Spidr

Final Report

Alex Dong **aqd2000**

Katherine Haas **kah2190**

Matt Meisinger **mrm2205**

Akshata Ramesh **ar3120**

Contents

1. Introduction

Spidr is a programming language that allows for users to quickly retrieve web pages and scan them for content. Spidr allows users to easily follow all links on a page, retrieve words from child pages, compile all links from a page into a list, get a list of the URLs of all the images, documents, etc. on a page, and scan for dead links.

1.1 Background

The name "Spidr" alludes to the language's focus on retrieving and scanning web pages. Intended use cases for this language range from applications like the Craigslist scraper PadMapper.com to language processing search engines like WolframAlpha.

1.2 Description

The language Spidr is developed using the Ocaml language. It compiles to Java as its intermediate language, but has its own primitives and diverse syntax. The language has a main function, like Java, but unlike Java each newline is the end of a statement. Spidr only requires a few lines of code to retrieve a page and parse out all of the links, words, and image references on the page. The user can then also follow all the links of the page through the few additional lines of code. Concise list manipulation operations are built into the language. These make integrating more sophisticated logic into the application easier. Functions can be called recursively, allowing for the implementation of complex algorithms within Spidr.

1.3 Features

Spidr is a language that can take a large amount of HTML data, parse through it, and output to the user a specific piece of information they are searching for. Our language is focused on being simple, clean and powerful.

Simplicity

One of the most important functionalities of Spidr was to keep it simple and approachable so that the user can quickly implement a program to solve their problem. Being that our language is similar to many other procedural languages, it provides an easy environment to quickly grasp the syntax and see results quickly. The compilation process was built to abstract as many of the details out of the process as possible, so getting to "Hello world!" takes only a minute.

Clean

Spidr requires minimal code in order to traverse through webs of pages. The goal was for Spidr code to contain as few unnecessary adornments as possible, making the intent of the original author more obvious to later reviewers.

Powerful

Though the language was built to be simple and clean, the core functionality is quite powerful. It can parse through moderately ill-formatted HTML, and has powerful element-matching selectors that make it easy to define which elements on a page you are looking for.

2. Language Tutorial

If users are familiar with the basic concepts of procedural programming, then Spidr syntax should be relatively easy to pick up. This section provides users with the basic tools needed to sift through an HTML page for specific data.

2.1 Sample Program

Below you will find a sample program implementing the Spidr language.

```
/*
The following demo craws site specified in startUrl, and returns all
active links the page, and all active links on those pages.
Warning: Two levels deep is a lot of links. It may take a couple minutes to
crawl any given site.
*/


function void main() {
       string site = "http://www.cs.columbia.edu/~sedwards/software.html"
       println("Starting to crawl site: " + site)

url startUrl = :site
url[] children = getChildUrls(startUrl, 2)

println("Completed!")
}
function url[] getChildUrls(url u, int depth) {
println(u)
```

```
        if (depth == 0) {
                return [u]
        }
        else {
                string[] links = u * <<a@href>>
                url[] activeChildren = []
                loop (links l) {
                        if (live(:l)) {
                        activeChildren = activeChildren + getChildUrls(:l, depth-1)
                        }
                }

                return [u] + activeChildren
        }
}
```

2.2 How to Compile and Run the Sample Program

This section will walk the user through running and compiling the sample program that is in the previous section.

- The prerequisites needed in order to effectively compile a Spidr program are the java development environment (javac), and the Ocaml compiler.
- Ubuntu linux is recommended, though other platforms may work as well.
- To compile the spidr compiler and supporting java libraries:

  Run '`make`' at the root of the unzipped spidr folders.

- To execute a source file:

Run './spidr -e < myfile.spidr' to compile and execute the spidr source file.

- To run the test suite, type either of the following when in the root of the project:

  ```
  make test
  ```

  (Tests whether the spidr tests compile into the expected java code.)

  OR type:

  ```
  make testexe
  ```

  (Also tests whether the java compiles and when the java is run the output is correct.)

- By typing:

  ‘`make clean`’

  You remove all unnecessary files from all of the spidr folders.

The three java classes (SUrl.java, SSelector.java, and SAttSelector.java) below are needed in order to compile the Spidr files.

SUrl.java:
```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;

public class SUrl {
	public String url;

	public SUrl(String url) {
		this.url = url;
	}
	public String toString(){
		return ":\"" + this.url + "\"";
	}
}
```

SSelector.java:

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;

public class SSelector {
      public String elementName;
      public String className;
      public String attr;
      public String attrValue;
      public SSelector innerSelector;
      public SAttSelector attSelector;

      public SSelector(String elementName, String className, String attr,
String attrValue) {
            this.elementName = elementName;
            this.className = className;
            this.attr = attr;
            this.attrValue = attrValue;
            this.innerSelector = null;
            this.attSelector = null;
      }

      public String toString(){
            return "<<" + this.elementName + (this.className.isEmpty() ?
"" : "." + this.className + (this.attr.isEmpty() ? "" : ("[" + this.attr +
(this.attrValue.isEmpty() ? "" : "=\"" + this.attrValue + "\"") + "]"))) +
(this.attSelector == null ? "" : this.attSelector.toString()) + ">>";
      }
}
```

SAttSelector.java

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;

public class SAttSelector {
      public String att;

      public SAttSelector(String att) {
            this.att = att;
      }

      public String toString(){
            return "@" + this.att;
      }
}
```

3. Language Reference Manual

3.1 Lexical Conventions

3.1.1 Tokens

The following constitute the tokens in Spidr: identifiers, reserved keywords, constants, string literals, operators, newlines, and other separators. Blanks, spaces and horizontal and vertical tabs may be used to separate tokens. In selectors, spaces are significant otherwise they are ignored.

3.1.2 Comments

The characters `/*` introduce a comment, which terminates with the characters `*/`. Comments cannot be nested. They do not occur within a string or character literals. Any characters within these comments are ignored.

3.1.3 Identifiers

An identifier is any alpha-numeric sequence. The first character of an identifier must be a letter. Upper and lower case letters in an identifier are considered to be different. Identifiers may have any length.

3.1.4 Keywords

The following identifiers are reserved for the use as keywords, and may not be used otherwise

`url`	`int`	`main`
`loop`	`string`	`false`
`selector`	`for`	`true`
`if`	`while`	`return`
`null`	`void`	`boolean`

3.1.5 Constants

In Spidr there are integer constants, and string literals.

3.1.6 Boolean Constants

Boolean constants can hold a value of true or false.

3.1.7 Integer Constants

An integer constant may contain any numbers from 0 to 9 and is stored as a signed integer.

3.1.8 String Literals

Anything between double quotes is considered a string literal. String literals may be concatenated using the '+' sign. A string literal may not be concatenated as a different type.

3.2 Identifiers

Each primitive, object and function is represented by an identifier.

3.3 Functions

3.3.1 Defining Functions

Functions are defined by using the function keyword. They follow the syntax:

```
function type functionName ([parameter list]) { expression }
```

Example:

```
function int addTwo(int num1, int num2) {
	return num1 + num2
}
```

Functions cannot be overloaded.

3.3.2 Return Types

Functions may return any type of object. They may also be marked as void, in which case no return statement is needed in the body of the function.

3.3.3 Main Function

Spider looks for a main function with a return type of `void` to use as a entry point when running an application. If this function is not found, an error is thrown at compile-time.

3.4 Lists

Primitives and objects may be declared either as a single-value variable, or a as an array variable.

3.4.1 List Types

A list may only contain elements of a single type. If an attempt is made to concatenate two lists of different types, an error will be thrown.

3.4.2 Instantiating Lists

Lists may be instantiated with initial values by listing identifiers and/or constants, separated by commas, and surrounding them with square brackets `(“[“`and`“]”)`. The following instantiate arrays:

```
["Value1"]
[exampleValue]
["Value1", "Value2", "Value3"]
["Value1", exampleValue]
[15, 42, 54]
```

An empty array can be initialized as the following:

```
string [] me =[]
```

When instantiating a new list with initial values, all of the values must be of the same type; otherwise, a compile-time error is thrown.

3.4.3 Accessing Elements

Members of a list may be accessed by placing square brackets after the list identifier. For instance:

```
int[] values = [1, 4, 6, 7]
int singleValue = values[2]
(* singleValue is 6 *)
```

3.4.4 List Concatenation Operator (&)

Lists may be concatenated using the **+** operator, resulting in a new list. The elements from the list on the right will be at the beginning of the resulting lists.

```
string[] newList = ["val1", "val2"] + "val3" + ["val4", "val5"]
```

3.5 Expressions

3.5.1 Primary Expressions

Primary expressions are identifiers, constants, strings, or expressions in parentheses.

primary-expression
identifier
constant
string
(expression)

An identifier is a primary expression that has type pointer, object, or value. An identifier is always an lvalue as its type is always a pointer. A constant is a primary expression. A string literal is a primary expression with type pointer to char, the address to the first character in the string array. A expression surrounded by parentheses is a primary expression identical to one without them.

3.5.2 Postfix Expressions

The operators in postfix expressions group left to right.

postfix-expression:

```
primary-expression

postfix-expression[expression]

postfix-expression++

postfix-expression----
```

argument-expression-list:

```
assignment-expression

assignment-expression-list, assignment-expression
```

All of these expressions behave as they do in C.

3.5.3 Array References

An array expression followed by an expression inside of square brackets denotes an array reference. The first element of the array is held at index 0, and the length of the array can be obtained using calling list.length.

```
string[] food = [“cake”, “apple”, “tiger”]

food[0] -> returns “cake”
```

3.5.4 Equality Expressions

The notation “==” compares whether the values of the adjoining expressions are equal. When more than two expressions are listed in succession, the comparison is made between all expressions. When the type of the two expressions are not the same, false is returned.

```
string urlList = [“http://www.google.com”, “http://www.microsoft.com”]

string urlList2 = [“http://www.google.com”, “http://www.columbia.edu”]

string urlList3 = [“http://www.google.com”, “http://www.columbia.edu”]
```

```
boolean test1 = urlList[0] == urlList2[0]           -> returns True
boolean test2 = urlList == urlList2                 -> returns False
boolean test3 = urlList2 == urlLIst3                -> returns True
boolean test4 = urlList == urlList2 == urlList3     -> returns False
boolean test5 = urlList[0] == urlList               -> returns False
```

This notation can be used to compare the pointer values of urls, strings, and any other types. When more than two expressions are listed in succession, the comparison is made between all expressions.

```
string urlList = ["http://www.google.com", "http://www.microsoft.com"]
string urlList2 = ["http://www.google.com", "http://www.columbia.edu"]
string urlList3 = ["http://www.google.com", "http://www.columbia.edu"]
boolean test1 = urlList .= urlList2    -> returns False
boolean test2 = urlList .= urlList3    -> returns False
boolean test3 = urlList .= urlList     -> returns True
```

The notation "`!=`" and "`.!=`" designate the negation of the values of the "`==`" and "`.=`" operator, respectively.

3.6 Declarations

To declare an identifier, the one of the following syntaxes must be used:

```
datatype identifier
datatype identifier = expression
datatype identifier = null
```

If the initial value expression is not provided as part of the declaration, the identifier is initialized with a null value.

The following *datatype* tokens are allowed:

- `int`
- `string`
- `url`
- `element`
- `selector`

The following are valid declarations:

```
int a

int b = null

int c = 0

int testList = [4, 2, 5, 6, 74, -4]

string e = [“first”, “second”]

url f =: http://www.columbia.edu
```

3.7 Statements

Except as indicated, assume that all statements are executed in sequence. Each statement must be terminated by a semicolon.

3.7.1 Expression Statement

Most statements will be expression statements. To view the form, refer to 6.2.

Usually expression statements are a pointer to an object, value, or pointer to another list.

3.7.2 Conditional Statement

The two forms of conditional statements are:

```
if (expression) statement
if (expression) statement else statement
```

In both cases the expression is evaluated, and it if it is non-zero, then the first statement will be executed. In the second case, the second statement will be executed if the first expression is equal to zero.

3.7.3 While Statement

The while statement takes the form of:

```
while (expression) statement
```

This statement can be executed repeatedly as long as the expression never takes the value of zero.

3.7.4 Loop Statement

The loop statement takes on the following form:

$$\texttt{loop (expression}_1\texttt{, expression}_2\texttt{, expression}_3\texttt{)}\ \textit{statement}$$

The first expression specifies initialization for the loop. The second expression specifies a test, made before each iteration, where the loop will exit when the expression becomes 0. The third expression specifies incrementing that is performed after each iteration.

3.7.5 Return Statement

The return statement is used when a function returns to its caller and it takes on the following forms:

```
return
```

```
return expression
```

In the first case the value is undefined, whereas in the second case the value of the expression is returned to the caller of the function.

3.8 Scope

An object that is declared in a block has its scope restricted to that block and any sub-blocks. All functions are declared in the global scope.

3.9 Built-in Types

3.9.1 url type

The `url` type may be instantiated by placing a colon directly in front of a string literal. For instance:

```
url microsoftUrl = :"http://www.microsoft.com"
```

Appending the colon to the front of a parenthesized expression yields the same result as if there weren't any parentheses:

```
url microsoftUrl = :"http://www.microsoft.com"

url micUrl = :("http://www.microsoft.com")
```

3.9.2 Element type

The element type represents a XML-type formatted string. It may have child elements. This type can be automatically cast into a string, or filtered by applying a `selector` to it.

3.9.3 Selector type

A selector object is used to parse through an element tree and returns an array of either element or string objects that match the selection criteria. A selector is instantiated using the following syntax:

```
<<element_selector@attribute_selector>>
```

A `selector` may be applied to any `element` object, `url` object, or list of either of these two types of object.

3.9.3.1 Element-selector

This is a special selector that has its own set of token rules, separate from the rest of the language. This token may contain any combination of the following types of example token patterns:

`input` - All elements on the page of with a certain name can be selected by simply using that name. This example code returns all input controls on the page.

`div input` - If two selectors are separated by a space, it matches the first selector, then finds all of their children that match the second selector. In this example, the selector returns all inputs that are children of a `div`.

