
symbolTableCheck(a);
symbolTableCheck(b);

if(a.getData() instanceof XawkDouble && b.getData() instanceof XawkDouble){
XawkDouble aa = (XawkDouble)a.getData();
XawkDouble bb = (XawkDouble)b.getData();
returnValue.setData(new XawkDouble( aa.getValue() < bb.getValue() ? 1 : 0));
}else{
String aa = coerceToString(a);
String bb = coerceToString(b);
returnValue.setData(new XawkDouble( aa.compareTo(bb) < 0 ? 1 : 0));
}

return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable le(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");
symbolTableCheck(a);
symbolTableCheck(b);

if(a.getData() instanceof XawkDouble && b.getData() instanceof XawkDouble){
XawkDouble aa = (XawkDouble)a.getData();
XawkDouble bb = (XawkDouble)b.getData();
returnValue.setData(new XawkDouble( aa.getValue() <= bb.getValue() ? 1 : 0));
}else{
String aa = coerceToString(a);
String bb = coerceToString(b);
returnValue.setData(new XawkDouble( aa.compareTo(bb) <= 0 ? 1 : 0));
}

return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable gt(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");
symbolTableCheck(a);
symbolTableCheck(b);

if(a.getData() instanceof XawkDouble && b.getData() instanceof XawkDouble){
XawkDouble aa = (XawkDouble)a.getData();
XawkDouble bb = (XawkDouble)b.getData();
returnValue.setData(new XawkDouble( aa.getValue() > bb.getValue() ? 1 : 0));
}else{
String aa = coerceToString(a);
String bb = coerceToString(b);
returnValue.setData(new XawkDouble( aa.compareTo(bb) > 0 ? 1 : 0));
}

return returnValue;
}
/**
 *
 */
public XawkVariable note(XawkVariable a, XawkVariable b){
XawkVariable returnValue = new XawkVariable("");
symbolTableCheck(a);
symbolTableCheck(b);

if(a.getData() instanceof XawkDouble && b.getData() instanceof XawkDouble){
XawkDouble aa = (XawkDouble)a.getData();
XawkDouble bb = (XawkDouble)b.getData();
returnValue.setData(new XawkDouble( aa.getValue() != bb.getValue() ? 1 : 0));
}else{
String aa = coerceToString(a);
String bb = coerceToString(b);
returnValue.setData(new XawkDouble( aa.compareTo(bb) != 0 ? 1 : 0));
}

return returnValue;
}
//TODO perhaps move to other class? of xv?
private boolean logicalValue(XawkVariable xv){
if(xv.getData() instanceof XawkString){
int len = ((XawkString)xv.getData()).getValue().length();
return len > 0;
}else if(xv.getData() instanceof XawkDouble){
double val = ((XawkDouble)xv.getData()).getValue();
return val != 0.0;
}else if(xv.getData() instanceof XawkArray){
throw new XawkException("Arrays must be used with subscripts in expressions.");
}else{
System.err.println("Uknown Data Type. Cannot coerce to string");
System.exit(1);
}
//unreachable code!
System.err.println("unreachable!");
System.exit(1);
return false;
}
/**
 * We do no promotion here.
 * If a is a number, then 0 => 1, non zero => 0
 * If a is a string, then "" => 1, non empty => 0
 * Awk p.46 (null means empty string to us).
 * @param a
 * @return
 */
public XawkVariable logicalnot(XawkVariable a) {
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);
boolean lvalue = logicalValue(a);

returnValue.setData(new XawkDouble(lvalue ? 0.0 :1.0));

return returnValue;
}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable and(XawkVariable a, XawkVariable b) {
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);
symbolTableCheck(b);


boolean lvalue = logicalValue(a) && logicalValue(b);

returnValue.setData(new XawkDouble(lvalue ? 1.0 :0.0));

return returnValue;

}
/**
 *
 * @param a
 * @param b
 * @return
 */
public XawkVariable or(XawkVariable a, XawkVariable b) {
XawkVariable returnValue = new XawkVariable("");

symbolTableCheck(a);
symbolTableCheck(b);


boolean lvalue = logicalValue(a) || logicalValue(b);

returnValue.setData(new XawkDouble(lvalue ? 1.0 :0.0));

return returnValue;
}
/**
 *
 * @param name
 * @param index
 * @return
 */
public XawkVariable getArrayVariable(XawkVariable array, XawkVariable index){
//System.err.println("Getting Array Variable");
XawkVariable returnValue;
if(array.isDummy()){
System.err.println("[XAWK Internal Error] array variable is a dummy...");
System.exit(1);
}

if(!(array.getData() instanceof XawkArray)){
if(inSymbolTable(array)){
throw new XawkException("Variable cannot be both an array and non-array within a program!");
}
//System.err.println("creating new xawk array data type");
array.setData(new XawkArray());
}

if(!inSymbolTable(array)){
//System.err.println("Adding array: " + array.getVarName());
symbolTable.add(array);
}

XawkArray xa = (XawkArray)array.getData();
```

```
String key = coerceToString(index);
//TODO clean up so we either use key or index, not both...
if(xa.isPresent(key)){
//System.err.println("Found variable, " + key +", in array.");
returnValue = xa.getVariable(key);
if(returnValue.isArrayElement() != true){
System.err.println("Array element not marked as such!");
System.exit(1);
}
}else{
//System.err.println("Adding variable, " + key +", to array.");
XawkVariable xv = new XawkVariable(key);
xv.setData(new XawkString(""));
xv.setArrayElementFlag(true);
xa.addVariable(xv);
returnValue = xv;
}
return returnValue;
}

public Iterator getArrayIterator(XawkVariable arrayName){

return null;
}
/**
 *
 * @param s
 * @return
 */
public XawkVariable getNumber(String s) {
try{
double aux = Double.parseDouble(s);
}catch (NumberFormatException e2){
System.out.println(e2);
}
//System.out.println(Double.parseDouble(s));
XawkDouble xawkDouble= new XawkDouble(Double.parseDouble(s));

