HAWK Final Report

HTML is All We Know

Team Members:

- Graham Gobieski (gsg2120)
- George Yu (gy2206)
- Ethan Benjamin (eb2947)
- Justin Chang (jc4137)
- Jon Adelson (jma2215)

0. Contents

1. Introduction

As of 2015 there are over one billion websites. The vast wealth of information contained by the internet motivates the creation of "web scraping" software, whose purpose is to to analyze websites and extract pertinent information from them.

Websites hold their information in the structure of an HTML document. To find information in a web page, a developer can either traverse the structure of the document (through CSS selectors, XPath, or related XML techniques) or treat the document as a raw string and use technqiues like regular expression search. Typically, effective web scraping applications require a mix of different approaches. This often necessitates managing a variety of libraries which have mutually incompatible abstractions.

Furthermore, many web scraping algorithms are naturally expressed using a workflow which requires a non-trivial amount of boilerplate in general purpose programming languages. In particular, web scraping often boils down to finding all parts of a web document which match a certain criteria, and then taking a relevant action in response to the pattern.

HAWK—**H**TML is **A**ll **W**e **K**now aims to make web scraping easy by unifying disparate pattern matching techniques under a single abstraction and making the widely applicable pattern-action workflow an intrinsic part of the language. We are heavily inspired by AWK's pattern-action mechanism as well as its syntactic simplicity. Just as AWK greatly simplified information extraction in text documents, we hope to do the same for web documents.

HAWK aims to be both concise and type safe. To this end, HAWK is statically typed and has full type inference. The current implementation of HAWK generates Java code, which is then compiled to Java Bytecode and run a Java Virtual Machine.

2. Code Samples

HAWK uses a syntax that is very similar to the syntax of AWK. There is a BEGIN block and an END block, which respectively start and end each program, and between the two blocks are any number of pattern-action blocks.

Below is a general program with no pattern matching done on its input. This simply creates a table of integers, increments each of its odd elements by 1, and prints the elements in the table.

2.1 General Syntax

```
BEGIN{

    table = {1, 2, 3, 4, 5, 6};

    keyValueTable = {"oddSum":0,"evenSum":0};


    i = 0;

    while(i < length(table)){

        if(table[i] % 2 == 1){

                table[i] = table[i] + 1;

                keyValueTable["oddSum"] = keyValueTable["oddSum"] + i;

        }

        else{

            keyValueTable["evenSum"] = keyValueTable["evenSum"] + i;

        }

    }

}

END{

    print(table);

}
```

2.2 CSS Selectors

HAWK is built to parse HTML documents; it can do this by either searching for CSS patterns or by searching for regex. Each item that is found is stored as a table containing the id, attributes, and the id of the next child, if they exist. All element tables go into a table called this, Say I wanted to print out the inner HTML of each element with class .get-me; here is how it would be done:

```
BEGIN{ }

[@.get-me@]{

    print(this["innerhtml"]);

}

END{ }
```

2.3 Regex Patterns

Here is a HAWK program that uses a regex pattern. Say we wanted to look for the heights of certain objects in feet. We can use a regex pattern to find these like so:

```
BEGIN{ }

[/ ['0'-'9']+ ft /]{

    height = int_to_string(this);

    if( height >= 14000 && height <=15000){

        print("found");

    }

}

END{ }
```

3. Language Reference Manual

3.1 Lexical Conventions

3.1.1 Tokens

There are several tokens in HAWK: identifiers, keywords, literals, operators, and patterns. Whitespace and comments are ignored except as separators of tokens.

3.1.2 Comments

Use /* to begin a comment, and terminate it with */. Comments can extend across multiple lines. Comments cannot be nested.

3.1.3 Identifiers

An identifier is a sequence of letters, digits, dashes, and underscores. Uppercase and lowercase letters are treated differently. Identifiers may have any length, and are separated from other tokens by whitespace. An identifier must have at least one letter and begin with a letter.

3.1.4 Keywords

The following identifiers are keywords reserved for particular use:

- `BEGIN`
- `END`
- `this`
- `fun`
- `for`
- `while`
- `return`

3.1.5 Literals

There are four types of literals, described in detail below.

3.1.5.A Integer Constants

An integer constant consists of a sequence of digits that does not have leading zeros. All integers in HAWK are taken to base 10. Negative integer constants consist of a sequence of digits prefixed by a dash.

3.1.5.B Double Constants

A double constant consists of an integer part, decimal point, fractional part, and optional exponential part, which consists of an integer prefixed by an e. Either the integer or fractional part, but not both, may be missing. Both integer and fractional parts are themselves integers, separated by the decimal point. The decimal point must be present.