`.headerimage` - A period prefixing a string indicates that all items matching that contain the class matching that string be returned.

`[href]` - If a string is surrounded with square brackets, all elements that contain that attribute will be returned. In this case, all elements that contain the attribute `href` will be returned (though whether `href` has a value or not is not checked).

`[href=“*images*”]` - In attribute selectors, the star may be used as a wildcard selector. It matches any character(s). In this example, only elements that have an `href` attribute and the contain the word ‘images’ in this attribute will be returned.

Here is an example of how an element selector can be used to gather a list of all input html elements that exist with the class of “survey”:

```
url testUrl = :“http://www.columbia.edu/”
element[] inputFields = testUrl <div.survey input>
```

3.9.3.2 Attribute-selector (optional)

This selector is optional, and indicates whether an attribute should be read from each of the elements selected and returned. An ‘at’ sign (@) must precede the attribute selector. The selector may be the name of an attribute, or an underscore to return the contents of the attribute. For instance, the following example shows how to retrieve a string array of all hrefs from all anchors on a page:

```
url testUrl = :“http://www.columbia.edu/”
string[] links = testUrl <<a@href>>
```

3.10 Built-in Functions

3.10.1 print() function

The print function converts any object to a string and displays it in the console.

For instance:

```
print(55)
```

Output:

```
55
```

If the type is a list, it uses the notation "`[ "element1", "element2", "element3" ]`" to to show the differing elements in the list.

If the object being printed has sub-lists of objects underneath it, it will print out all child objects also, up to a depth of 5. After 5, it will show all child lists as "`[ … ]`".

Example code:

```
print(example)
```

Output:

```
[ [ 1, 2 ] , [ 30, 40 ] ]
```

4. Project Plan

4.1 Management Process

After the project was first introduced, our group met during the second week of classes in September, and during the first five minutes of initially meeting one another at the end of the lecture, we immediately started the planning process. We agreed to meet later on that week to brainstorm ideas for a functional and practical language to implement.

From then on, we met every Wednesday night at precisely 7:30 in the library to combine our ideas. Our meetings would last anywhere from 1-4 hours, and we would meet extra days if necessary, depending on the different material we were working on and if we met our weekly goal(s).

After submitting the first project proposal, having feedback from both the TA and the Professor, we agreed that our syntax needed a little sugar; it was too similar to that of java. Idea after idea, we finally came up with designs on how to make our language have a different look that would not be too intricate to implement.

Because at first everyone was still unfamiliar with the Ocaml Language, we all agreed to sit down and wrap our heads around the different files and concepts we needed to implement in order to have an operational language. Concrete tasks were assigned and checked upon at each meeting between team members. If someone was assigned a task, they would bring it to the table the following week to show the progress and/or to ask any questions.

Email and text messaging were the two forms of primary communication used between the group; each member felt comfortable enough to contact one another with questions or comments regarding the project. GoogleDocs were used to edit and share the project proposal and language reference manual, and BitBucket was the version control system we used in order to commit, push, and pull all of the code for the project.

As each file was coded, testing was conducted immediately after in order to fix errors in the beginning stages. Each member was responsible to assist in the testing and development process so that they could familiarize themselves with the code and how the compilation process functioned.

4.2 Programming Style Guide

- Keep the code neat and clean; commenting not required but helpful

- Update code regularly in order to keep team members in sync

- When committing code via BitBucket, always submit a commit messages explaining what you did and errors if any

- If ran into certain code issues, submit the issue through BitBucket so it notified each member and outlined it in the code

- Filenames should all be consistent with one another, using lowercase letters and hyphens in between words (i.e. test-if-else.spidr, test-for-loop.spidr)

- Keep all code in appropriate files within BitBucket (i.e. all test files should stay in test folder, all src files stay in src folder)

4.3 Project Timeline

4.3.1 Estimated Event Log

Event Number	Estimate Date Done	Event Name
1	9/12/2012	Finalized group, decided on first meet date
2	9/15/2012	Decided on Spidr Language
3	9/26/2012	Submitted Project Proposal
4	9/28/2012	Met with TA to go over proposal feedback
5	10/29/2012	Submitted Language Reference Manual
6	11/7/2012	"Hello World" Program working
7	11/21/2012	Parser/Scanner working
8	11/28/2012	Semantic Analysis working
9	12/19/2012	Compiler complete, Presentation, Submitted Final Report

4.3.2 Roles and Responsibilities

Name	Role and Responsibilities
Alex Dong	Responsible for Parser, Scanner (in corporation with Matt and Akshata), and Testing
Katherine Haas	Responsible for helping with SAST (in corporation with Matt), testing, and assembling final report
Matt Meisinger	Responsible for Parser, Scanner (in corporation with Alex and Akshata), SAST, Testing, Makefile
Akshata Ramesh	Responsible for Parser, Scanner (in corporation with Matt and Alex), Testing

4.4 Software Development

4.4.1 Operating System Environment

When testing, compiling and creating code, each member worked in a Linux environment. Some members used a virtual machine in order to use Linux from their laptop. However, the project has been tested on both Windows and Linux, and runs normally on both.

4.4.2 Language Used

The Language we used in order to develop and implement our language was mainly Ocaml and Java. We also used the jsoup Java library that is used for HTML parsing, finding and extracting data, and manipulating HTML elements and attributes.

4.4.3 Version Control System Used

The version control system we used in order to commit and keep all of our code in sync with all of the group members was GIT, hosted by BitBucket. It was an easy way to keep all of our source code together and organized. For file sharing, we used GoogleDocs (for documents such as the project proposal and the language reference manual) which made it efficient for editing back and forth between team members; it also allowed multiple users to edit at the same time, which was extremely useful.

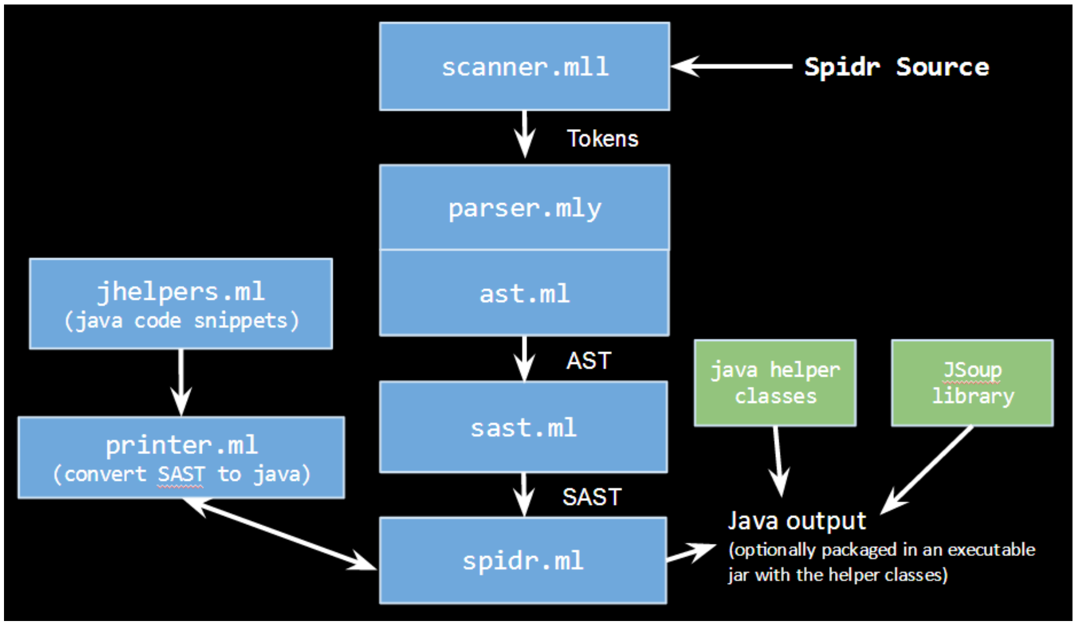
4.5 Project Log

Date	Task Accomplished
9/12/2012	Team formed
9/15/2012	First meeting, initial plan, brainstorm language ideas
9/19/2012	Second meeting, project/language title, set time and day of week to meet every week (Wednesday)
9/26/2012	Third meeting, submitted proposal, started LRM logistics
10/3/2012	Fourth meeting, talk through project objectives and goals
10/10/2012	Fifth meeting, discuss what needs to be done in terms of LRM
10/17/2012	Sixth meeting, continue to work on LRM and language syntax
10/24/2012	Seventh meeting, finalize LRM and any last minute details we wanted to change after meeting with TA
10/31/2012	Eighth meeting, submitted LRM, discussed what happens next in terms of responsibilities
11/7/2012	Ninth meeting, start to decide how to split up work and what files we need for compiler, "hello world" program working
11/14/2012	Tenth meeting, assign different roles to team members, work on compiler, start parser/scanner
11/21/2012	Eleventh meeting, parser/scanner working, start semantic analysis, continue working on compiler
11/28/2012	Twelth meeting, semantic analysis working, start creating test cases
12/5/2012	Thirteenth meeting, continue working on compiler, testing
12/7/2012	Fourteenth meeting, most of language completed, continue testing, fixing shift reduce errors
12/10/2012	Fifteenth meeting, work on final report, testing, very minimal shift reduce errors left
12/14/2012	Seventeenth meeting, continue to work on final report, testing, debugging, work on final test case for exact language
12/17/2012	Eighteenth meeting, finish up final report, continue testing all cases
12/18/2012	Nineteenth meeting, finalize report, finalize compiler, practice presentation points
12/19/2012	Project Presentation, final report due, compiler complete

5. Architectural Design

5.1 Block Diagram – Major Components

Below is a block diagram identifying the major components of the translator for Spidr.



scanner.mll
Spidr Source
Tokens
parser.mly
ast.ml
jhelpers.ml (java code snippets)
AST
java helper classes
JSoup library
sast.ml
printer.ml (convert SAST to java)
SAST
spidr.ml
Java output
(optionally packaged in an executable jar with the helper classes)

5.1.1 Interfaces Explained

The above diagram describes how the Spidr source is compiled. The spidr.ml file is the first file that is called when one gives a command to compile the Spidr source. The compiler goes through the following steps:

- The scanner.mll scans the source and produces a string of tokens from the source.
- These tokens are parsed by the parser.mly, and then converted into an abstract syntax tree that represents the entire program.
- Next, the sast.ml distills the ast into finer types, and performs the type-checking for the various type-constrained portions of Spidr such as ensuring formal and actual parameters of functions match, binary operations are done on compatible types, etc.

- The printer.ml file uses the results of the sast.ml to generate the actual java source code. The file jhelpers.ml contains Java snippets and references that complete the generated Java code. Together, the printer.ml and jhelpers.ml produce the full Java code necessary to execute the program described in the Spidr source.
- This Java source code needs the other helper classes and libraries for it to be compiled and executed. It is at this step that other helper classes, and the JSoup HTML parser library are packaged along with the generated Java code to create the complete executable .jar file.

When the Spidr source is compiled with the –s flag, only the .java file is generated without the packaging of the other dependencies. Using the –e flag instead will automatically package the .java file with the helper classes and the JSoup library as mentioned above, and will execute the resulting .jar file.

5.2 Task Distribution

Component	Implementer(s)
scanner.mll	Matt, Alex
parser.mly	Matt, Alex
ast.ml	Matt, Alex, Akshata
sast.ml	Kate, Matt, Alex, Akshata
Makefile	Matt, Alex
spidr.ml	Matt, Alex
jehlpers.ml	Matt, Alex
printer.mll	Matt, Alex
Tests	All Members
Report	Kate

6. Test Plan

As soon as we had the Scanner, Parser and Ast in place, we created a Hello World test case, to test whether it would compile out the java code as expected. From there, we built up more test cases for each of the new features that created, and exceptions that should be thrown at compile-time. We kept the details from the LRM in mind, and our primary example program as we tried to build up all of the features required to run that program.

6.1 Representative Source Programs and Java Output

The following source programs can be found in the 'demos' directory in the source package. In order to compile the java as shown below, run the compiler with the '-s' option. The following commands were used to export the java of these two files:

```
$ ./spidr -s < demos/demo2.spidr > demo1.java

$ ./spidr -s < demos/demo1.spidr > demo2.java
```

6.1.1 Demo 1

This demo prints a long array of all image urls on the page the following demo craws site specified in startUrl, and returns all active links the page, and all active links on those pages.

```
Warning: Two levels deep is a lot of links. It may take a couple minutes
to crawl any given site.

Spidr Source:

function void main() {

      string site = "http://www.cs.columbia.edu/~sedwards/software.html"
      println("Starting to crawl site: " + site)

      url startUrl = :site
      url[] children = getChildUrls(startUrl, 2)

      println("Completed!")
}

function url[] getChildUrls(url u, int depth) {

      println(u)
      if (depth == 0) {
            return [u]
      }
      else {
            string[] links = u * <<a@href>>
            url[] activeChildren = []
            loop (links l) {
                  if (live(:l)) {
                        activeChildren = activeChildren + getChildUrls(:l,
depth-1)
                  }
            }

            return [u] + activeChildren
      }
}
```

Java Output (please note that the Spidr helper java classes and JSoup library are required for this java code to compile):