XawkVariable returnValue = new XawkVariable("");
returnValue.setData(xawkDouble);

return returnValue;
}
/**
 *
 * @param s
 * @return
 */
public XawkVariable getVariable(String s) {
Iterator it = symbolTable.iterator();
XawkVariable xv = null;
while(it.hasNext()){
xv = (XawkVariable)it.next();
if(s.equals(xv.getVarName())){
return xv;
}
}

xv = new XawkVariable(s);
xv.setData(new XawkString("")); //just to be safe...
return xv;
}
/**
 *
 * @param s
 * @return
 */
public XawkVariable getString(String s) {
XawkString xs = new XawkString(s);
XawkVariable returnValue = new XawkVariable("");
returnValue.setData(xs);
return returnValue;
}
/**
 *
 *
 */
public void println(){
System.out.println();
}
/**
 *
 * @return
 */
public XawkVariable getCA(){
return this.CA;
}
/**
 *
 * @param b
 * @return
 */
```

```
public boolean getForCondition(XawkVariable b){
double val = 0.0;
XawkDataType bb = b.getData();
XawkDouble xi = null;
if (bb instanceof XawkDouble){
val = ((XawkDouble)bb).getValue();
}
if (val==1.0) return true;
return false;

}
/**
 *
 * @param out
 */
public void printOutput(XawkVariable out){
symbolTableCheck(out);

XawkDataType xdt = out.getData();

if(xdt instanceof XawkString){
System.out.print(((XawkString)xdt).getValue());
}else if(xdt instanceof XawkDouble){
//only print decimal if needed
double d = ((XawkDouble)xdt).getValue();
int i = (int)d;
if(i == d)
System.out.print(i);
else
System.out.print(d);
}else if(xdt instanceof XawkArray){
throw new XawkException("Arrays must be used with subscripts in expressions.");
}else{
System.err.println("Uknown Data Type. Cannot coerce to string");
System.exit(1);
}
}
/**
 *
 * @return
 */
public boolean advanceXML(){
if(this.currentParser != null)
return this.currentParser.advanceXML();
return true; //no current document, but let's look at the patters to get the doc.
}
/**
 *
 * @param p
 * @return
 */
private String getFileNameFromPattern(Pattern p){
Element e1 = (Element)p.getFirst();
Element e2 = (Element)p.get(1);
if(e1.getType() != Element.CHILD){
throw new XawkException("Outer path must start with single slash.");
}

if(e2.getType() == Element.DOC_ELEMENT){
return e2.getName();
}else{
throw new XawkException("Outer pattern block must start with a doc() element.");
}
}
/**
 *
 *
 */
public void pushPath(){
if(this.currentParser != null){
pathStack.push(currentParser.getCurrentPath());
this.currentParser.pinPath();
}
//TODO tell currentParser to pin the current path.
}
/**
 *
 *
 */
public void popPath(){
if(pathStack.size() > 0){
pathStack.pop();
this.currentParser.unpinPath();
}
if(pathStack.size() ==0){
this.currentParser = null;
}
//TODO tell currentParser to unpin the most recent path.
}
/**
 *
 * @param p
 * @return
 */
public Match match(Pattern p){
//System.out.println("Start of Match, pattern is: " + p.toString());
if(this.currentParser == null){
String fileName = "";
fileName = getFileNameFromPattern(p);
// find or make the parser
Iterator it = this.xmlParsers.iterator();
XawkXML x = null;
while(it.hasNext()){
x = (XawkXML)it.next();
if(x.getFileName() == fileName){
this.currentParser = x;
break;
}
}

if(x == null){
x = new XawkXML(new File(fileName));
this.xmlParsers.add(x);
this.currentParser = x;
}
p.removeFirst();
p.removeFirst(); //now that we've used doc, dump it.
x.advanceXML();
}else{
if(pathStack.size() == 0){
String fileName = getFileNameFromPattern(p);
if(!fileName.equals(this.currentParser.getFileName()))
throw new XawkException("Pattern block may only process one XML file.");
p.removeFirst();
p.removeFirst(); //now that we've used doc, dump it.s

}else{
//System.out.println(p.toString());
Element e = (Element)p.get(1);
if(e.getType() == Element.DOC_ELEMENT)
throw new XawkException("Doc element may not appear in nested pattern.");

//need to prefix the current path stored in the pathStack to the pattern.
LinkedList path = (LinkedList)pathStack.peek();
LinkedList patternPrefix= new LinkedList();
Iterator it = path.iterator();
while(it.hasNext()){
Element elem = new Element(Element.CHILD);
elem.setName("/");
patternPrefix.add(elem);
patternPrefix.add((Element)it.next());
}

it = p.iterator();
while(it.hasNext()){
patternPrefix.addLast((Element)it.next());
}
// patternPrefix.add(p);

p = new Pattern(patternPrefix);

// System.out.println("New Pattern:" + p.toString());
}
}
// System.out.println("End of Match, pattern is: " + p.toString());
// Pattern temp = new Pattern(currentParser.getCurrentPath());
// System.out.println("Current Path: "+temp.toString());
// System.out.println();
return this.currentParser.match(p);
}

public void updateBuiltInVariables(Match match){
CV.setData(new XawkString(match.getValue()));
CE.setData(new XawkString(match.getTagName()));
Hashtable ht = match.getAttributes();
XawkArray array = new XawkArray();
Iterator it = ht.keySet().iterator();
while(it.hasNext()){
String key = (String)it.next();
if(!key.equals("")){
String value = (String)ht.get(key);
XawkVariable xv = new XawkVariable(key);
xv.setData(new XawkString(value));
xv.setArrayElementFlag(true);
array.addVariable(xv);
}
}
CA.setData(array);
PATH.setData(new XawkString(match.getPath()));
}
}
```

6.2.3 XawkVariable.java

```
/*
 * File: XawkVariable.java
 * XAWK (c) 2004 - 2005
 * Created on Nov 21, 2004
 * Revised Dec 11 - 12
 *
 * Author:  Gabriela Cretu  [gcretu@cs.columbia.edu]
 *  John Cieslewicz  [johnc@cs.columbia.edu]
 */
package edu.columbia.xawk;