3.1.5.C String Literals

A string literal is a sequence of characters surround by double quotes. HAWK contains several escape sequences which can be used as characters within string literals:

- newline `\n`
- tab `\t`
- backslash `\\`
- single quote `\'`
- double quote `\"`

String literals are immutable, and thus cannot be altered. Any operations performed on a string literal will not affect the original literal but instead generate a new string.

3.1.5.D Table Literals

A table literal is a comma-separated sequence delimited by curly braces. The comma-separated sequence can take two forms.

- A comma separated sequence of zero or more literals, where each literal is of the same type. This generates a table where the values are keyed in sequential order from the integer 0 to sequence length-1.
- A sequence of comma separated key-value pairs written in the form key : val, where values are literals that must be of the same type. Keys are restricted to string types; integers are treated as strings.

3.1.6 Patterns

Patterns utilize valid Regex or CSS selector syntax and are defined with special offset characters within brackets before an action block in the pattern section of the program. Please see the following sections for a discussion on what a comprises a valid CSS selector and Regex expression.

3.1.6.A CSS Selector Patterns

HAWK implements a limited syntax of the standard W3 definition of CSS selectors. A CSS selector contains special syntax used to find elements in an HTML document which match particular criteria. In HAWK, a CSS selector pattern consists of a CSS selector enclosed by @ symbols which are themselves enclosed by brackets.

```
[@ css-selector @]
```

3.1.6.A.1 CSS Selectors

A CSS selector consists of one or more simple selector sequences chained by combinators. See below for details on simple selector sequences and combinators.

css-selector:

- *simple-selector-sequence*
- *simple-selector-sequence combinator css-selector*

3.1.6.A.2 Simple Selector Sequences

A simple selector sequence consists of a single type selector, or an optional type selector followed by one or more property selectors. These are the essential building blocks of CSS selectors.

Intuitively, a type selector is used to find HTML elements with specific tags (e.g. `<div>` or `<td>` or `<span>`) while property selectors are predicates on attributes and associated values within a tag (e.g. blah1, blah2, "something1", and "something2" in `<td blah1="something" blah2="somethingelse">`).

simple-selector-sequence:

- *type-selector*
- *type-selector property-selector-list*
- *property-selector-list*

3.1.6.A.3 Type Selectors

A type selector can either specify a specific tag type, or can select any tag using the universal selector. We allow matching on non-standard tag types named by any valid identifier, even those not typically supported in web browsers.

type-selector:

- *identifier*
- *universal-selector*

universal-selector:

- *

3.1.6.A.4 Property Selectors

Property selectors are predicates on attributes and associated values within an HTML tag.

property-selector-list:

- *property-selector*
- *property-selector property-selector-list*

property-selector:

- *attribute-exists-selector*
- *attribute-string-selector*
- *class-selector*
- *id-selector*

Attribute existence selectors check if a tag contains a given attribute.

attribute-exists-selector:

- *[identifier]*

Attribute string matching selectors check if a tag contains a given attribute, and additionally filters for attributes whose associated values have certain string properties. Most of these are self explanatory with the exception of the whitespace separated containment selector. This selects for attribute values which, when split into a list of words by whitespace, contain the chosen word (e.g. `[years ~= "1992"]` would match the element `<div years = "1488 1875 1992 1995"/>`.).

attribute-string-selector:

- *attribute-equals-selector*
- *attribute-contains-selector*
- *attribute-beginswith-selector*
- *attribute-endswith-selector*
- *attribute-whitespace-separated-contains*

attribute-equals-selector:

- [identifier = string]

attribute-contains-selector:

- [identifier *= string]

attribute-beginswith-selector:

- [identifier ^= string]

attribute-endswith-selector:

- [identifier $= string]

attribute-whitespace-separated-contains:

- [identifier ~= string]

class-selector:

- .identifier

Class selectors are used to find HTML elements whose "class" attribute contains a given value (doesn't have to be the exact value).

id-selector:

- #identifier

ID Selectors are used to find HTML elements whose "id" attributes equal a given value.er

3.1.6.A.5 Combinators

Combinators chain together multiple simple sequence selectors, and are used to find elements who have a certain hierarchical relation to other elements. For instance, the direct child combinator > can be used to find an element that is a direct child of another element in the HTML XML hierarchy (e.g. `@div[attr1="value1"] > span@` will find the span in `<div attr1="value1"> Blah blah blah <span></span></div>`).

combinator:

- *direct-child-combinator*
- *descendent-combinator*
- *direct-sibling-combinator*
- *any-sibling-combinator*

direct-child-combinator:

- >

The direct child combinator finds elements matched by a simple selector sequence that are immediate children of elements matched by another simple selector sequence.

descendent-combinator:

- `\space`

The descendent combinator finds elements matched by a simple selector sequence that are descendents of elements matched by another simple selector sequence. For instance, `@div span@` will match the nested span in `<div><span></span/></div>`.

direct-sibling-combinator:

- +

The direct sibling combinator finds elements matched by a simple selector sequence that are the next sibling of elements matched by another simple selector sequence. For instance, `@tr + span@` will match the span in `<div><tr></tr><span><span/></div>`.