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;

public class app {
  public static app spidr_app = new app();
public static SUrl[] getChildUrls(SUrl u, int depth) throws Exception {
System.out.println(u);
if (depth==0)
{
return app.array(u);
}
else {
String[] links = applyAttSelector(u,combineSelectors(new
SSelector("a","","",""),new SAttSelector("href")));
SUrl[] activeChildren = {};
for(String l : links){
if (live(new SUrl(l))) {
activeChildren = arrayConcat(activeChildren,app.getChildUrls(new
SUrl(l),depth-1));
}
}
return arrayConcat(app.array(u),activeChildren);
}
}
public static void main(String[] args) throws Exception {
            try {
String site = "http://www.cs.columbia.edu/~sedwards/software.html";
System.out.println("Starting to crawl site: "+site);
SUrl startUrl = new SUrl(site);
SUrl[] children = app.getChildUrls(startUrl,2);
System.out.println("Completed!");
            } catch (Exception e) {
                  e.printStackTrace();
            }
}



      public static SUrl[] array(SUrl... values){
            return values;
      }
      public static SSelector[] array(SSelector... values){
            return values;
      }
      public static SUrl[] arrayConcat(SUrl[] array1, SUrl[] array2){
            SUrl[] array3 = new SUrl[array1.length + array2.length];
            for(int i=0; i<array3.length; i++){
                  if(i<array1.length)
                        array3[i] = array1[i];
                  else
                        array3[i-array1.length] = array2[i-array1.length];
            }
            return array3;
      }
      public static SSelector[] arrayConcat(SSelector[] array1,
SSelector[] array2){
            SSelector[] array3 = new SSelector[array1.length +
array2.length];
            for(int i=0; i<array3.length; i++){
                  if(i<array1.length)
                        array3[i] = array1[i];
                  else
                        array3[i-array1.length] = array2[i-array1.length];
            }
            return array3;
      }

      private static String[] applyAttSelector(SUrl u, SSelector s) throws
Exception {
            Element[] urlElements =
Jsoup.connect(u.url).get().children().toArray(new Element[] {});
            return applyAttSelector(urlElements, s);
      }
      private static String[] applyAttSelector(Element[] sourceList,
SSelector a) throws Exception {
            if (a.attSelector != null) {
                  return
applyAttSelector(getElementsMatchingSelector(sourceList, a),
a.attSelector);
            }
            else if (a.innerSelector != null) {
                  return applyAttSelector(sourceList, a.innerSelector);
            }
            else {
                  throw new Exception("Internal error #1");
            }
      }
      private static String[] applyAttSelector(Element[] sourceList, 
SAttSelector a){
            List<String> ret = new ArrayList<String>();
            for (Element e : sourceList) {
                  if (e.hasAttr(a.att)) {
                        ret.add(e.attr(a.att));
                  }
            }
            return ret.toArray(new String[] {});
      }
      private static SSelector combineSelectors(SSelector s1, SSelector 
s2){
            if (s1.innerSelector == null){
                  s1.innerSelector = s2;
                  return s1;
            }
            else {
                  combineSelectors(s1.innerSelector, s2);
                  return s1;
            }
      }
      private static SSelector combineSelectors(SSelector s1, SAttSelector 
s2) throws Exception{
            if (s1.attSelector != null)
                  throw new Exception("This selector already has an 
attribute selector applied to it. Only one attribute selector may be 
applied per selector.");
            if (s1.innerSelector == null){
                  s1.attSelector = s2;
                  return s1;
            }
            else {
                  combineSelectors(s1.innerSelector, s2);
                  return s1;
            }
      }

      private static Element[] getElementsMatchingSelector(SUrl u, 
SSelector s) throws Exception {
            Element[] urlElements = 
Jsoup.connect(u.url).get().children().toArray(new Element[] {});
            return getElementsMatchingSelector(urlElements, s);
      }
      private static Element[] getElementsMatchingSelector(Element[] 
sourceList, SSelector s) throws Exception {

            List<Element> ret = new ArrayList<Element>();
            for (Element e : sourceList) {
                  boolean isMatching = true;
                  if (!s.elementName.isEmpty() && e.tagName() !=
s.elementName) isMatching = false;
                  if (!s.className.isEmpty() &&
!e.classNames().contains(s.className)) isMatching = false;
                  if (!s.attr.isEmpty() && !e.hasAttr(s.attr)) isMatching =
false;
                  if (!s.attr.isEmpty() && !s.attrValue.isEmpty() &&
e.attr(s.attr) != s.attrValue) isMatching = false;

                  if (isMatching && s.innerSelector != null) {
                        Element[] matches =
getElementsMatchingSelector(e.children().toArray(new Element[] {}),
s.innerSelector);
                        for (Element c : matches) {
                              ret.add(c);
                        }
                  }
                  else if (isMatching && s.innerSelector == null) {
                        ret.add(e);
                  }
                  else {
                        Element[] matches =
getElementsMatchingSelector(e.children().toArray(new Element[] {}), s);
                        for (Element c : matches) {
                              ret.add(c);
                        }
                  }
            }
            return ret.toArray(new Element[] {});
      }

      private static boolean live(SUrl s) {
            try {
                  java.net.HttpURLConnection connection =
(java.net.HttpURLConnection)new java.net.URL(s.url).openConnection();

                  connection.setRequestMethod("HEAD");
                  int responseCode = connection.getResponseCode();
                  if (responseCode >= 200 && responseCode < 400) {
                      return true;
                  }
                  else {
                        return false;
                  }
            } catch (Exception e) {
                  return false;
            }
      }
      private static SUrl[] live(SUrl[] u) {
            List<SUrl> ret = new ArrayList<SUrl>();
            for (SUrl s : u) {
                  if (live(s)) {
                        ret.add(s);
                  }
            }
            return ret.toArray(new SUrl[] {});
      }

      public static String[] arrayConcat(String[] array1, String[]
array2){
            String[] array3= new String[array1.length+array2.length];
            System.arraycopy(array1, 0, array3, 0, array1.length);
            System.arraycopy(array2, 0, array3, array1.length,
array2.length);
            return array3;
      }

      public static int[] arrayConcat(int[] array1, int[] array2) {
            int[] array3= new int[array1.length+array2.length];
            System.arraycopy(array1, 0, array3, 0, array1.length);
            System.arraycopy(array2, 0, array3, array1.length,
array2.length);
            return array3;
      }

      public static double[] arrayConcat(double[] array1, double[]
array2){
            double[] array3= new double[array1.length+array2.length];
            System.arraycopy(array1, 0, array3, 0, array1.length);
            System.arraycopy(array2, 0, array3, array1.length,
array2.length);
            return array3;
      }

      public static boolean[] arrayConcat(boolean[] array1, boolean[]
array2){
            boolean[] array3= new boolean[array1.length+array2.length];
            System.arraycopy(array1, 0, array3, 0, array1.length);
            System.arraycopy(array2, 0, array3, array1.length,
array2.length);
            return array3;
      }


      public static int[] array(int... values){
            return values;
      }
      public static String[] array(String... values){
            return values;
      }
      public static boolean[] array(boolean... values){
            return values;
      }

}
```

6.1.2 Demo 2

This demo prints images urls on a page and crawls a page and returns all image urls found on the page.

Spidr Source:

```
function void main() {

      string site =
"http://www.cs.columbia.edu/~sedwards/classes/2012/w4115-fall/index.html"
      println("Looking for images on site: " + site)

      string[] imageReferences = :site * <<img@src>>

      println("Found images:")
      print(imageReferences)

      println()
      println("Finished.")
}
```

Java Output (please note that the Spidr helper java classes and JSoup library are required for this java code to compile):

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;

public class app {
  public static app spidr_app = new app();
public static void main(String[] args) throws Exception {
            try {
String site = "http://www.cs.columbia.edu/~sedwards/classes/2012/w4115-
fall/index.html";
System.out.println("Looking for images on site: "+site);
String[] imageReferences = applyAttSelector(new
SUrl(site),combineSelectors(new SSelector("img","","",""),new
SAttSelector("src")));
System.out.println("Found images:");
System.out.print(java.util.Arrays.toString(imageReferences));
System.out.println("");
System.out.println("Finished.");
            } catch (Exception e) {
                  e.printStackTrace();
            }
}



      public static SUrl[] array(SUrl... values){
            return values;
      }
```

```
	public static SSelector[] array(SSelector... values){
		return values;
	}
	public static SUrl[] arrayConcat(SUrl[] array1, SUrl[] array2){
		SUrl[] array3 = new SUrl[array1.length + array2.length];
		for(int i=0; i<array3.length; i++){
			if(i<array1.length)
				array3[i] = array1[i];
			else
				array3[i-array1.length] = array2[i-array1.length];
		}
		return array3;
	}
	public static SSelector[] arrayConcat(SSelector[] array1,
SSelector[] array2){
		SSelector[] array3 = new SSelector[array1.length +
array2.length];
		for(int i=0; i<array3.length; i++){
			if(i<array1.length)
				array3[i] = array1[i];
			else
				array3[i-array1.length] = array2[i-array1.length];
		}
		return array3;
	}

	private static String[] applyAttSelector(SUrl u, SSelector s) throws
Exception {
		Element[] urlElements =
Jsoup.connect(u.url).get().children().toArray(new Element[] {});
		return applyAttSelector(urlElements, s);
	}
	private static String[] applyAttSelector(Element[] sourceList,
SSelector a) throws Exception {
		if (a.attSelector != null) {
			return
applyAttSelector(getElementsMatchingSelector(sourceList, a),
a.attSelector);
		}
		else if (a.innerSelector != null) {
			return applyAttSelector(sourceList, a.innerSelector);
		}
		else {
			throw new Exception("Internal error #1");
		}
	}
	private static String[] applyAttSelector(Element[] sourceList,
SAttSelector a){
		List<String> ret = new ArrayList<String>();
		for (Element e : sourceList) {
			if (e.hasAttr(a.att)) {
				ret.add(e.attr(a.att));
			}
		}
```

```
            return ret.toArray(new String[] {});
      }
      private static SSelector combineSelectors(SSelector s1, SSelector 
s2){
            if (s1.innerSelector == null){
                  s1.innerSelector = s2;
                  return s1;
            }
            else {
                  combineSelectors(s1.innerSelector, s2);
                  return s1;
            }
      }
      private static SSelector combineSelectors(SSelector s1, SAttSelector 
s2) throws Exception{
            if (s1.attSelector != null)
                  throw new Exception("This selector already has an 
attribute selector applied to it. Only one attribute selector may be 
applied per selector.");
            if (s1.innerSelector == null){
                  s1.attSelector = s2;
                  return s1;
            }
            else {
                  combineSelectors(s1.innerSelector, s2);
                  return s1;
            }
      }

      private static Element[] getElementsMatchingSelector(SUrl u, 
SSelector s) throws Exception {
            Element[] urlElements = 
Jsoup.connect(u.url).get().children().toArray(new Element[] {});
            return getElementsMatchingSelector(urlElements, s);
      }
      private static Element[] getElementsMatchingSelector(Element[] 
sourceList, SSelector s) throws Exception {

            List<Element> ret = new ArrayList<Element>();
            for (Element e : sourceList) {
                  boolean isMatching = true;
                  if (!s.elementName.isEmpty() && e.tagName() != 
s.elementName) isMatching = false;
                  if (!s.className.isEmpty() && 
!e.classNames().contains(s.className)) isMatching = false;
                  if (!s.attr.isEmpty() && !e.hasAttr(s.attr)) isMatching = 
false;
                  if (!s.attr.isEmpty() && !s.attrValue.isEmpty() && 
e.attr(s.attr) != s.attrValue) isMatching = false;

                  if (isMatching && s.innerSelector != null) {
                        Element[] matches = 
getElementsMatchingSelector(e.children().toArray(new Element[] {}), 
s.innerSelector);
```

```
                        for (Element c : matches) {
                              ret.add(c);
                        }
                  }
                  else if (isMatching && s.innerSelector == null) {
                        ret.add(e);
                  }
                  else {
                        Element[] matches = 
getElementsMatchingSelector(e.children().toArray(new Element[] {}), s);
                        for (Element c : matches) {
                              ret.add(c);
                        }
                  }
            }
            return ret.toArray(new Element[] {});
      }

      private static boolean live(SUrl s) {
            try {
                  java.net.HttpURLConnection connection = 
(java.net.HttpURLConnection)new java.net.URL(s.url).openConnection();

                  connection.setRequestMethod("HEAD");
                  int responseCode = connection.getResponseCode();
                  if (responseCode >= 200 && responseCode < 400) {
                      return true;
                  }
                  else {
                        return false;
                  }
            } catch (Exception e) {
                  return false;
            }
      }
      private static SUrl[] live(SUrl[] u) {
            List<SUrl> ret = new ArrayList<SUrl>();
            for (SUrl s : u) {
                  if (live(s)) {
                        ret.add(s);
                  }
            }
            return ret.toArray(new SUrl[] {});
      }


      public static int[] array(int... values){
            return values;
      }
      public static String[] array(String... values){
            return values;
      }
      public static boolean[] array(boolean... values){
            return values;
      }
}
```

6.2 Test Suites

We created a suite of 74 tests to test different aspects of our language. Of the test cases, 54 were created to test each piece of functionality within the language, and 20 were created to test the exceptions that are supposed to be thrown by the Spidr compiler. Most of the tests were created as we were working on new features, but we also created additional tests as we found bugs in the compiler, and used the test as a sort of to-do list to keep track of what we still needed to fix before the project should be considered complete.

We created a test script to run all the tests quickly. It was inspired by the MicroC test script (and named identically to it). Normally, the Makefile will be used to build the compiler and execute the test script.

```
The command...

      make testexe

... compiles the Spidr core and runs the entire suite of tests, displaying
which tests passed and failed to the user, along with a summary at the end
of the tests.  On Ubuntu it usually takes about 30 seconds to a minute to
run the tests.

The following tests are used to test functionality and should compile:
      test-array-assign-element
      test-array-concat
      test-array-declare-simple
      test-array-declare
      test-array-empty-declare
      test-array-loop-simple-str
      test-array-loop-simple
      test-array-null
      test-array-single-string
      test-array-single
      test-binop
      test-comment
      test-empty-print
      test-for-if-else
      test-for-init-out-inner-scope
      test-for-init-out
      test-for-initialize-scope
      test-for
      test-func-arg-array
      test-func-arg-nullarray
      test-func-arg
      test-func-call
      test-func-multiple-args
      test-gcd
      test-get-link-urls-for-imgs
      test-global-var-mult-func
      test-global-var
      test-hello-world-newline
      test-hello-world
      test-if-withoutbrace
      test-ifelse-withoutbrace
      test-ifelse
      test-internal-func-call-withargs
      test-internal-func-call
      test-live
      test-loop
      test-mult-global-var-init-decl
      test-mult-globalvar-init
      test-multiple-global-dec
      test-return-int
      test-return-literal-int
      test-return-literal-string
      test-return-string
      test-selector-apply-to-url
      test-selector
      test-selectors-chained
      test-shorthand-binop
      test-ultimate
      test-url-init
      test-variable-multiple
      test-variable-one-line-decl-assign
      test-variable
      test-while
      test-whitespace

The following tests test compile-time exceptions and should not compile:
      test-err-initlist-arg
      test-err-invalid-return
      test-err-loop-outside-scope
      test-err-loop-requires-arr
      test-error-func-notfound
      test-error-func-overload
      test-error-if-cond
      test-error-index-of-nonarray
      test-error-invalid-args
      test-error-listinit-nonarray
      test-error-multiple-var-wrong-type-switch
      test-error-no-return
      test-error-selector-applied-to-string-array
      test-error-type-exp-simple
```

```
test-error-type-exp
test-error-type-expected-url
test-error-type-mismatch-binop
test-error-type-undeclared-url
test-error-undeclared-var-main
test-error-var-overload-main
```

7. Lessons Learned

Below you will find each team member's perspective on the lesson(s) he/she has learned throughout the process of this project.