/**
 * <p>
 * The XawkVariable class is essentially a container for any data used by an XAWK
 * program. Variables have names and associated datatypes. Changing a variable's
 * type requires simply changing its data.
 * <p>
 * XawkVariables can be <i>real</i> or <i>dummy</i>. A <i>real</i> variable has a
 * variable name and is managed either by the symbol table or by an array contained
 * in the symbol table. A <i>dummy</i> variable is used simply as a container for a
 * value, for instance the result of an expression. <i>Dummy</i> variables have null
 * or empty variable names.
 *
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @author Gabriela Cretu  [gcretu@cs.columbia.edu]
 */
public final class XawkVariable{
//varName is immutable after variable creation.
private String varName;
private XawkDataType data;
private boolean arrayElement;

/**
 * Disallow the default constructor.
 *
 */
private XawkVariable(){}
/**
 * Constructor creates a new variable with the specified name.
 * @param var String containing the name of this variable.
 */
public XawkVariable(String var){
//TODO make sure var is valid
varName = var;
arrayElement = false;
}
/**
 * Getter for the variable name.
 * @return
 */
public String getVarName(){
return varName;
}
/**
 * Setter for the data associated with this variable.
 * @param data
 */
public void setData(XawkDataType data){
//TODO make sure data is valid
this.data = data;
}
/**
 * Getter for the data associated with this variable.
 * @return
 */
public XawkDataType getData(){
return this.data;
}
/**
 * Boolean function for determining if this variable is a <i>dummy</i> variable
 * or a <i>real</i> variable.
 * @return
 */
public boolean isDummy(){
return varName == null | varName.equals("");
}
/**
 * Function for determining name equivalence betwen variables.
 * @deprecated
 */
public boolean equals(Object o){
//TODO - Doesn't seem to work in "contains" functions.
XawkVariable xv = (XawkVariable)o;
return (xv.getVarName() == this.varName);
}
/**
 * Boolean function for determining if this variable is an element in an array.
 * @return True if variable is part of an array, false otherwise.
 */
    public boolean isArrayElement(){
```

```
return arrayElement;
    }
    /**
     * Sets the flag for whether or not this variable is an element in an array.
     * @param flag True if variable is part of an array, false otherwise.
     */
    public void setArrayElementFlag(boolean flag){
this.arrayElement = flag;
    }
}
```

6.2.4 XawkDataType.java

```
/*
 * File: XawkDataType.java
 * XAWK (c) 2004 - 2005
 *
 * Authors: Prashant Puri [pp2119@columbia.edu]
 *  Shi Tak Man [sm2173@columbia.edu]
 *  John Cieslewicz [johnc@cs.columbia.edu]
 */

package edu.columbia.xawk;

/**
 * The base class for all XawkDataTypes.
 * @author  Prashant and Shi Tak
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 */
public abstract class XawkDataType implements Cloneable {
    public XawkDataType() {
    }

    public XawkDataType(XawkDataType xdt){
    }
    /**
     * Helper function for determining if a variable is a number.
     * @deprecated
     * @return
     */
public boolean isNumber(){
return (this instanceof XawkDouble);
}

/**
 * added by shi tak
 * @param xstr
 * @return
 */
public static XawkDouble parse(XawkString xstr) {
XawkDouble b2 = null;
String str = xstr.getValue();
double tempval = 0.0;
int i = str.length()-1;

if (str.length() == 0) { // Special Case - Empty String
b2 = new XawkDouble(tempval);
return b2;
}

while ( i >= 0 )
{
try {
tempval = Double.parseDouble(str);
b2 = new XawkDouble(tempval);
return b2;
}
catch (Exception e) {
str = str.substring(0,i);
i--;
}
}
b2 = new XawkDouble(tempval);
return b2;
}

    /**
     * Added by johnc (12/10/2004)
     * @param xdt
     * @return
     */
    public static XawkString toXawkString(XawkDataType xdt){
     if(xdt instanceof XawkString){
return (XawkString)xdt; //TODO it already is, should I make a copy?
}else if(xdt instanceof XawkDouble){
//only include the decimal if needed.
double x = ((XawkDouble)xdt).getValue();
int i = (int)x;
if(x == i)
return new XawkString(Integer.toString(i));
else
return new XawkString(Double.toString(x));
}else{
//should not be reached
System.err.println("[XaxkDataType.java] INTERNAL ERROR: unknown XawkDataType!");
System.exit(1);
}
//unreachable.
return null;
    }
    /*
     *  (non-Javadoc)
     * @see java.lang.Object#clone()
     */
    public abstract Object clone();
}
```

6.2.5 XawkDouble.java

```
/*
 * File: XawkDouble.java
 * XAWK (c) 2004 - 2005
 * Created on Nov 21, 2004
 *
 * Author: Gabriela Cretu [gcretu@cs.columbia.edu]
 *  John Cieslewicz [johnc@cs.columbia.edu]
 */
package edu.columbia.xawk;

/**
 * The <b>XawkDouble</b> class is used to represent all numeric
 * values within XAWK. An <b>XawkDouble</b> is an <b> XawkDataType</b>.
 * @author Gabriela Cretu [gcretu@cs.columbia.edu]
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 */
public final class XawkDouble extends XawkDataType{
private double doubleNumber;
/**
 * Constructor to create an <b>XawkDouble</b> from the specified double.
 * @param val The double value that should be associated with this
 * <b>XawkDouble</b> object.
 */
public XawkDouble(double val){
super();
doubleNumber = val;
}
/**
 * Constructor to create a new <b>XawkDouble</b> from an existing
 * <b>XawkDouble</b>. The new object has the same associate double value.
 * @param val The <b>XawkDouble</b> from which to create a new object.
 */
public XawkDouble(XawkDouble val){
super(val);
doubleNumber = val.getValue();
}
/**
 * Getter for the double value represented by this object.
 * @return The double value.
 */
public double getValue(){
return doubleNumber;
}
/**
 * Setter for the double value represented by this object.
 * @param val The new double value for this object.
 */
public void setValue(double val){
doubleNumber = val;
}
/*
 *  (non-Javadoc)
 * @see java.lang.Object#clone()
 */
public Object clone(){
return new XawkDouble(this);
}

}
```

6.2.6 XawkException.java

```
/*
```

```
 * File: XawkException.java
 * XAWK (c) 2004 - 2005
 * Author: Prashant Puri [pp2119@columbia.edu]
 *
 * Created on November 12, 2004, 9:50 AM
 */
package edu.columbia.xawk;