7.1 Alex Dong

Writing a compiler is much more complicated than simple string parsing. If you are confused on some part, chances are someone else in your group can help you. Having a different perspective on the problem is also helpful. Definitely start early on the compiler because, even though we met at least once every week after finishing the LRM, we're still crunched for time trying to finish the entire thing. OCaml takes a while to get used to, so, if you're struggling at the beginning, that's natural. The more you work on the compiler, the more you get used to it. Shift reduce and reduce conflicts are the most trivial and tedious errors to work through.

7.2 Katherine Haas

Throughout the course of this project, I have learned many lessons; the most important being I was at first very intimidated and discouraged because I barely understood Ocaml. This journey forced me to learn a language I was unfamiliar with and to also absorb the necessary components and actions taken to create a programming language. I also didn't fully comprehend how each file was pieced together and how the compiler patched each module together to allow for Spidr to be implemented properly, which is what the testing stages allowed me to understand.

I have also learned how to use a virtual machine through my personal laptop in order to run a Linux environment, how to program in Ocaml, how to test and debug a new and unfamiliar language, and how to work with team mates of different levels. Each team member had a different technique to bring to the table, allowing for a very diverse and strong project. Alex, Matt and Akshata did an awesome job of showing up promptly to each meeting, always having tasks accomplished on time, and putting forth more effort than a group could ask for.

I can honestly say this project has given me a much better interpretation of how languages are compiled, created, implemented, and the deep through process that goes behind new language ideas.

7.3 Matt Meisinger

When we started on this project, I expected Ocaml to be difficult to learn. What I didn't expect was how hard it would be to grasp the concepts of ASTs and SASTs, to design a non-ambiguous yet simple syntax, and to root out all the obscure bugs in our language. It took last month before the project was due that we were able to figure out how to implement an SAST. But once we got a hold of that concept as a group, it was smooth sailing. By the end of the project, we were amazed at how quickly we were able to identify areas to improve the syntax and functionality of our language and how fast and effectively we could implement the changes.

If I had it all to do over again, I would have used the TAs more, and consulted with them earlier about the basics of Ocaml. That may have sped up the learning process. I learned far more about GIT, Ocaml, lexical analysis, and the Linux development environment than I thought I would going into the class. Additionally, I learned how simple the rules for the syntax of a programming language are, but how much thought and planning has to go into those simple rules to make the language simple and powerful simultaneously.

For future groups, when initially planning your language, try focusing on the example programs. If you analyze them enough in the beginning, you may be able to avoid a lot of pitfalls later on in the project, and may avoid having to change your syntax. Discussions about how to create a language can sometimes be too abstract, and getting sample code down on paper most quickly shows the disconnect between the visions different team members. And finally, put aside a lot of time for the project. It just takes time to figure out Ocaml and the structure of a compiler, and there are plenty of examples from previous classes to refer to. And it takes time. But it pays off in the end.

7.4 Akshata Ramesh

Before starting this project, I was extremely excited to get a chance to create something of our own. Although that excitement fluctuated through the course of the project, I still remain quite excited about the finished product. What I feel I learnt most was a better understanding of the inner workings of a computer language. Moreover, I learnt that the parallels of scanning, parsing, and 'ast generation' that are fundamental in communication is something we use everyday.

Ocaml was a tricky one: grasping the functional programming style of OCaml is the biggest obstacle I had to face. It's one of those things that just 'clicks' all of a sudden when you're doing the most mundane things. The seemingly recursive notion of having functions operating on functions, which make more functions all seemed to make sense, slowly but surely. I feel like it is one of those things that must be understood at a deeper level, after which that understanding can bubble-up to understand concrete pieces of code.

While building parts of a compiler, another obstacle I faced was that it was hard to keep a mental map of all that was going on in the rest of the compiler. But I learned that for each functionality/enhancement that was introduced into the language, the trick was to mentally decouple it from the rest of the language, and owing to the nature of OCaml syntax/structure, it often required only minute to moderate changes. Moreover, keeping track of what each person is doing and how they are doing it is a hard task, in any team scenario. But communication with the team is what I felt really helped our project forward.

I learnt a great deal about how applying simple rules in different quantities and flavors, and abstracting this out from the user, can create a programming language! It is tedious, and time consuming, and can get very monotonous at times, but you will be highly satisfied with the end result.

7.5 Future Advice

Although this is a very cliché point to emphasize, it truly is very important; start early on your project and get a head start on all the components. It may like you have a long time to complete this project, but it is a lot of work that takes up the whole semester. If there is ever a dull moment and you aren't working on some element of the project, then you need to figure out something to work on, because in the end it will make a huge difference.

Once you have an idea, make sure you stick to it and just implement, implement, implement. Obviously, if minor ideas change such as syntax, that is perfectly fine. After all, this is a learning process and finding out which functionalities are too complicated to implement are perfectly okay. Do not be afraid to get feedback from the TA's or the Professor; they are only there to help guide you through the vigorous process. The more help you get, the stronger your project will be.

Organization is a key tool to keep throughout the project; set a day or two to meet weekly, with a consistent time and place; it makes it easier for team members to take the routine of meeting with the group and add it into their schedules.

It can be a challenge to work with other students when not everyone has the same schedule; but that it what technology is for. Using a control version system allows everyone to work remotely almost any time of the day and anywhere. Take advantage of this system and stay open to any and all communication with your group!

Lastly, *good luck* to you and keep in mind all of the different references you are given from the Professor and other teams; it definitely is useful and helps direct you on what this project entails.

Appendix A - Code Listing of Translator

Appendix A.1 – scanner.mll

```
{
        open Parser

        let lineHasKeyword = ref false
        let inSelector = ref false
}

rule token = parse
  ['\t' '\r'] { token lexbuf }
| (' ')* { if !inSelector = true then SPACE else token lexbuf }
| "/*"      { comment lexbuf }            (* Comments *)
| '\n'      { if !lineHasKeyword != true then
                                          (lineHasKeyword := false; token lexbuf)
                                  else
                                          (lineHasKeyword := false; NEWLINE) }
| '('       { LPAREN }
| ')'       { RPAREN }
| '{'       { lineHasKeyword := false; LBRACE }
| '}'       { RBRACE }
| '['       { LBRACKET }
| ']'       { RBRACKET }
| ','       { COMMA }
| '.'       { PERIOD }
| '+'       { PLUS }
| "++"          { PLUSPLUS }
| "+="          { PLUSEQ }
| '-'       { MINUS }
| "--"          { MINUSMINUS }
| "-="          { MINUSEQ }
| '_'       { UNDERSCORE }
| '*'       { TIMES }
| "*="          { TIMESEQ }
| '/'       { DIVIDE }
| "/="          { DIVIDEEQ }
| '='       { ASSIGN }
| ';'       { SEMICOLON }
| ':'       { COLON }

| ('"')[^ '"']*('"') as string_decl  { lineHasKeyword := true;
       STRING_DEC(String.sub string_decl 1 ((String.length string_decl) - 2)) }
```

```
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "string" { STRING }
| "int"    { INT }
| "url"    { URL }
| "double" { DOUBLE }
| "element" { ELEMENT }
| "selector" { SELECTOR }
| "function" { FUNCTION }
| "bool"        { BOOLEAN }
| "true"         { TRUE }
| "false"        { FALSE }
| "loop"        { LOOP }
| "@" { AT }
| ['0'-'9']+ as lxm { lineHasKeyword := true; LITERAL(int_of_string lxm) }

| ['0'-'9']* ['.'] ['0'-'9']* as lxm { lineHasKeyword :=
                                             true; D_LITERAL(float_of_string lxm) }

| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as
                                           lxm { lineHasKeyword := true; ID(lxm) }
| eof { EOF }
| _ as char { lineHasKeyword := true;
              raise (Failure("illegal character " ^ Char.escaped char)) }
```

Appendix A.2 – parser.mly

```
 %{ open Ast %}

 %token NEWLINE NEWLINE_OPT
 %token LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET LCARAT RCARAT
 %token COMMA SEMICOLON
 %token AT PERIOD UNDERSCORE ARRAYDEC
 %token <string> EL_ID
 %token <string> EL_CLASS
 %token <string> EL_ATTR
 %token <string> EL_ATT
 %token PLUS MINUS TIMES DIVIDE ASSIGN PLUSPLUS
               MINUSMINUS PLUSEQ MINUSEQ TIMESEQ DIVIDEEQ
%token EQ NEQ LT LEQ GT GEQ
%token RETURN IF ELSE FOR WHILE INT URL ELEMENT
               SELECTOR FUNCTION VOID DOUBLE STRING COLON LOOP
%token BOOLEAN TRUE FALSE
%token <string> STRING_DEC
%token <int> LITERAL
%token <float> D_LITERAL
%token <int> ARRAY_SIZE
%token <string> ID
%token EOF
%token SPACE

%nonassoc NEWLINE
%nonassoc NOELSE
%nonassoc NOVALUE
%nonassoc ELSE
%right ASSIGN
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS PLUSPLUS
%left TIMES DIVIDE
%left ARRAYDEC COMMA
%left RCARAT
%right LCARAT
%right COLON

%start program
%type <Ast.program> program

%%
```

```
program:
   /* nothing */ { [], [] }
 | program vdecl { ($2 :: fst $1), snd $1 }
 | program fdecl { fst $1, ($2 :: snd $1) }

fdecl:
   FUNCTION ftype ID LPAREN formal_list RPAREN LBRACE stmt_list RBRACE
   { {    fname = $3;
         ftype = $2;
         formals = List.rev $5;
         body = List.rev $8
  } }
 |
  FUNCTION ftype ID LPAREN formal_list RPAREN NEWLINE LBRACE stmt_list RBRACE
  { {    fname = $3;
         ftype = $2;
         formals = List.rev $5;
         body = List.rev $9
  } }

ftype:
    ID { $1 }
  | vtype { $1 }

vtype:
    INT                 { "int" }
  | DOUBLE              { "double" }
  | STRING              { "string" }
  | URL                 { "url" }
  | BOOLEAN             { "boolean" }
  | INT ARRAYDEC        { "int[]" }
  | DOUBLE ARRAYDEC     { "double[]" }
  | STRING ARRAYDEC     { "String[]" }
  | URL ARRAYDEC        { "url[]" }
  | BOOLEAN ARRAYDEC    { "boolean[]" }
  | SELECTOR            { "selector" }

   | SELECTOR ARRAYDEC { "selector[]" }

 formal_list:
     /* nothing */ { [] }
   | formal_list formals  { $2 :: $1 }
   | formal_list formals COMMA { $2 :: $1 }

 array_dec:
     ARRAYDEC      { 0 }
   | ARRAY_SIZE    { $1 }
```

```
formals:
    INT ID                 { Int($2, Noexpr) }
  | INT array_dec ID       { Array(Int($3, Noexpr), $2) }
  | DOUBLE ID              { Double($2, Noexpr) }
  | DOUBLE array_dec ID    { Array(Double($3, Noexpr), $2)   }
  | STRING ID              { Str($2, Noexpr) }
  | STRING array_dec ID    { Array(Str($3, Noexpr), $2) }
  | URL ID                 { Url($2, Noexpr) }
  | URL array_dec ID       { Array(Url($3, Noexpr), $2) }
  | BOOLEAN ID             { Bool($2, Noexpr) }
  | BOOLEAN array_dec ID   { Array(Bool($3, Noexpr), $2) }
  | SELECTOR ID            { Sel($2, Noexpr) }
  | SELECTOR array_dec ID  { Array(Sel($3, Noexpr), $2) }

vdecl:
    INT ID initial_val NEWLINE { Int($2, $3) }
  | INT ARRAYDEC ID initial_val NEWLINE { Array(Int($3,$4), 0) }
  | DOUBLE ID initial_val NEWLINE { Double($2, $3) }
  | DOUBLE ARRAYDEC ID initial_val NEWLINE { Array(Double($3,$4), 0) }
  | STRING ID initial_val NEWLINE { Str($2, $3) }
  | STRING ARRAYDEC ID initial_val NEWLINE { Array(Str($3,$4), 0) }
  | URL ID initial_val NEWLINE { Url($2, $3) }
  | URL ARRAYDEC ID initial_val NEWLINE { Array(Url($3,$4), 0) }
  | BOOLEAN ID initial_val NEWLINE { Bool($2, $3) }
  | BOOLEAN ARRAYDEC ID initial_val NEWLINE { Array(Bool($3,$4), 0) }
  | SELECTOR ID initial_val NEWLINE { Sel($2, $3) }
  | SELECTOR ARRAYDEC ID initial_val NEWLINE { Array(Sel($3,$4), 0) }

initial_val:
  /* nothing */        { Noexpr }
  | ASSIGN expr        { $2 }
  | ASSIGN ARRAYDEC    { ArrayLiteral([]) }
  | ASSIGN ARRAY_SIZE  { ArrayLiteral([IntLiteral($2)]) }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    vdecl { Vdecl($1) }
  | expr NEWLINE { Expr($1) }
  | RETURN expr NEWLINE { Return($2) }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }
| FOR LPAREN vtype assign_for SEMICOLON boo_exp SEMICOLON for_incr RPAREN stmt
        /*loop var declared in loop*/
  { For($3, (List.hd (List.tl $4)), (List.hd $4), $6, $8, $10) }
| FOR LPAREN assign_for SEMICOLON boo_exp SEMICOLON for_incr RPAREN stmt
        /*if loop var is declared outside of loop*/

     { For("None", (List.hd (List.tl $3)), (List.hd $3), $5, $7, $9) }
  | WHILE LPAREN boo_exp RPAREN stmt { While($3, $5) }
  | LOOP LPAREN ID ID RPAREN stmt { Loop($3, $4, $6) }
  | incr NEWLINE { Expr($1) }

boo_exp:
    expr EQ   expr { Binop($1, Equal, $3) }
  | expr NEQ  expr { Binop($1, Neq,   $3) }
  | expr LT   expr { Binop($1, Less,  $3) }
  | expr LEQ  expr { Binop($1, Leq,   $3) }
  | expr GT   expr { Binop($1, Greater,  $3) }
  | expr GEQ  expr { Binop($1, Geq,   $3) }

assign_for:
    ID ASSIGN expr   { [Assign($1, $3); Id($1)] }

assign:
    ID ASSIGN expr   { Assign($1, $3) }
  | ID ARRAY_SIZE ASSIGN expr { ArrayAssign($1, $2, $4) }

for_incr:
    incr       { $1 }
  | double_op  { $1 }

incr:
    ID PLUSEQ expr    { AssignOp ( $1, Add , $3 ) }
  | ID MINUSEQ expr   { AssignOp( $1, Sub, $3 ) }
  | ID TIMESEQ expr   { AssignOp( $1, Mult, $3 ) }
  | ID DIVIDEEQ expr  { AssignOp( $1, Div, $3 ) }

double_op:
    ID PLUSPLUS       { DoubleOp( $1 , Add , IntLiteral(1) ) }
  | ID MINUSMINUS     { DoubleOp( $1 , Sub, IntLiteral(1) ) }

array_literal:
  LBRACKET array_value RBRACKET { $2 }

array_value:
  /* nothing */ { [] }
  | expr COMMA array_value { $1 :: $3 }
  | expr %prec NOVALUE { [$1] }

expr:
    LITERAL             { IntLiteral($1) }
  | D_LITERAL           { DoubleLiteral($1) }
  | array_literal       { ArrayLiteral ($1) }
  | STRING_DEC          { StringLiteral($1) }
  | ID ARRAY_SIZE       { ArrayAccess($1, $2)}
  | TRUE                { BoolLiteral(true) }
  | FALSE               { BoolLiteral(false)}
  | ID                  { Id($1) }
  | boo_exp             { $1 }
  | assign              { $1 }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | double_op           { $1 }
  | COLON expr          { UrlConstructor($2) }
  | ID LPAREN actuals_opt RPAREN  { Call($1, $3) }
  | LPAREN expr RPAREN            { $2 }
  | LCARAT selector RCARAT        { $2 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }

/*  Selector declarations
    Expected in format: element.class[attrname=attrvalue] */

selector:
    elem_selector_list AT ID { Selector($1, AttrSelector($3)) }
  | elem_selector_list       { Selector($1, NoAttr) }

elem_selector_list:
    elem_selector            { [$1] }
  | elem_selector_list SPACE elem_selector { $3 :: $1 }

elem_selector:
    elem_selector_name_class { let (a,b) = $1 in ElemSelector(a, b, "", "")}
  | elem_selector_att { let (c,d) = $1 in ElemSelector("", "", c, d)}
| elem_selector_name_class elem_selector_att
```

```
              { let (a,b) = $1 in let (c,d) = $2 in ElemSelector(a, b, c, d)}

elem_selector_name_class:
    ID            { ($1, "") }
  | PERIOD ID     { ("", $2) }
  | ID PERIOD ID  { ($1, $3) }

elem_selector_att:
    LBRACKET ID RBRACKET { ($2, "") }
  | LBRACKET ID ASSIGN STRING_DEC RBRACKET { ($2, $4) }
```