/**
 * This class creates customized XAWK exceptions.
 * @author Prashant Puri [pp2119@columbia.edu]
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 */
class XawkException extends RuntimeException {
//12/8/04
//johnc: fixed this c-tor, it now correctly calls super()
// and no longer prints to stderr from withing the c-tor.
XawkException( String msg ) {
super(msg);
}
}
```

6.2.7 XawkString.java

```
/*
 * File: XawkString.java
 * XAWK (c) 2004 - 2005
 * Created on Nov 21, 2004
 *
 * Author: Gabriela Cretu [gcretu@cs.columbia.edu]
 *
 */
package edu.columbia.xawk;

/**
 * The <b>XawkString</b> class is used to represent all string
 * values within XAWK. An <b>XawkStringe</b> is an <b> XawkDataType</b>.
 * @author Gabriela Cretu [gcretu@cs.columbia.edu]
 *
 */
public final class XawkString extends XawkDataType{
private String string;
/**
 * Constructor to create an <b>XawkString</b> representing the provided
 * String.
 * @param string The String associated with this <b>XawkString</b>.
 */
public XawkString(String string){
super();
this.string = string;
}
/**
 * Constructor to create an <b>XawkString</b> from an existing
 * <b>XawkString<b>. The new <b>XawkString</b> represents the same string
 * value as the object from which it is constructed.
 * @param string
 */
public XawkString(XawkString string){
super(string);
this.string = string.getValue();
}
/**
 * Getter for the string value of this <b>XawkString</b>.
 * @return The String object contained in this XawkString.
 */
public String getValue(){
return string;
}
/**
 * Setter for the string value of this <b>XawkString</b>.
 * @param string The String object to be represented by this
 * <b>XawkString</b>.
 */
public void setValue(String string){
this.string = string;
}
/*
 *  (non-Javadoc)
 * @see java.lang.Object#clone()
 */
public Object clone(){
return new XawkString(this);
}
}
```

6.2.8 XawkArray.java

```
/*
 * File: XawkArray.java
 * XAWK (c) 2004 - 2005
 * Created on Dec 5, 2004
 *
 * Author: John Cieslewicz [johnc@cs.columbia.edu]
 */
package edu.columbia.xawk;

import java.util.Hashtable;
import java.util.Iterator;
/**
 * Xawk arrays are associative arrays.
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @version December 12, 2004
 */
public final class XawkArray extends XawkDataType implements Iterator{
private Hashtable hashtable;
private Iterator myIterator;
/**
 * Default constructor. Creates an empty associative array.
 */
public XawkArray(){
super();
hashtable = new Hashtable();
}
/**
 * Contructor from another XawkArray.
 * We should never be doing this, but the other XawkDataTypes allow it, so
 * it is included here for symmetry.
 * @param xa The XawkArray to clone.
 */
public XawkArray(XawkArray xa){
super(xa);
hashtable = new Hashtable(xa.hashtable);
System.err.println("[XAWK Internal Error] Creating an array from an array.");
System.exit(1);
}
/**
 * Get an iterator for this array. Used to implement the (for x in array)
 * statement. The iterator returns the keys which can be used to index the array.
 * @return
 */
public Iterator getArrayIterator(){
return this.hashtable.keySet().iterator();
}
/**
 * Adds the variable to the array.
 * Causes an internal error if the same index is added twice. Always check
 * before adding!
 * @param xdt
 */
public void addVariable(XawkVariable xv){
if(xv.getVarName() == null | xv.getVarName().equals("")){
System.err.println("[XAWK Internal Error] Adding a variable with no name to an array!");
System.exit(1);
}
if(isPresent(xv.getVarName())){
System.err.println("[XAWK Internal Error] Re-adding a variable to an array");
System.exit(1);
}
xv.setArrayElementFlag(true);
hashtable.put(xv.getVarName(), xv);
}
/**
 * Checks to see if a given index is already present in the array.
 * @param varName Name of the index variable
 * @return Boolean that is true if the index is in the array, false otherwise.
 */
public boolean isPresent(String varName){
Iterator it = hashtable.keySet().iterator();
while(it.hasNext()){
String key = (String)it.next();
if(key.equals(varName))
return true;
}
return false;
}
/**
 * Finds the variable corresponding to a provided index and returns it.
 * @param index A String representing the array index to be returned.
 * @return The XawkVariable found at the specified index.
 */
public XawkVariable getVariable(String index){
return (XawkVariable)hashtable.get(index);
}
/**
 * Provided for completeness. Needed for other XawkVariables, causes error here.
 */
public Object clone(){
```

```
System.err.println("cloning an array; not allowed. INTERNAL ERROR!");
return null;
}
public Iterator iterator(){
//TODO set up code for making this an iterator...
myIterator = hashtable.keySet().iterator();
return this;
}
/* (non-Javadoc)
 * @see java.util.Iterator#hasNext()
 */
public boolean hasNext() {
if(myIterator != null)
return myIterator.hasNext();
return false;
}
/* (non-Javadoc)
 * @see java.util.Iterator#next()
 */
public Object next() {
if(myIterator != null && myIterator.hasNext()){
String key = (String)myIterator.next();
XawkVariable xv = (XawkVariable)hashtable.get(key);
return xv;
}
System.err.println("[INTERNAL XawkAray Error] Iterator null or called when hasNext() false.");
System.exit(1);
return null;
}
/* (non-Javadoc)
 * @see java.util.Iterator#remove()
 */
public void remove() {
System.err.println("[XAWK INTERNAL ERROR] Cannot remove from XAWK Array!");
System.exit(1);

}
}
```

6.3 XML Parsing and Pattern Matching

6.3.1 XawkXML.java

```
/*
 * File: XawkXML.java
 * XAWK (c) 2004 - 2005
 * Created on Oct 31, 2004
 *
 * Authors: John Cieslewicz [johnc@cs.columbia.edu]
 *  Gabriela Cretu [gcretu@cs.columbia.edu]
 */

package edu.columbia.xawk;

import java.io.File;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Stack;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.SAXException;

/**
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @author Gabriela Cretu [gcretu@cs.columbia.edu]
 * @version October 31, 2004
 */
public class XawkXML implements Runnable{
//private i-vars
private File xmlFile;
private SAXParser saxParser;
private ThreadSafeQueue queue;
private LinkedList path;
private Stack pathSizeStack;

/**
 * Disallow the default constructor
 */
private XawkXML(){};
/**
```

```
 * Constructor for a new XawkXML object to process the file specified by
 * the constructor's parameter.
 * @param xmlFile The File to be processed by this XawkXML object.
 */
public XawkXML(File xmlFile){
this.xmlFile = xmlFile;
queue = new ThreadSafeQueue();
path = new LinkedList();
pathSizeStack = new Stack();
SAXParserFactory factory = SAXParserFactory.newInstance();
try {
saxParser = factory.newSAXParser();
} catch (ParserConfigurationException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (SAXException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

Thread parseThread = new Thread(this, "Parse Thread");
parseThread.start();
}
/**
 *
 * @return
 */
public boolean advanceXML(){
XMLQueueEntry entry = getNext();
if(entry == null)
return false; // we're at the end of the document.
//System.out.println(entry.toString());
//printLinkedList(path);
Element tag;
switch(entry.getType()){
case XMLQueueEntry.START_ELEMENT:
removeCharacterTail();
tag = new Element(Element.XML_ELEMENT);
tag.setName(entry.getQName());
tag.setAttributes(entry.getAttributes());
path.add(tag);
break;
case XMLQueueEntry.CHARACTERS:
removeCharacterTail();
tag  = new Element(Element.TEXT_ELEMENT);
tag.setText(entry.getCharacters());
path.add(tag);
break;
case XMLQueueEntry.END_ELEMENT:
removeCharacterTail();
path.removeLast();
if(pathSizeStack.size() >0){
int size = ((Integer)pathSizeStack.peek()).intValue();
if(path.size() >= size)
return advanceXML(); //move on to the next element
}
return false;
case XMLQueueEntry.EOF:
removeCharacterTail();
return false;
default:
throw new XawkException("Unknown XMLQueueEntry type.");
}
return true;
}
/**
 *
 */
private void removeCharacterTail(){
if(path.size() > 0){
Element e = (Element)path.getLast();
if(e.getType() == Element.TEXT_ELEMENT){
path.removeLast();
}
}
}
/**
 *
 * @param p
 * @return
 */
public Match match(Pattern p){
Match match = null;
Element tag;
if(p.isMatch(this.path)){
tag = (Element)path.getLast();
Hashtable attributes = tag.getAttributes();


match = new Match(attributes, tag.getText(), tag.getName(), this.pathToString() );

}
```

```
return match;
}
/**
 *
 */
private XMLQueueEntry getNext(){
XMLQueueEntry entry;
while(null == (entry = (XMLQueueEntry)queue.dequeue())){
try {
Thread.sleep(10);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
return entry;
}

/* (non-Javadoc)
 * @see java.lang.Runnable#run()
 */
public void run() {
try {
saxParser.parse( xmlFile, new XawkXMLHandler(queue));
} catch (SAXException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
queue.enqueue(new XMLQueueEntry(XMLQueueEntry.EOF));
}
/**
 *
 *
 */
public void printParse(){
XMLQueueEntry entry;
for(;;){
while( null == (entry = (XMLQueueEntry)queue.dequeue())){
try {
Thread.sleep(10);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}

switch(entry.getType()){
case XMLQueueEntry.START_ELEMENT:
System.out.println("<" + entry.getQName() + ">");
break;
case XMLQueueEntry.END_ELEMENT:
System.out.println("</" + entry.getQName() + ">");
break;
case XMLQueueEntry.CHARACTERS:
System.out.print(new String(entry.getCharacters()));
    break;
case(XMLQueueEntry.EOF):
return;
default:
return;
}
}
}
/**
 * returns a new linked list containing the current path.
 * @return
 */
public LinkedList getCurrentPath(){
return new LinkedList(this.path);
}
/**
 *
 * @return
 */
public String getFileName(){
return this.xmlFile.getName();
}
/**
 *
 * @return
 */
public String getAbsolutePath(){
return this.xmlFile.getAbsolutePath();
}
/**
 *
 *
 */
public void pinPath(){
```

```
Integer size = new Integer(path.size());
pathSizeStack.push(size);
}
/**
 *
 *
 */
public void unpinPath(){
if(pathSizeStack.size() > 0)
pathSizeStack.pop();
}
private String pathToString(){
String pathString = "/";
for(int i = 0 ; i < path.size(); i++){
pathString += path.get(i) + "/";
}
return pathString;
}
/*
 *
 */
public static void main(String[] args){
File xmlFile = new File(args[0]);
XawkXML x = new XawkXML(xmlFile);
x.printParse();
}
public static void printLinkedList(LinkedList list){
Iterator it = list.iterator();
while(it.hasNext()){
Element e = (Element)it.next();
System.out.println("Path element name: " +e.getName());
}
}
}
```

6.3.2 XawkXMLHandler.java

```
/*
 * File: XawkXMLHandler.java
 * XAWK (c) 2004 - 2005
 *
 * Authors: John Cieslewicz [johnc@cs.columbia.edu]
 *  Gabriela Cretu [gcretu@cs.columbia.edu]
 */

package edu.columbia.xawk;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
/**
 * The <b>XawkXMLHandler</b> class implements the callback functions
 * required by the SAX parser.
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @author Gabriela Cretu [gcretu@cs.columbia.edu]
 * @version October 31, 2004
 * @see org.xml.sax.helpers.DefaultHandler
 */
public class XawkXMLHandler extends DefaultHandler{
/**
 *
 */
private ThreadSafeQueue queue;
private boolean processingChars;
private String characters;

private static final int QUEUE_SIZE = 8;
/**
 * Don't allow a default constructor to be used.
 */
private XawkXMLHandler(){};
/**
 * Build a new XawkXMLHandler.
 * @param queue The threadsafe queue shared between the handler and the
 * main Xawk xml class.
 */
public XawkXMLHandler(ThreadSafeQueue queue){
super();
this.queue = queue;
this.processingChars = false;
}
/**
 *
 */
    public void startElement(String namespaceURI, String localName,
                             String qName, Attributes attrs)
throws SAXException{
     if(this.processingChars){
addToMyQueue(new XMLQueueEntry(this.characters));
this.characters = null;
this.processingChars = false;
}