Appendix A.3 – ast.ml

```
type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq

type elem_selector =
    ElemSelector of string * string * string * string

type attr_selector =
    AttrSelector of string
  | NoAttr

type expr =
    IntLiteral of int
  | StringLiteral of string
  | DoubleLiteral of float
  | ArrayLiteral of expr list
  | BoolLiteral of bool
  | Id of string
  | Binop of expr * op * expr
  | Assign of string * expr
  | ArrayAssign of string * int * expr
  | AssignOp of string * op * expr
  | DoubleOp of string * op * expr
  | Call of string * expr list
  | Selector of elem_selector list * attr_selector
  | UrlConstructor of expr
  | ArrayAccess of string * int
  | Noexpr

type vdecl =
    Int of string * expr (* name, initial value, has initial value  *)
  | Bool of string * expr
  | Str of string * expr
  | Double of string * expr
  | Url of string * expr
  | Sel of string * expr
  | Array of vdecl * int (* name, size -- Does not allow for array
                             of arrays (I think this is too much to try for this
                             project. Shall we cut it? - Matt) *)

type stmt =
    Block of stmt list
  | Vdecl of vdecl
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of string * expr * expr * expr * expr * stmt
  | While of expr * stmt
  | Loop of string * string * stmt
  | Nostmt
```

```
type func_decl = {
    fname : string;
    ftype : string;
    formals : vdecl list;
    body : stmt list;
  }

type program = vdecl list * func_decl list

(* Printing of AST -- for initial debugging only.
    Was superceded by SAST and Printer (after we figured out
    what they were). *)
let rec string_of_elem_selector = function
    ElemSelector(elem, classname, attr, attrval) ->

let rec str "(new Selector(\"" ^ elem ^ "\",\"" ^ classname ^ "\",\"" ^ attr ^ "\",\"" ^
    AttrSelector(s) -> "selector(" ^ s ^ ")"
  | NoAttr -> ""

let rec string_of_expr = function
    IntLiteral(l) -> string_of_int l
  | DoubleLiteral(d) -> string_of_float d
  | StringLiteral(s) -> "\"" ^ s ^ "\""
  | ArrayLiteral(l) -> "[" ^ String.concat ", " (List.map string_of_expr l) ^ "]"
  | BoolLiteral(l) -> (string_of_bool l)
  | Id(s) -> s
  | ArrayAccess(id, index) -> id ^ "[" ^ string_of_int index ^ "]"
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^
      (match o with
                Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
          | Equal -> "==" | Neq -> "!="
          | Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">=")
      ^ " " ^ string_of_expr e2

 | AssignOp(id, sign, e) -> id ^ "=" ^ id ^ (match sign with Add -> "+"

       | Sub -> "-" | Mult -> "*" | Div -> "/" | Equal -> "=="

| DoubleOp(id, sign, e) -> id ^ (match sign with Add -> "++" | Sub -> "--" | _ -> "")
| Assign(v, e) -> v ^ " = " ^ string_of_expr e
| ArrayAssign(id, index, e) -> id ^ "[]=" ^ string_of_expr e
| Call(f, el) ->
  (match f with
    | "print" ->
      "System.out.print(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
    | "println" ->
      "System.out.println(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
    | _ ->
      "app." ^ f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")")
| Selector(e, a) -> List.fold_left (fun acc a -> string_of_elem_selector a ^ acc) "" e
```

```
        ^ string_of_attr_selector a
  | UrlConstructor(e) -> "new Url(" ^ string_of_expr e ^ ")"
  | Noexpr -> ""

(* Todo: These helper methods could be made more efficient *)
let rec string_of_vdecl_type = function
    Int(_,_) -> "int"
  | Double(_,_) -> "double"
  | Str(_,_) -> "String"
  | Url(_,_) -> "Url"
  | Sel(_,_) -> "Selector"
  | Array(ty,_) -> (string_of_vdecl_type ty) ^ "[]"
  | Bool(_,_) -> "Bool"

let rec string_of_vdecl_name = function
    Int(n,_) -> n
  | Double(n,_) -> n
  | Str(n,_) -> n
  | Url(n,_) -> n
  | Sel(n,_) -> n
  | Array(n,_) -> (string_of_vdecl_name n)
  | Bool(n,_) -> n

let rec string_of_vdecl_expr = function
    Int(_,e) -> string_of_expr e
  | Double(_,e) -> string_of_expr e
  | Str(_,e) -> string_of_expr e
  | Url(_,e) -> string_of_expr e
  | Sel(_,e) -> string_of_expr e
  | Array(e,s) -> (if s = 0 then "" else (string_of_vdecl_expr e))
  | Bool(_, e) -> string_of_expr e

let rec string_of_vdecl = function
    Int(id, value) -> "int " ^ id ^ (if value !=
            Noexpr then "=" ^ (string_of_expr value) else "") ^ ";"
| Double(id, value) -> "double "^ id ^ (if value !=
            Noexpr then "=" ^ (string_of_expr value) else "") ^ ";"
| Str(id, value) -> "String " ^ id ^ (if value !=
            Noexpr then "=" ^ (string_of_expr value) else "") ^ ";"
| Url(id, value) -> "Url " ^  id ^ (if value !=
           Noexpr then "=" ^ (string_of_expr value) else "") ^ ";"
|     Sel(id, value) -> "Selector " ^  id ^ (if value !=
           Noexpr then "=" ^ (string_of_expr value) else "") ^ ";"
```

```
 | Array(decl, size) ->
        string_of_vdecl_type decl ^
        (match size with
            0 -> "[]"
          | size -> "[" ^ string_of_int size ^ "]") ^
        " " ^ string_of_vdecl_name decl ^
        (let expr_text = string_of_vdecl_expr decl in
          if String.length expr_text > 0 then "=" ^ expr_text else "")
        ^ ";"
 | Bool(id, value) -> "boolean " ^ id ^ (if value != Noexpr then "="^
         (string_of_expr value) else "") ^ ";"

let string_of_formals = function
     Int(id, value) -> "int " ^ id
   | Double(id, value) -> "double " ^ id
   | Str(id, value) -> "String " ^ id
   | Url(id, value) -> "url " ^ id
   | Sel(id, value) -> "Selector " ^ id
   | Array(decl, size) ->
       string_of_vdecl_type decl ^
       (match size with
           0 -> "[]"
         | size -> "[" ^ string_of_int size ^ "]") ^
       " " ^ string_of_vdecl_name decl
   | Bool(id, value) -> "boolean " ^ id

let rec string_of_stmt = function
     Vdecl(vdecl) -> string_of_vdecl vdecl
   | Block(stmts) ->
       "{\n" ^ String.concat "\n" (List.map string_of_stmt stmts) ^ "}\n"
   | Expr(expr) -> string_of_expr expr ^ ";\n";
   | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
   | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
   | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
       string_of_stmt s1 ^ "else " ^ string_of_stmt s2
   | For(t, id, e1, e2, e3, s) ->
      "for (" ^ (if t="None" then "" else t) ^" " ^ string_of_expr e1
            ^ " ; " ^ string_of_expr e2 ^ " ; " ^
let string_of_fdecl fdecl =
    "public static " ^ fdecl.ftype ^ " " ^ fdecl.fname ^ "(" ^ (if fdecl.fname
      = "main" then "String[] args" else "") ^ String.concat ", "
      (List.map string_of_formals fdecl.formals) ^ ") {\n" ^
    String.concat "\n" (List.map string_of_stmt fdecl.body) ^
    "}\n"

let string_of_program (vars, funcs) =
    "public class app {\n\n" ^
    String.concat "\n" (List.map string_of_vdecl (List.rev vars)) ^ "\n" ^
    String.concat "\n" (List.map string_of_fdecl (List.rev funcs)) ^
    "\n}"
```