//     System.out.println("[XAWK Handler] namespaceURI: " + namespaceURI);
//     System.out.println("[XAWK Handler] localName: " + localName);
//     System.out.println("[XAWK Handler] qName: " + qName);
//     System.out.println("[XAWK Handler] attributes: " + attrs.toString());


     addToMyQueue(new XMLQueueEntry(namespaceURI, localName, qName, attrs));
    }
    /**
     *
     */
    public void endElement(String namespaceURI, String localName, String qName)
throws SAXException
{
     if(this.processingChars){
     addToMyQueue(new XMLQueueEntry(this.characters));
     this.characters = null;
     this.processingChars = false;
     }
//     System.out.println("[XAWK Handler/] namespaceURI: " + namespaceURI);
//     System.out.println("[XAWK Handler/] localName: " + localName);
//     System.out.println("[XAWK Handler/] qName: " + qName);

     addToMyQueue(new XMLQueueEntry(namespaceURI, localName, qName));

    }
    /**
     *
     */
    public void characters(char[] ch, int start, int length)
throws SAXException
{
     //System.out.println("PROCESSING CHARACTERS!");
     this.processingChars = true;
     char[] temp = new char[length];
     for(int i = 0; i < length; i++)
     temp[i] = ch[i+start];
     String tempStr = new String(temp);
     if(this.characters == null)
     this.characters = tempStr;
     else
     this.characters = this.characters + tempStr;
    }
    /**
     * A helper function to handle enqueueing new XMLQueueEntry objects;
     * waits until the queue is below the size threshold before
     * enqueueing a new entry.
     * @param entry The XMLQueueEntry that the calling function wants to enqueue.
     */
    private void addToMyQueue(XMLQueueEntry entry){
  //   System.out.println("Entry to add to queue: ");
  //   System.out.println(entry.toString());
     while(queue.size() >= QUEUE_SIZE){
try {
Thread.sleep(10);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
     }

// if(attributes != null){
//      for(int i = 0; i < attributes.getLength(); i++){
// String key = attributes.getLocalName(i) == ""
// ? attributes.getQName(i)
// : attributes.getLocalName(i);
// String value = attributes.getValue(i);
// System.out.println("Putting: " + key +", " + value);
// }
// }
this.queue.enqueue(entry);

}
}
```

6.3.3 XMLQueueEntry.java

```
/*
 * File: XMLQueueEntry.java
 * XAWK (c) 2004 - 2005
 * Created on Oct 31, 2004
 *
 * Authors: John Cieslewicz  [johnc@cs.columbia.edu]
 *     Gabriela Cretu [gcretu@cs.columbia.edu]
 */
```

```
package edu.columbia.xawk;

import java.util.Hashtable;

import org.xml.sax.Attributes;

/**
 * This class contains information passed to the SAX parser callback functions.
 * This data is passed via the queue from the SAX thread to the main Xawk XMM thread.
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @author Gabriela Cretu [gcretu@cs.columbia.edu]
 * @version October 31, 2004
 */
public class XMLQueueEntry {
//constants
public static final int START_ELEMENT = 0;
public static final int END_ELEMENT = 1;
public static final int CHARACTERS = 2;
public static final int EOF = 3;
//i-vars
private String namespaceURI;
    private String localName; // local name
    private String qName; // qname
    private Hashtable attributes;
    private int type;
    private String characters;

/**
 * We don't want anyone making a queue entry with the defaul constructor.
 */
private XMLQueueEntry(){}
/**
 * A constructor for a XMLQueueEntry for start tag information from the
 * SAX parser.
 * @param namespaceURI
 * @param localName
 * @param qName
 * @param attrs
 */
public XMLQueueEntry(String namespaceURI, String localName,
              String qName, Attributes attributes){
this.namespaceURI = namespaceURI;
this.localName = localName;
this.qName = qName;
this.attributes = new Hashtable();
for(int i = 0; i < attributes.getLength(); i++){
String key = attributes.getLocalName(i) == ""
? attributes.getQName(i)
: attributes.getLocalName(i);
String value = attributes.getValue(i);
//System.out.println("Putting: " + key +", " + value);
this.attributes.put(key, value);
}

this.type = START_ELEMENT;
}
/**
 * A constructor for a XMLQueueEntry for end tag information from the SAX parser.
 * @param namespaceURI
 * @param localName
 * @param qName
 */
public XMLQueueEntry(String namespaceURI, String localName, String qName){
this.namespaceURI = namespaceURI;
this.localName = localName;
this.qName = qName;
this.type = END_ELEMENT;
}
/**
 * A constructor for a XMLQueueEntry for character information.
 * @param characters
 * @param start
 * @param length
 */
public XMLQueueEntry(String characters){
this.characters = characters;
this.type = CHARACTERS;
}
/**
 * A constructor for a XMLQueueEntry holding only a type (i.e. for the end
 * of file)
 * @param type
 */
public XMLQueueEntry(int type){
this.type = type;
}
/**
 * Getter for the namespaceURI.
 * @return
 */
public String getNamespaceURI(){
```

```
return this.namespaceURI;
}
/**
 * Setter for the namespaceURI
 * @param namespaceURI
 */
public void setNamespaceURI(String namespaceURI){
this.namespaceURI = namespaceURI;
}
/**
 * Getter for the local name.
 * @return
 */
public String getLocalName(){
return this.localName;
}
/**
 * Setter for the local name.
 * @param localName
 */
public void setLocalName(String localName){
this.localName = localName;
}
/**
 * Getter for the q name.
 * @return
 */
public String getQName(){
return this.qName;
}
/**
 * Setter for the q name.
 * @param qName
 */
public void setQName(String qName){
this.qName = qName;
}
/**
 * Getter for the attributes.
 * @return
 */
public Hashtable getAttributes(){
return this.attributes;
}
/**
 * Setter for the attributes.
 * @param attributes
 */
public void setAttributes(Hashtable attributes){
this.attributes = attributes;
}
/**
 * Getter for the type.
 * @return
 */
public int getType(){
return this.type;
}
/**
 * Setter for the type.
 * @param type
 */
public void setType(int type){
this.type = type;
}
/**
 * Getter for the characters.
 * @return
 */
public String getCharacters(){
return this.characters;
}
/**
 * Setter for the characters.
 * @param characters
 */
public void setCharacters(String characters){
this.characters = characters;
}
public String toString(){
return "Local name: " + this.localName
+ "\n Qname: " +this.qName
+ "\n Characters: " + this.characters;
}
}
```

6.3.4 ThreadSafeQueue.java