Appendix A.4 – sast.ml

```
open Ast
module StringMap = Map.Make(String)

exception Function_Name_Already_Exists of string
exception Variable_Name_Already_Exists of string
exception Undeclared_Identifier of string
exception Array_Of_Arrays_Not_Allowed
exception Mismatched_Types of string
exception Not_Implemented
exception Type_Expected of string list
exception Invalid_Operation of string
exception Function_Not_Found of string
exception Index_of_NonArray of string
exception Invalid_Arguments of string
exception Initalizer_List_On_NonArray of string
exception Array_Not_Initialized of string (* name of identifier *)
exception LoopRequiresArrayTarget of string
exception Invalid_Return_Type of string list
exception Expression_Expected of string
exception Invalid_Return_Type of string list
exception Expected_No_Return
exception Invalid_Init_List_Arg
exception Variable_Not_Initalized of string

let type_of_op op = match op with
    Ast.Add -> "Arithmetic"
  | Ast.Sub -> "Arithmetic"
  | Ast.Mult -> "Arithmetic"
  | Ast.Div -> "Arithmetic"
  | Ast.Equal -> "Relational"
  | Ast.Neq -> "Relational"
  | Ast.Less -> "Relational"
  | Ast.Leq -> "Relational"
  | Ast.Greater -> "Relational"
  | Ast.Geq -> "Relational"

type t =
  Void
| Bool
| Int
| Str
| Double
| Url
| Sel of bool (* bool whether has an attribute attached to it *)
| Arr of t * int option (* type of array and size of array (if size=
None then array is uninstantiated) *)
| Elements (* Internal type only. Collection of elements when a URL is applied to a 'Sel' *)
type s_expr =
  S_IntLiteral of int
| S_StringLiteral of string
| S_DoubleLiteral of float
| S_ArrayLiteral of s_expr list * int
| S_BoolLiteral of bool
| S_Id of string * bool
| S_Binop of s_expr * op * s_expr
| S_Assign of string * s_expr
| S_ArrayAssign of string * int * s_expr
| S_AssignOp of string * op * s_expr
| S_DoubleOp of string * op * s_expr
| S_Call of string * s_expr list
| S_PrintArray of s_expr
| S_Selector of Ast.elem_selector list * Ast.attr_selector
| S_UrlConstructor of s_expr
| S_ArrayAccess of string * int
| S_ArrayConcat of s_expr * s_expr
| S_Noexpr
| S_ApplySelector of bool * s_expr * s_expr (* will return string
          array, url or elements, selector *)

type s_var_decl = {
  s_vname: string;
  s_vtype : t;
  mutable s_has_value : bool; (* tracks whether it has been
          initialized yet *)
 s_initial_value : s_expr;
 (* removed array size from here, since it whether a variable
          is an array is needed many more
places in type checking, so it needs to be a part of the type *)
(*s_array_size : int option;*) (* is None if not an array, else
           Some(arraysize) *)
(* Initial values are instantiated...
    Globals: at beginning of main function
    In functions: where they appear in the code *)
}
```

```
type s_stmt =
  S_Block of s_stmt list
| S_Vdecl of s_var_decl
| S_Expr of s_expr
| S_Return of s_expr * t
| S_If of s_expr * s_stmt * s_stmt
| S_For of t * s_expr * s_expr * s_expr * s_stmt
| S_While of s_expr * s_stmt
| S_Loop of string * s_var_decl * s_stmt
| S_Nostmt

type s_func_decl = {
  s_fname : string;
  s_return_type : t;
  s_formals : s_var_decl list;
  mutable s_body: s_stmt list;
}

type s_symbol_table = {
  mutable parent: s_symbol_table option;   (* parent scope *)
  mutable variables: s_var_decl list;
}
type s_env_def = {
  mutable functions : s_func_decl StringMap.t;
           (* function definitions *)
mutable scope : s_symbol_table;
           (* variables (also contains reference to parent scope) *)
   mutable has_selectors : bool;
   mutable has_array_concat : bool;
 }

 let include_built_in_functions map =
   StringMap.add "print" { s_fname = "print";
       s_return_type = Void;
       s_formals =
             [{  s_vname = "message";
                 s_vtype = Str;
                 s_has_value = false;
                 s_initial_value = S_Noexpr;
             }];
       s_body = []
```

```
     _
}
(StringMap.add "println" { s_fname = "println";
s_return_type = Void;
s_formals =
      [{  s_vname = "message";
          s_vtype = Str;
          s_has_value = false;
          s_initial_value = S_Noexpr;
      }];
s_body = []
}
(StringMap.add "live" { s_fname = "live";
    s_return_type = Bool;
    s_formals =
          [{  s_vname = "url_to_check";
              s_vtype = Url;
              s_has_value = false;
              s_initial_value = S_Noexpr;
          }];
    s_body = []
} map))
(* determine whether a variable exists in a scope (and checks parent scopes) *)
let rec find_variable (scope : s_symbol_table) name =
  try
    List.find (fun s -> s.s_vname = name) scope.variables
  with Not_found ->
    match scope.parent with
      Some(parent) -> find_variable parent name
    | _ -> raise Not_found

let rec confirm_variable_does_not_exist (scope : s_symbol_table) name =
  if (List.exists (fun s -> s.s_vname = name) scope.variables) then
    raise (Variable_Name_Already_Exists name)
  else
    match scope.parent with
      Some(parent) -> confirm_variable_does_not_exist parent name
    | _ -> ()

let rec loc_var_exists vars name =
        if vars = [] then false
        else if (List.hd vars).s_vname = name then true
        else (loc_var_exists (List.tl vars) name)

(* converts string to Sast type *)
let c_type = function
    "void" -> Void
  | "int" -> Int
  | "string" -> Str
  | "double" -> Double
  | "url" -> Url
  | "selector" -> Sel(false)
  | "boolean" -> Bool
  | "None" -> Void
  | "int[]" -> Arr(Int, None)
  | "string[]" -> Arr(Str, None)
  | "double[]" -> Arr(Double, None)
  | "url[]" -> Arr(Url, None)
  | "boolean[]" -> Arr(Bool, None)
  | "selector[]" -> Arr(Sel(false), None)
  | a -> raise (Type_Expected [a; ""])

let rec type_string = function
        Int -> "int"
  | Double -> "double"
  | Str -> "string"
  | Url -> "url"
  | Bool -> "bool"
  | Void -> "void"
  | Sel(_) -> "Selector"
  | Elements -> "elements"
  | Arr(a, _) -> type_string a ^ "[]"

let is_array = function
        Ast.Array(_,_) -> true
  | _ -> false

let is_s_array = function
        Arr(a, _ ) -> true

  | _ -> false


let rec name_extract = function
    Ast.Int(name, _ ) -> name
  | Ast.Double(name,_) -> name
  | Ast.Str(name,_) -> name
  | Ast.Url(name,_) -> name
  | Ast.Sel(name,_) -> name
  | Ast.Array(arr,_) -> name_extract arr
  | Ast.Bool(name,_) -> name

let rec exp_extract = function
    Ast.Int(_, expr ) -> expr
  | Ast.Double(_,expr) -> expr
  | Ast.Str(_,expr) -> expr
  | Ast.Url(_,expr) -> expr
  | Ast.Sel(_,expr) -> expr
  | Ast.Array(vdecl,_) -> exp_extract vdecl
  | Ast.Bool(_,expr) -> expr

let str_id = function
        Ast.Id(name) -> name
  | _ -> "ERROR"

let u_arr_type = function
        Arr(t,_) -> t
  | _ -> Void

let rec array_size_extract = function
    Ast.Array(_,size) -> size
  | _ -> 0

let some_size = function
        Some(size) -> size
  | _ -> 0

let requireArith op =
        let op_type = (type_of_op op) in
        if op_type = "Arithmetic" then
                op
        else
                raise (Invalid_Operation op_type)

let loosely_typed t1 t2 =
        let type1 = type_string t1 in
        let type2 = type_string t2 in
        if(type1="int" || type1="double") && (type2="int"
                       || type2="double") then true
        else false

let is_initList = function
```

```
        S_ArrayLiteral(expr_list, size) -> true
  | _ -> false

(* Validate that arguments being passed to a function are
consistant with the function's  formals. *)
let rec checkArgs fname formals args exps =
        if List.length formals <> List.length args && (not
     ((fname = "print" || fname = "println") && (List.length args)
 raise (Invalid_Arguments ("Incorrect number of arguments for " ^
                              fname ^ ". Found " ^ string_of_int
else if formals = [] && args = [] then
        exps
else if ((fname = "print" || fname = "println") && (List.length args) = 0)

         then [S_StringLiteral("")]
 else
         let (e, y) = (List.hd args) in
         let init_id = function
                 S_Id(name,init) -> if init then "" else name
          |  _ -> "" in
(if (init_id e) <> "" then raise(Variable_Not_Initalized (init_id e))
else if (is_initList e) then
        raise(Invalid_Init_List_Arg)
else if(type_string (List.hd formals).s_vtype) = (type_string y) then
        (checkArgs fname (List.tl formals) (List.tl args) (e :: exps))
(* Print and println are the only two overloaded functions.
They can accept any type. *)
else if ((fname = "print" || fname = "println") && (y <> Void)) then
       (checkArgs fname (List.tl formals) (List.tl args) (e :: exps))
else
       raise(Invalid_Arguments ((type_string y) ^ " instead of " ^ (type_string
                             (List.hd formals).s_vtype)

let get_size = function
        S_ArrayLiteral(expr_list, size) -> size
  | _ -> 0

let rec initList_size expr_list count =
        if expr_list = [] then 0
        else initList_size (List.tl expr_list) count+1

 (* extract type from VDecl *)
let rec type_extract env scope exp =
  match exp with
    Ast.Int(_,_) -> Int
  | Ast.Double(_,_) -> Double
  | Ast.Str(_,_) -> Str
  | Ast.Array(t,s) -> Arr(type_extract env scope t, Some(s))
  | Ast.Url(_,_) -> Url
  | Ast.Sel(_,s) ->
      let has_attr = get_selector_exp_has_attr env scope s in
        Sel(has_attr)
  | Ast.Bool(_,_) -> Bool

(* takes an expression (that should evaluate to a Sel()) and returns
 whether it is a Sel(true) or Sel(false). *)
 and get_selector_exp_has_attr env scope selector_exp =
   match c_expr env scope selector_exp with
       (_, Sel(i)) -> i

       (* rule: when selector is declared, it must be initialized to a type
 immediately *)
 | (S_Noexpr, _) -> raise (Expression_Expected "A selector must be initialized

 with a value immediately upon declaration.'
      | _ -> raise (Mismatched_Types "Expected selector")

 and handle_list_valued_params env scope str expr_list =
      raise Not_Implemented

 and c_expr env scope exp =
    match exp with
      IntLiteral(int_val) -> S_IntLiteral(int_val), Int
    | StringLiteral(str) -> S_StringLiteral(str), Str
    | DoubleLiteral(db_val) -> S_DoubleLiteral(db_val), Double
    | ArrayLiteral(expr_list) -> (*S_ArrayLiteral([]), Void *)
      if expr_list = [] then S_ArrayLiteral([], 0), Arr(Void, None) else
 let ele = c_expr env scope (List.hd expr_list) in
 let (ex, ty) = ele in
 let same_type = (fun exp ->
   let (e, t) = (c_expr env scope exp) in
   if(t = ty) then e
   else raise(Mismatched_Types (type_string ty ^ " " ^ type_string t))) in
 S_ArrayLiteral(List.map same_type expr_list,
             (initList_size expr_list 0) ), Arr(ty, None)
 | BoolLiteral(bool_value) -> S_BoolLiteral(bool_value), Bool
 | Id(str) ->  let vdecl = try
                find_variable scope str (* locate a variable by name *)
               with Not_found ->
                 raise (Undeclared_Identifier str) (*(String.concat ""
(List.map (fun s -> s.s_vname) scope.variables)))*)
              in
              let typ = vdecl.s_vtype in (* get the variable's type *)
              S_Id(vdecl.s_vname, vdecl.s_has_value), typ
| ArrayAccess(id, index) ->
       try find_variable scope id
       with Not_found -> raise(Undeclared_Identifier id) in
     (match arr.s_vtype with
       Arr(atype, Some(_)) -> S_ArrayAccess(id, index), atype
     | _ -> raise (Index_of_NonArray id))

 | Binop(e1,op,e2) ->
     let (e1, t1) = (c_expr env scope e1) in
     let (e2, t2) = (c_expr env scope e2) in

     (* Handle selectors.  Idea: Selectors can be applied like:
:"myurl" * <<div.banner img@src>>
    ... using asterisk to apply selector to url. *)
if op = Ast.Mult && (t1 = Url || t1 = Elements) && t2 = Sel(false) then
  S_ApplySelector(false, e1, e2), Elements
else if op = Ast.Mult && (t1 = Url || t1 = Elements) && t2 = Sel(true) then
  S_ApplySelector(true, e1, e2), Arr(Str, None)
else
  (if (match (t1, t2) with (Arr(a,_), Arr(b,_)) -> (if a = b then true else
        false) | _ -> false) then
   (env.has_array_concat <- true; S_ArrayConcat(e1, e2), t1)

          else if t1=t2 then
         (if (t1 = Int || t1 = Double) then
                (if type_of_op op = "Arithmetic" then
                        S_Binop(e1,op,e2), t1
                 else
                        S_Binop(e1,op,e2), Bool)
         else if t1 = Str && op=Ast.Add then
                S_Binop(e1,op,e2), Str
         else
                raise(Invalid_Operation (type_string t1)))
 else if (loosely_typed t1 t2) then
         S_Binop(e1,op,e2), Double
 else
raise(Mismatched_Types (type_string t1 ^ " " ^ type_string t2)))
```

```
| Assign(id,expr) ->
        let var_exists = try find_variable scope id
      with Not_found ->
        raise(Undeclared_Identifier id)
    in
        (var_exists.s_has_value <- true);
    (let (e, ty ) = (c_expr env scope expr) in
        if (is_initList e) then raise(Initalizer_List_On_NonArray id)
else if (type_string (var_exists.s_vtype)) <> (type_string ty)
        then raise(Type_Expected [(type_string (var_exists.
 else S Assign(id, e), var exists.s vtype)
| ArrayAssign(id, index, expr) ->
        let var = try find_variable scope id with Not_found ->
         raise(Undeclared_Identifier id) in

    let (e, _ ) = (c_expr env scope expr) in
    (match var.s_vtype with
        Arr(atype, _) -> S_ArrayAssign(id, index, e), atype
      | _ -> raise(Index_of_NonArray (string_of_int index)))
| AssignOp(str1,op,expr) ->
    let (e, _ ) = (c_expr env scope expr) in
      S_AssignOp(str1, requireArith(op), e), Int
| DoubleOp(str1,op,expr) ->
    let (e, _ ) = (c_expr env scope expr) in
      S_DoubleOp(str1, requireArith(op), e), Int
| Call(str,expr_list) ->
              if StringMap.mem str (include_built_in_functions
                                    env.functions) then
let f = (StringMap.find str (include_built_in_functions env.functions)) in
let f_name = f.s_fname in
let f_type = f.s_return_type in
let f_args = f.s_formals in
let list_check = (fun ex -> (c_expr env scope ex)) in
let is_arg_array arg = (match c_expr env scope arg with (_,Arr(_,_)) -> true
                                          | _ -> false) in
 if (str = "print" || str = "println") && ((List.length expr_list) = 1)
                    && (is_arg_array (List.hd expr_list)) then
   let (arg_expr,_) = c_expr env scope (List.hd expr_list) in
     (S_PrintArray(arg_expr), Void)
 else if str = "live" && ((List.length expr_list) = 1) && (is_arg_array
                      (List.hd expr_list)) then
   let (arg_expr,_) = c_expr env scope (List.hd expr_list) in
     (S_Call("live", [arg_expr]), Arr(Url, None))
 else
                  S_Call(str, List.rev (checkArgs f_name f_args
(List.map list_check expr_list) [])), f_type

              else
                      raise (Function_Not_Found str)
 | Ast.Selector(elem_selector_list,attr_selector) ->
     env.has_selectors <- true;
     let ret_type = (if attr_selector = NoAttr then Sel(false)

                    else Sel(true)) in

      S_Selector(elem_selector_list,attr_selector), ret_type
| UrlConstructor(expr) ->
    env.has_selectors <- true;
              let s_exp = c_expr env scope expr in
              let (e, ty) = s_exp in
              if ty = Str then
                      S_UrlConstructor(e), Url
              else
                      raise (Type_Expected ["url"; (type_string ty)])
| Noexpr -> S_Noexpr, Void

let get_vdecl env scope id initial (vartype : t) =
    (confirm_variable_does_not_exist scope id);
    let initial_val = c_expr env scope initial in
    let (initial_val_expr, initial_val_type) = initial_val in
        { s_vname = id;
          s_vtype = vartype;
          s_initial_value = (if (((type_string initial_val_type) <>

     (type_string vartype))
 && initial_val_expr <> S_Noexpr
 && (not ((is_s_array vartype) && ((type_string initial_val
 raise(Type_Expected [(type_string vartype); (type_string i
s_has_value = (initial_val_expr <> S_Noexpr);
}

let c_vdecl env scope v =
  match v with
    Ast.Int(id, initial) ->
      get_vdecl env scope id initial Int
  | Ast.Str(id, initial) ->
      get_vdecl env scope id initial Str
  | Ast.Double(id, initial) ->
      get_vdecl env scope id initial Double
  | Ast.Url(id, initial) ->
      get_vdecl env scope id initial Url
  | Ast.Bool(id, initial) ->
      get_vdecl env scope id initial Bool
  | Ast.Sel(id, initial) ->
      get_vdecl env scope id initial (Sel(get_selector_exp_has_attr
env scope initial))
```

```
| Ast.Array(vartype, size) ->
    (match vartype with
      Ast.Int(id, initial) ->
        (confirm_variable_does_not_exist scope id);
        get_vdecl env scope id initial (Arr(Int, Some(size)))
    | Ast.Str(id, initial) ->
        (confirm_variable_does_not_exist scope id);
        get_vdecl env scope id initial (Arr(Str, Some(size)))
    | Ast.Double(id, initial) ->
        (confirm_variable_does_not_exist scope id);
        get_vdecl env scope id initial (Arr(Double, Some(size)))
    | Ast.Url(id, initial) ->
        (confirm_variable_does_not_exist scope id);
        get_vdecl env scope id initial (Arr(Url, Some(size)))
    | Ast.Sel(id, initial) ->
        (confirm variable does not exist scope id);
get_vdecl env scope id initial (Arr(Bool, Some(size))))

let is_return = function
        S_Return(_,_) -> true
  | _ -> false

let get_return_type = function
        S_Return(_,rtype) -> rtype
  | _ -> Void

let rec c_stmt_list env scope sl =
  List.map (fun a -> c_stmt env scope a) sl
and c_stmt env scope s =
  match s with
    Block(stmt_list) ->
      let inner_scope = { parent = Some scope; variables = [] } in
        S_Block(c_stmt_list env inner_scope stmt_list)
  | Vdecl(vdec) ->
      let decl = c_vdecl env scope vdec in
      scope.variables <- (decl :: scope.variables);

      S_Vdecl(decl)
  | Expr(exp) ->
      let (e1,_) = c_expr env scope exp in
        S_Expr(e1)
  | Return(ex) ->
      (* TODO: Check return type *)
      let (e1,ex_type) = c_expr env scope ex in
        S_Return(e1, ex_type)
  | If(ex,st1,st2) ->
      let (e1,ex_type) = c_expr env scope ex in
              if ex_type = Bool then
        S_If(e1 ,c_stmt env scope st1, c_stmt env scope st2)
else
        raise (Type_Expected ["boolean"; (type_string ex_type)])
 | For(vartype,id,expr1,expr2,expr3,stmt) ->
         let inner_scope = { parent = Some scope; variables = [] } in
         (if vartype <> "None" then
     inner_scope.variables <-
       { s_vname = (str_id id);
         s_vtype = (c_type vartype);
         s_has_value = true;
         s_initial_value = S_Noexpr; } :: inner_scope.variables);
     let (expr_val1, expr_type1) = c_expr env inner_scope expr1 in
     let (expr_val2, expr_type2) = c_expr env inner_scope expr2 in
     let (expr_val3, expr_type3) = c_expr env inner_scope expr3 in
     let vtype = c_type vartype in
       if expr_type1 != Int then
         raise (Invalid_Operation "First expression in for loop must be
an integer declaration.")
 else if expr_type2 != Bool then
   raise (Invalid_Operation "Second expression in for loop must
be boolean test.")
 else if expr_type3 != Int then
   raise (Invalid_Operation "Third expression in for loop must increment.")
 else
   S_For(vtype, expr_val1, expr_val2, expr_val3, c_stmt env inner_scope stmt)
| While(expr,stmt) ->
        let inner_scope = { parent = Some scope; variables = [] } in
    let (expr_val,expr_type) = c_expr env inner_scope expr in
      S_While(expr_val,c_stmt env inner_scope stmt)
| Loop(arr, target, stmt) ->
        let decl = (find_variable scope arr) in
(if (not (is_s_array decl.s_vtype))
         then raise(LoopRequiresArrayTarget arr) else
(let inner_scope = { parent = Some scope; variables =
           [{ s_vname = target; s_vtype = (u_arr_type decl.s_
S_Loop(arr, { s_vname = target; s_vtype = (u_arr_type decl.s_vtype);
           s_has_value = true; s_initial_value
```

```
  | Nostmt -> S_Nostmt

let c_func_def env (b : Ast.func_decl) =
  if StringMap.mem b.fname env.functions then
    raise (Function_Name_Already_Exists b.fname)
  else
    let formals =
      if b.fname = "main" then
        Ast.Array(Ast.Str("args", Noexpr), 1) :: b.formals
      else
        b.formals in
    env.functions <- StringMap.add b.fname
      { s_fname = b.fname;
        s_return_type = (c_type b.ftype);
        s_formals = (List.map (fun vd ->
          { s_vname = (name_extract vd);
            s_vtype = (type_extract env env.scope vd);
            s_has_value = false;
            s_initial_value = S_Noexpr;
          }) formals);
        s_body = []
      } env.functions

let c_func_body env (b : Ast.func_decl)=
        let formals =
                  if b.fname = "main" then
                        Ast.Array(Ast.Str("args", Noexpr), 1) :: b.formals
                else
                      b.formals in
 let local_scope = {
                  parent = Some env.scope;
                  variables = (List.map (fun vd ->
                  { s_vname = (name_extract vd);
                        s_vtype = (type_extract env env.scope vd);
                        s_has_value = true;
                        s_initial_value = S_Noexpr;
                  }) formals)
               } in
  if not (StringMap.mem b.fname env.functions) then
    raise (Function_Not_Found b.fname)
  else
    let f = StringMap.find b.fname env.functions in
        let f_return = f.s_return_type in
        let f_body = c_stmt_list env local_scope b.body in
        let check_return = (fun stmt -> if (is_return stmt) then
                              (if (type_string f_return = "void") then
        raise(Expected_No_Return)
else if (type_string f_return) <> (type_string
(get_return_type stmt)) then
raise(Invalid_Return_Type [(type_string f_return);
                                (type_string (get_return_type
                    else stmt) else stmt) in
(f.s_body <- (List.map check_return f_body))
let rec v_decl_to_s_decl v_decl env =
        match v_decl with
                Ast.Int(id, expr) -> {

        s_vname = id;
        s_vtype = Int;
        s_has_value = (expr <> Ast.Noexpr);
        s_initial_value = (if expr = Ast.Noexpr then S_Noexpr else let (s, _)
 }
| Ast.Bool(id, expr) -> {

        s_vname = id;
        s_vtype = Bool;
        s_has_value = (expr <> Ast.Noexpr);
        s_initial_value = (if expr = Ast.Noexpr then S_Noexpr else let (s, _)
 }
| Ast.Str(id, expr) -> {

        s_vname = id;
        s_vtype = Str;
        s_has_value = (expr <> Ast.Noexpr);
        s_initial_value = (if expr = Ast.Noexpr then S_Noexpr else let (s, _)
 }
| Ast.Double(id, expr) -> {

        s_vname = id;
        s_vtype = Double;
        s_has_value = (expr <> Ast.Noexpr);
        s_initial_value = (if expr = Ast.Noexpr then S_Noexpr else let (s, _)
  }
| Ast.Url(id, expr) -> {

        s_vname = id;
        s_vtype = Url;
        s_has_value = (expr <> Ast.Noexpr);
        s_initial_value = (if expr = Ast.Noexpr then S_Noexpr else let (s, _)
 }
| Ast.Sel(id, expr) -> {

       s_vname = id;
       s_vtype = (Sel(get_selector_exp_has_attr env env.scope expr));
       s_has_value = (expr <> Ast.Noexpr);
       s_initial_value = (if expr = Ast.Noexpr then S_Noexpr else let (s, _)
}
```

```
| Ast.Array(vdecl, size) -> {
        s_vname = id;
        s_vtype = (Sel(get_selector_exp_has_attr env env.scope expr));
        s_has_value = (expr <> Ast.Noexpr);
        s_initial_value = (if expr = Ast.Noexpr then S_Noexpr else let (s, _)
 }
| Ast.Array(vdecl, size) -> {

        s_vname = (name_extract vdecl);
        s_vtype = Arr((type_extract env env.scope vdecl), Some(size));
        s_has_value = ((exp_extract vdecl) <> Ast.Noexpr);
        s_initial_value = (if (exp_extract vdecl) = Ast.Noexpr then S_Noexpr
 }
 let c_glob_var env a =
   if loc_var_exists env.scope.variables (name_extract a) then
     raise (Variable_Name_Already_Exists (name_extract a))
   else
env.scope.variables <- ((v_decl_to_s_decl a env) :: env.scope.variables)
(* Convert an Ast to an Sast. Throws exceptions for type errors. *)
let compile (vars, (funcs : Ast.func_decl list)) =
  let env = { functions = StringMap.empty; scope = { parent = None; variables =
 [] }; has_selectors = false; has_array_concat
 (*Declare the main function even before starting to parse through global
   variables.  Global variables may have initial instantiations, which will
   need to be added to beginning of the main function. *)
 List.iter (fun a -> c_glob_var env a) vars;
 List.iter (fun a -> c_func_def env a) funcs;
 List.iter (fun a -> c_func_body env a) funcs;
 env
(* add globals data into SAST environment *)
  (* add function declarations into environment *)
:  (* add function bodies into functions *)
```

Appendix A.5 – spidr.ml

```
open Printer

type action = Ast | Compile | Execute

(* Returns strings representing errors *)
let get_error_message = function
    Sast.Function_Name_Already_Exists(e) ->
      "Error: Function_Name_Already_Exists '" ^ e ^ "'."
  | Sast.Variable_Name_Already_Exists(e) ->
      "Error: Variable_Name_Already_Exists '" ^ e ^ "'."
  | Sast.Array_Of_Arrays_Not_Allowed ->
      "Error: Array_Of_Arrays_Not_Allowed"
  | Sast.Not_Implemented ->
      "Error: Not_Implemented"
  | Sast.Type_Expected(e) ->
      "Error: Type_Expected. Expected '" ^ (List.hd e) ^ "'
instead found '" ^ (List.hd (List.tl e)) ^ "'."
| Sast.Invalid_Operation(e) ->
    "Error: Invalid_Operation. " ^ e
| Sast.Function_Not_Found(e) ->
    "Error: Function_Not_Found. " ^ e
| Sast.Index_of_NonArray(e) ->
    "Error: Index_of_NonArray. " ^ e
| Sast.Array_Not_Initialized(e) ->
    "Error: Array_Not_Initialized. " ^ e
| Sast.Undeclared_Identifier(e) ->
    "Error: Variable '" ^ e ^ "' not declared."
| Sast.Invalid_Init_List_Arg ->
    "Error: Initializer List cannot be passed as the
argument of a function."
| Sast.Expected_No_Return ->
    "Error: Return statement found in void function."
| Sast.Invalid_Return_Type(e) ->
    "Error: Mismatched Return Type. Expected " ^ (List.hd e) ^
" but found " ^ (List.hd (List.tl e)) ^ "."
| Sast.Expression_Expected(e) ->
    "Error: Expression_Expected. " ^ e
| Sast.LoopRequiresArrayTarget(e) ->
    "Error: Variable '" ^ e ^ "' must be an array."
| Sast.Variable_Not_Initalized(e) ->
```

```
      "Error: Variable '" ^ e ^ "' has not been initialized."
  | e ->
      Printexc.to_string e

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1)
        [ ("-a", Ast);  (* Print Abstract Syntax Tree (just for

debugging) *)
("-s", Compile); (* Compile Java using SAST *)
("-e", Execute)] (* Compile using SAST then execute immediately *)
else Ast in
try
  let lexbuf = Lexing.from_channel stdin in
  let program = Parser.program Scanner.token lexbuf in
    match action with

    (* For initial testing only -- Print primitive java *)
      Ast -> let listing = Ast.string_of_program program in
                print_string listing

    (* Compile to Java *)
    | Compile ->
        ( try
            let sast = Sast.compile program in
            let listing = Printer.print_program sast in
                    print_string listing
          with
            e -> print_string (get_error_message e ^ "\n")
        )

    (* Compile to Java, package into executable Jar and execute *)
    | Execute ->
        ( try
            let sast = Sast.compile program in
            let listing = Printer.print_program sast in
(* print the java file *)
let oc = open_out "app.java" in
  Printf.fprintf oc "%s\n" listing;
  close_out oc;

  (* compile the java file *)
  ignore (Sys.command "javac -cp spidr-package.jar app.java");
```

```
    (* put it into the package *)
    ignore (Sys.command "jar uf spidr-package.jar app.class");

    (* run the file *)
    ignore (Sys.command "java -jar spidr-package.jar")
with
  e -> print_string (get_error_message e ^ "\n")
        )

with
  e -> print_string (Printexc.to_string e ^ "\n")
```