```
/*
```

```
 * File: ThreadSafeQueue.java
 * XAWK (c) 2004 - 2005
 * Created on Nov 1, 2004
 *
 * Author: John Cieslewicz [johnc@cs.columbia.edu]
 */
package edu.columbia.xawk;

import java.util.LinkedList;
import java.util.NoSuchElementException;

/**
 * A thread safe queue.
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @version November 1, 2004
 */
public class ThreadSafeQueue {
//private i-var
private LinkedList list;

/**
 * Build a new ThreadSafeQueue. Only one thread can access this object at a time.
 */
public ThreadSafeQueue(){
list = new LinkedList();
}
/**
 * Adds an object to the end of the queue.
 * @param obj The object that will be added to the queue.
 */
synchronized public void enqueue(Object obj){
list.addLast(obj);
}
/**
 * Removes the object at the front of the queue.
 * @return The object at the front of the queue, null if the queue is empty.
 */
synchronized public Object dequeue(){
try{
XMLQueueEntry e = (XMLQueueEntry)list.removeFirst();
return e;
}catch(NoSuchElementException e){
return null;
}
}
/**
 * Returns a pointer to the first element in the queue, if such an element
 * exists, without dequeueing the element.
 * @return The element at the head of the queue or null if the queue is empty.
 */
synchronized public Object peek(){
try{
return list.getFirst();
}catch(NoSuchElementException e){
return null;
}
}
/**
 * Returns the current size of the queue.
 * @return The current number of enqueued items.
 */
synchronized public int size(){
return list.size();
}
}
```

6.3.5 Element.java

```
/*
 * File: Element.java
 * XAWK (c) 2004 - 2005
 * Created on Oct 31, 2004
 *
 * Authors: John Cieslewicz [johnc@cs.columbia.edu]
 *  Gabriela Cretu [gcretu@cs.columbia.edu]
 *
 */
package edu.columbia.xawk;

import java.util.Hashtable;

/**
 * The <b>Element</b> class contains the information about a XML element
 * or special pattern components that specify how a pattern can match.
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @author Gabriela Cretu [gcretu@cs.columbia.edu]
 * @version December 6, 2004
 */
public class Element {
```

```
//public constants defining "type"
public static final int XML_ELEMENT = 0;
public static final int DESCENDENT = 1;
public static final int CHILD = 2;
public static final int MATCH_ALL = 3;
public static final int MATCH_ANY_ATTRIBUTE = 4;
public static final int MATCH_ATTRIBUTE = 5;
public static final int TEXT_ELEMENT = 6;
public static final int NULL_ELEMENT = 7;
public static final int DOC_ELEMENT = 8;

//private i-vars
private String name;
private Hashtable attr;
private String text = "";
private int type;

/**
 * Disallow use of the default constructor.
 */
private Element(){}
/**
 * The Element constructor must supply an element type.
 * @param type
 */
public Element(int type){
this.type = type;
//TODO error check?
}
/**
 * Getter for the name of this element.
 * @return
 */
public String getName(){
return this.name;
}
/**
 * Setter for the name of this element.
 * @param name
 */
public void setName(String name){
this.name = name;
}
/**
 * Getter for the type of this element.
 * @return
 */
public int getType(){
return this.type;
}
/**
 * Setter for the type of this element.
 * @param type
 */
public void setType(int type){
this.type = type;
}
/**
 * Getter for this element's text.
 * @return
 */
public String getText(){
return this.text;
}
/**
 * Setter for this element's text.
 * @param text
 */
public void setText(String text){
this.text = text;
}
/**
 * Getter for this element's attributes.
 * @return
 */
public Hashtable getAttributes(){
return this.attr;
}
/**
 * Setter for this element's attributes.
 * @param attr
 */
public void setAttributes(Hashtable attr){
this.attr = attr;
}
/**
 * Override of the toString method inherited from the Object class.
 * @see java.lang.Object#toString()
 */
public String toString(){
return this.name;
}
```

```
}
```

6.3.6 Match.java

```
/*
 * File: Match.java
 * XAWK (c) 2004 - 2005
 * Created on Oct 31, 2004
 *
 * Author: John Cieslewicz [johnc@cs.columbia.edu]
 */
package edu.columbia.xawk;

import java.util.Hashtable;

/**
 * This class contains information about an element that has matched a pattern.
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @version October 31, 2004
 */
public final class Match {
//private i-vars
private String tagName;
private String value;
private Hashtable attributes;
private String path;

/**
 * A constructor for a match object.
 * @param attributes
 * @param value
 * @param tagName
 */
public Match(Hashtable attributes,
String value, String tagName, String path){
this.attributes = attributes;
this.value = value;
this.tagName = tagName;
this.path = path;
}
/**
 * Getter for the tag name of the xml tag represented by this match.
 * @return
 */
public String getTagName(){
return this.tagName;
}
/**
 * Setter for the tag name of the xml element represented by this match.
 * @param tagName
 */
public void setTagName(String tagName){
this.tagName = tagName;
}
/**
 * Getter for the value of this match.
 * @return
 */
public String getValue(){
return this.value;
}
/**
 * Setter for the value of this match.
 * @param value
 */
public void setValue(String value){
this.value = value;
}
/**
 * Getter for the attributes of this match.
 * @return
 */
public Hashtable getAttributes(){
return this.attributes;
}
/**
 * Setter for the attributes of this match.
 * @param attributes
 */
public void setAttributes(Hashtable attributes){
this.attributes = attributes;
}
/**
 * Getter for a string representing the current path.
 * @return
 */
public String getPath(){
return this.path;
}
/**
 * Setter for the current path.
 * @param path
 */
public void setPath(String path){
this.path = path;
}
}
```

6.3.7 Pattern.java

```
/*
 * File: Pattern.java
 * XAWK (c) 2004 - 2005
 * Created on Oct 31, 2004
 *
 * Authors: John Cieslewicz  [johnc@cs.columbia.edu]
 *  Gabriela Cretu [gcretu@cs.columbia.edu]
 */
package edu.columbia.xawk;

import java.util.Iterator;
import java.util.LinkedList;

/**
 * @author John Cieslewicz [johnc@cs.columbia.edu]
 * @author Gabriela Cretu [gcretu@cs.columbia.edu]
 * @version October 31, 2004
 * <p>
 * A pattern is a list of "elements". They can be xml elements, or special path
 * elements. The pattern class also supports the comparison of patterns.
 */

//TODO what to do when we have / only?

public class Pattern extends LinkedList {
/**
 * Disallow the default constructor.
 */
private Pattern(){};
/**
 * Build a new pattern object from the given linked list.
 * @param list
 */
public Pattern(LinkedList list){
super(list);
}
/**
 * An override of the toString method to print out the contents of the pattern.
 *  @see java.lang.Object#toString()
 */
public String toString(){
Iterator it = this.iterator();
String result = "";
while(it.hasNext()){
result += it.next().toString() + " ";
}
return result;
}
/**
 * A pattern p matches if I can satisfy the pattern specified by p.
 * @param p
 * @return
 */
public boolean isMatch(LinkedList path){
Element[] patternElement;// = new Element[0];
Element[] pathElement;// = new Element[0];
int i = 0, j = 0;

patternElement = (Element[])this.toArray(new Element[0]);
pathElement = (Element[])path.toArray(new Element[0]);


while(i < patternElement.length && j < pathElement.length){

if (patternElement[i].getType() == Element.CHILD){
i++;
if (patternElement[i].getType() == Element.XML_ELEMENT){
if (!patternElement[i].getName().equals(pathElement[j].getName()))
return false;
}
else if (patternElement[i].getType() == Element.MATCH_ALL){
return true; //TODO assert that we are at the end
}
else if (patternElement[i].getType() == Element.MATCH_ANY_ATTRIBUTE){
if (pathElement[j].getAttributes().size() == 0)
return false;
else return true;
//TODO assert at end of pattern
}
else if (patternElement[i].getType() == Element.MATCH_ATTRIBUTE){
if (!pathElement[j].getAttributes().containsKey(patternElement[i].getName()))
return false;
else return true; //TODO assert that we are at the end
}
else if (patternElement[i].getType() == Element.TEXT_ELEMENT){
if (pathElement[j].getType() == Element.TEXT_ELEMENT)
return true;
else return false;//TODO assert that we are at the end
}
}

else if (patternElement[i].getType() == Element.DESCENDENT){
i++;
if (patternElement[i].getType() == Element.XML_ELEMENT){
while (j < pathElement.length && !patternElement[i].getName().equals(pathElement[j].getName()))
j ++;
}
else if (patternElement[i].getType() == Element.MATCH_ALL){
return true; //TODO assert that there IS a decendent.
}
else if (patternElement[i].getType() == Element.MATCH_ANY_ATTRIBUTE){
//TODO - note that we are checking to make sure the LAST element in the path has an attribute.
if (pathElement[pathElement.length-1].getAttributes().size() == 0)
return false;
else
return true;
//TODO assert at end of pattern
}
else if (patternElement[i].getType() == Element.MATCH_ATTRIBUTE){
if (!pathElement[pathElement.length-1].getAttributes().containsKey(patternElement[i].getName()))
return false;
else return true; //TODO assert that we are at the end of the pattern
}
else if (patternElement[i].getType() == Element.TEXT_ELEMENT){
if (pathElement[pathElement.length -1].getType() == Element.TEXT_ELEMENT)
return true;
else return false;//TODO assert that we are at the end of the patter
}

if(j >= pathElement.length)
return false;
}
i++;
if(i < patternElement.length - 1){
//do not increment j if we are next matching attributes of the current spot in the path.
if(!(patternElement[i+1].getType() == Element.MATCH_ANY_ATTRIBUTE
|| patternElement[i+1].getType() == Element.MATCH_ATTRIBUTE))
j++; //TODO assert that j's current value is pathElement.lengh -1 :)
}else if(i == patternElement.length){
j++;
}
}
if(i == patternElement.length && j == pathElement.length)
return true;
else
return false;
}
}
```

Appendix A

Example XML File

This is a sample XML file that could be processed by the XAWK program in Section 1.4.

```
<class>
  <teacher name = "Stephen Edwards" email ="sedwards@cs.columbia.edu"/>
  <student name = "John Cieslewicz" gender="male">
    <email>johnc@cs.columbia.edu</email>
    <assignment type = "problem set" name = "Problem Set #1">
      <grade>95</grade>
    </assignment>
    <assignment type = "project" name="compiler">
      <grade>82.5</grade>
    </assignment>
  </student>
  <student name = "Gabriela Cretu" gender="female">
    <email>gcretu@cs.columbia.edu</email>
    <assignment type = "problem set" name = "Problem Set #1">
      <grade>96.2</grade>
    </assignment>
    <assignment type = "project" name="compiler">
      <grade/>
    </assignment>
  </student>
  <student name = "Prashant Puri" gender="male">
    <email>pp2119@columbia.edu</email>
    <assignment name = "Problem Set #1" type = "problem set">
      <grade>96</grade>
    </assignment>
    <assignment name = "compiler" type="project>
    </assignment>
  </student>
</class>
```

Bibliography

[1] Stephen A. Edwards, Seema Gupta, Miqdad Mohammed, and Peter T. Chen. XAWK Language Reference Manual. Unpublished, Columbia University, 11 2003.

[2] Rusty Harold. Processing XML with java. http://www.cafeconleche.org/books/xmljava/chapters/index.html.

[3] Stefan Tramm and Jürgen Kahrs. XML GAWK. http://homepage.mac.com/stefan.tramm/iWiki/XmlGawkTutorial.html.

[4] Peter von der Ahé and Jakob Justsen. XAWK. http://www.daimi.au.dk/ just/IWS/man.html.