Appendix A.6 – printer.ml

```
(* This file takes prints out an SAST to a Java file.  The main function
   is named 'print_program', takes an SAST environment, and returns a string. *)

open Sast

let rec string_of_type = function
  Void -> "void"
| Bool -> "boolean"
| Int -> "int"
| Str -> "String"
| Double -> "double"
| Url -> "SUrl"
| Sel(_) -> "SSelector"
| Elements -> "Element[]"
| Arr(a,_) -> string_of_type a ^ "[]"


let string_of_op = function
    Ast.Add -> "+"
  | Ast.Sub -> "-"
  | Ast.Mult -> "*"
  | Ast.Div -> "/"
  | Ast.Equal -> "=="
  | Ast.Neq -> "!="
  | Ast.Less -> "<"
  | Ast.Leq -> "<="
  | Ast.Greater -> ">"
  | Ast.Geq -> ">="

let print_formal f =
  string_of_type f.s_vtype ^ " " ^ f.s_vname

let print_elem_sel es =
  let Ast.ElemSelector(a,b,c,d) = es in
  "new SSelector(\"" ^ a ^ "\",\"" ^ b ^ "\",\""
^ c ^ "\",\"" ^ d ^ "\")"
let rec print_elem_sel_list l =
  match l with
    a :: b when (List.length b = 0) -> print_elem_sel a
  | a :: b -> "combineSelectors(" ^ print_elem_sel_list b ^
"," ^ print_elem_sel a ^ ")"
  | _ -> ""

let print_att_sel a =
  "new SAttSelector(\"" ^ a ^ "\")"
```

```
let rec print_expr = function
   S_IntLiteral(i) -> string_of_int i
 | S_StringLiteral(str) -> "\"" ^ str ^ "\""
 | S_DoubleLiteral(d) -> string_of_float d
 | S_ArrayLiteral(arr, size) ->
     if List.length arr > 0 then
       "app.array(" ^ (String.concat "," (List.map print_expr arr)) ^ ")"
     else
       "{}"
 | S_BoolLiteral(b) -> string_of_bool b
 | S_Id(s,b) -> s
 | S_Binop(s_expr1,op,s_expr2) ->
         print_expr s_expr1 ^ (string_of_op op) ^ print_expr s_expr2
 | S_Assign(id,e) -> id ^ " = " ^ print_expr e


| S_AssignOp(s,op,s_expr) -> s ^ "=" ^ s ^ string_of_op op
 ^ (print_expr s_expr)
| S_DoubleOp(id, sign, e) -> id ^ (match sign with Ast.Add
               -> "++" | Ast.Sub -> "--" | _ -> "")
 | S_Call(f_name,exp_list) ->
     if f_name = "print" then
       ("System.out.print(" ^ print_expr (List.hd exp_list) ^ ")")
     else if f_name = "println" then
       "System.out.println(" ^ print_expr (List.hd exp_list) ^ ")"
     else if f_name = "live" then
       "live(" ^ print_expr (List.hd exp_list) ^ ")"
     else
 "app." ^ f_name ^ "(" ^ (String.concat "," (List.map print_expr exp_list)) ^ ")"
 | S_PrintArray(exp) ->
     "System.out.print(java.util.Arrays.toString(" ^ print_expr exp ^ "))"
 | S_Selector(e_l,a) ->
     (match a with
       Ast.NoAttr -> print_elem_sel_list e_l
 | Ast.AttrSelector(att) -> "combineSelectors(" ^
print_elem_sel_list e_l ^ "," ^ print_att_sel att ^ ")")
| S_UrlConstructor(exp) ->
    "new SUrl(" ^ print_expr exp ^ ")"
| S_ArrayAssign(id,index,expr) ->  id ^ "[" ^ string_of_int
 index ^ "]=" ^ (print_expr expr)
| S_ArrayAccess(id,i) -> id ^ "[" ^ string_of_int i ^ "]"
| S_ArrayConcat(a,b) -> "arrayConcat(" ^ print_expr a ^ "," ^
print_expr b ^ ")"
| S_ApplySelector(return_string_array, url, sel) ->
    if (return_string_array) then
      "applyAttSelector(" ^ print_expr url ^ "," ^ print_expr sel

^ ")"
 else
   "getElementsMatchingSelector(" ^ print_expr url ^ "," ^ print_expr sel ^ ")"
| S_Noexpr -> ""

let print_v_decl v =
  string_of_type v.s_vtype ^ " " ^ v.s_vname
  ^ (match v.s_initial_value with
        S_Noexpr -> ""
      | exp -> " = " ^ print_expr exp)

let rec print_stmt_list body =
  String.concat "\n" (List.map print_stmt body)
and print_stmt = function
  S_Block(stmt_l) ->
      "{\n"
      ^ print_stmt_list stmt_l
      ^ "\n}"
| S_Vdecl(v) -> print_v_decl v ^ ";"
| S_Expr(e) -> print_expr e ^ ";"
| S_Return(e, rtype) -> "return " ^ print_expr e ^ ";"
| S_If(e, s, S_Block([])) -> "if (" ^ print_expr e ^ ")
" ^ print_stmt s
| S_If(e, s1, s2) ->  "if (" ^ print_expr e ^ ")\n"
^ print_stmt s1 ^ "\nelse " ^ print_stmt s2
| S_For(typ,s_expr1,s_expr2,s_expr3,s_stmt) -> "for("^
(let ty = string_of_type typ in if ty <> "void" then ty ^ " " else "")
| S_While(s_expr,s_stmt) -> "while(" ^
print_expr s_expr ^ ")" ^ print_stmt s_stmt
| S_Loop(arr, target, stmt) -> "for(" ^ (string_of_type target.s_vtype)
^ " " ^ target.s_vname ^ " : " ^ arr ^ ")" ^ print
```

```
| S_Nostmt -> ""

let print_func_decl f env =
  let main_surround_try body =
    if f.s_fname = "main" && env.has_selectors then
      Jhelpers.main_try ^ body ^ Jhelpers.main_catch
    else
      body in
    "public static " ^ (string_of_type f.s_return_type) ^ " " ^ f.s_fname
    ^ "("
    ^ String.concat ", " (List.map print_formal f.s_formals)
    ^ ") " ^ (if env.has_selectors then "throws Exception " else "") ^ "{\n"
    ^ main_surround_try (print_stmt_list f.s_body)
    ^ "\n}\n"

let print program (a : Sast.s env def) =
  (if a.has_selectors = true then Jhelpers.imports else "")
^ "public class app {\n"
    ^ (let globals = (String.concat ";\n" (List.map print_v_decl
a.scope.variables)) in if globals = "" then "" else glob
^ (if a.has_selectors = true then Jhelpers.instantiate_app else "")
^ String.concat "" (List.map (fun (_,f) ->
print_func_decl f a) (StringMap.bindings a.functions))
 ^ (if a.has_selectors = true then Jhelpers.selector_functions else "")
     ^ (if a.has_array_concat = true then Jhelpers.array_concat else "")
     ^ Jhelpers.array_init
 ^ "\n}"
```

Appendix A.7 – jhelpers.ml

```
let instantiate_app =
"  public static app spidr_app = new app();\n"

let imports =
"import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Element;\n\n"

let main_try =
"                try {\n"
let main_catch =
"\n             } catch (Exception e) {
                        e.printStackTrace();
                }"

let array_init = "\n\n" ^
"
        public static int[] array(int... values){
                return values;
        }
        public static String[] array(String... values){
                return values;
        }
        public static boolean[] array(boolean... values){
                return values;
        }
"
let array_concat = "\n\n" ^
"       public static String[]
arrayConcat(String[] array1, String[] array2){
          String[] array3= new String[array1.length+array2.length];
          System.arraycopy(array1, 0, array3, 0, array1.length);
          System.arraycopy(array2, 0, array3, array1.length, array2.length);
          return array3;
  }
```

```
public static int[] arrayConcat(int[] array1, int[] array2) {
        int[] array3= new int[array1.length+array2.length];
        System.arraycopy(array1, 0, array3, 0, array1.length);
 }

 public static double[] arrayConcat(double[] array1, double[] array2){
         double[] array3= new double[array1.length+array2.length];
         System.arraycopy(array1, 0, array3, 0, array1.length);
         System.arraycopy(array2, 0, array3, array1.length, array2.length);
         return array3;
 }

 public static boolean[] arrayConcat(boolean[] array1, boolean[] array2){
         boolean[] array3= new boolean[array1.length+array2.length];
         System.arraycopy(array1, 0, array3, 0, array1.length);
         System.arraycopy(array2, 0, array3, array1.length, array2.length);
         return array3;
 }"



(* These functions are only printed in the file if at /

least one selector exists in the source *)
let selector_functions = "\n\n" ^
"
        public static SUrl[] array(SUrl... values){
                return values;
        }
        public static SSelector[] array(SSelector... values){
                return values;
        }
public static SUrl[] arrayConcat(SUrl[] array1, SUrl[] array2){
        SUrl[] array3 = new SUrl[array1.length + array2.length];
        for(int i=0; i<array3.length; i++){
                if(i<array1.length)
                        array3[i] = array1[i];
                else
                        array3[i-array1.length] = array2[i-array1.length];
        }
        return array3;
 }
```

```
public static SSelector[] arrayConcat(SSelector[] array1, SSelector[] array2){
        SSelector[] array3 = new SSelector[array1.length + array2.length];
        for(int i=0; i<array3.length; i++){
                if(i<array1.length)
                        array3[i] = array1[i];
                else
                        array3[i-array1.length] = array2[i-array1.length];
        }
}

private static String[]
applyAttSelector(SUrl u, SSelector s) throws Exception {


Element[] urlElements = Jsoup.connect(u.url).get()
.children().toArray(new Element[] {});
        return applyAttSelector(urlElements, s);
}
private static String[] applyAttSelector
(Element[] sourceList, SSelector a) throws Exception {
if (a.attSelector != null) {
        return applyAttSelector
(getElementsMatchingSelector(sourceList, a), a.attSelector);
}
else if (a.innerSelector != null) {
        return applyAttSelector(sourceList, a.innerSelector);
}
else {
        throw new Exception(\"Internal error #1\");
}
}
private static String[] applyAttSelector(Element[] sourceList, SAttSelector a){
        List<String> ret = new ArrayList<String>();
        for (Element e : sourceList) {
                if (e.hasAttr(a.att)) {
                        ret.add(e.attr(a.att));
                }
        }
        return ret.toArray(new String[] {});
}
```

```
private static SSelector combineSelectors(SSelector s1, SSelector s2){
        if (s1.innerSelector == null){
                s1.innerSelector = s2;
                return s1;
        }
        else {
                combineSelectors(s1.innerSelector, s2);
                return s1;
        }
}


private static SSelector combineSelectors(SSelector s1, SAttSelector s2)
throws Exception{
if (s1.attSelector != null)
        throw new Exception(\"This selector already has an
attribute selector applied to it.·\");
if (s1.innerSelector == null){
        s1.attSelector = s2;
        return s1;
}
else {
                combineSelectors(s1.innerSelector, s2);
                return s1;
        }
}

private static Element[] getElementsMatchingSelector(SUrl
u, SSelector s) throws Exception {
Element[] urlElements = Jsoup.connect
(u.url).get().children().toArray(new Element[] {});
return getElementsMatchingSelector(urlElements, s);
}
private static Element[]
getElementsMatchingSelector(Element[] sourceList, SSelector s) throws Exception {
List<Element> ret = new ArrayList<Element>();
for (Element e : sourceList) {
        boolean isMatching = true;
        if (!s.elementName.isEmpty() && e.tagName() !=
s.elementName) isMatching = false;


!e.classNames().contains(s.className)) isMatching = false;

if (!s.attr.isEmpty() && !e.hasAttr(s.attr)) isMatching = false;
if (!s.attr.isEmpty() && !s.attrValue.isEmpty() && e.attr(s.attr)

!= s.attrValue) isMatching = false;





            for (Element c : matches) {
                    ret.add(c);
            }
   }
   else if (isMatching && s.innerSelector == null) {
            ret.add(e);
   }
   else {
            Element[] matches = getElementsMatchingSelector(

e.children().toArray(new Element[] {}), s);

                            for (Element c : matches) {
                                    ret.add(c);
                            }
                    }
            }
            return ret.toArray(new Element[] {});
   }

   private static boolean live(SUrl s) {
            try {

java.net.HttpURLConnection connection =
 (java.net.HttpURLConnection)new java.net.URL(s.url)
                connection.setRequestMethod(\"HEAD\");
                int responseCode = connection.getResponseCode();
                if (responseCode >= 200 && responseCode < 400) {
                    return true;
                }
                else {
                        return false;
                }
        } catch (Exception e) {
                return false;
        }
}
private static SUrl[] live(SUrl[] u) {
        List<SUrl> ret = new ArrayList<SUrl>();
        for (SUrl s : u) {
                if (live(s)) {
                        ret.add(s);
                }
        }
        return ret.toArray(new SUrl[] {});
}"
```

```
                    if (!s.className.isEmpty() &&
```

```
if (isMatching && s.innerSelector != null) {
        Element[] matches = getElementsMatchingSelector
```

Appendix A.8 – Makefile

```
OBJS = jhelpers.cmo ast.cmo sast.cmo parser.cmo scanner.cmo printer.cmo spidr.cmo
spidr : $(OBJS)
        ocamlc -o spidr $(OBJS)

.PHONY : test
test : spidr testall.sh
        ./testall.sh

.PHONY : testexe
testexe : spidr testall.sh
        ./testall.sh -e

scanner.ml : scanner.mll
        ocamllex scanner.mll

parser.ml parser.mli : parser.mly
        ocamlyacc parser.mly

%.cmo : %.ml
        ocamlc -c $<

%.cmi : %.mli
        ocamlc -c $<

.PHONY : clean
clean :
        rm -f spidr parser.ml parser.mli scanner.ml testall.log \
        *.cmo *.cmi *.out *.diff app.java app.jar app.class

# Generated by ocamldep *.ml *.mli
ast.cmo:
ast.cmx:
jhelpers.cmo:
jhelpers.cmx:
parser.cmo: ast.cmo parser.cmi
parser.cmx: ast.cmx parser.cmi
printer.cmo: sast.cmo jhelpers.cmo ast.cmo
printer.cmx: sast.cmx jhelpers.cmx ast.cmx
sast.cmo: ast.cmo
sast.cmx: ast.cmx
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
```

```
spidr.cmo: scanner.cmo sast.cmo printer.cmo parser.cmi ast.cmo
spidr.cmx: scanner.cmx sast.cmx printer.cmx parser.cmx ast.cmx
parser.cmi: ast.cmo
```