3.1.6.A.6 Examples

- * : selects all elements
- *#id* : selects all elements with an id attribute that matches the provided string
    - **Example:** `@#first-name@` will select all elements with attribute `id="first-name"`
- *.class* : selects all elements with a class attribute that matches the provided string
    - **Example**: `@.first-name@` will select all elements with attribute `class="first-name"`
- *element* : selects all elements that have the provided tag name. Please see the HTML language reference manual for a complete list of valid HTML element tags.
    - **Example**: `@p@` will select for all paragraph elements
- *element1 element2* : law of the descendent. This pattern will select for all element2's that are child elements of element1
    - **Example**: `@div #first-name@` will select for all elements with id attribute, `id="first-name"` that are children of div elements
- *element1* > *element2* : strict law of the descendent. This pattern will select for the elements that are direct children of the parent element. In other words, these children cannot be grandchildren (elements nested within other elements)
- *element1* + *element2* : selects the sibling, element2, of parent, element1
- *element1* ~ *element2* : selects every element, element2, that is preceded by element1
- *[attribute]* : selects all elements with the given attribute. Example: [title] will select all elements with a title attribute

- *[attribute op value]* : selects all elements that have an attribute with a value that evaluates the expression to true.
    - *[attribute = value]* : selects elements that have attribute value equal to provided value.
    - *[attribute ~= value]* : selects elements that have an attribute value that contains provided space-separated value.
    - *[attribute |= value]* : selects elements that have an attribute value that begins with provided value.
    - *[attribute $= value]* : selects elements that have an attribute value that ends with provided value.
    - *[attribute *= value]* : selects elements that have an attribute value that contains provided value.

Please see CSS and HTML language reference manuals for additional explanation of each selector pattern.

3.1.6.B Regex Patterns

HAWK implements a limited version of the standard regex expression syntax of the AWK language. A regex expression is used to find a particular pattern in a text document using the operations defined below (please see the POSIX and AWK language reference manuals for more). Moreover, a regex expression pattern must be offset with / symbols on both sides and be contained within a bracket before an action section.

```
[/.../]{

    /*action*/

}
```

Regex expression operations may be combined and standard regex expression operator precedence will be assumed (see POSIX reference manual and note the order of the operators below) or order may be defined using parentheses. Below are a description of the operators implemented:

- *'c'* : represents a single character.
- _ : represents any character.
- *eof* : represents the end of file character.
- *"string"* : represents a literal string of characters.

- *[‘a’ - ‘z’]* : represents a range of characters. Must be contained within a pair of brackets.
- *[‘b’ ‘c’]* : evaluates to true if current character matches any character provided within brackets.
- ^ : compliments a set of characters.
- *(pattern)* : evaluates pattern inside parentheses before patterns outside. Parentheses suggest an order of evaluation.
- *pattern** : represents the kleene closure of a pattern with zero or more of the pattern present.
- *pattern+* : represents the kleene closure of a pattern with one or more of the pattern present.
- *pattern?* : represents a pattern that is optional.
- *pattern1 pattern2* : represents a pattern followed by a pattern.
- *pattern1* | *pattern2* : represents either pattern1 or pattern2

Please see the POSIX and AWK language reference manuals for additional explanation of each regex expression operator.

3.2 Syntax Notation

3.2.1 Meaning of Identifiers/Variables

Identifiers are names which can refer to functions, variables, and table fields. Each identifier is a string consisting of digits, letters, and underscores, and has to start with a letter.

Variables are storage locations that contain values. Depending on where in a program variables are initialized, they are either global or local to a particular scope. See Storage Scope for more details.

HAWK is statically typed, which means that every variable has a type. The type of a variable determines the meaning and behavior of its values, and also the nature of storage needed for those values.

3.2.2 Storage Scope

The visibility of an identifier and liftetime of a variable's storage depends on where a variable is initialized. If a variable is initialized within a *BEGIN* or *END* block, it is a global variable. A global variable can be accessed by any part of the program below the global variable's initialization. It's storage stays alive throughout the entire execution of the program.

If a variable is initialized within any block other than a *BEGIN* or *END* block, it is a local variable.

A local variable can be accessed within the scope it is initialized, at or below its initialization. Its storage will be destroyed at the end of the scope.

3.2.3 Basic Types

There three basic types in HAWK:

- integer
- double
- string

And one parameterized type:

- table T

Where T can be any type.

Integers are 32-bit signed two's complement integer. Doubles are double-precision 64-bit IEEE 754 floating point. We will refer to doubles and integers as arithmetic types.

Strings are a sequence of 0 or more unicode characters. They are guaranteed to occupy O(n) space in memory, where n is the number of characters in the string.

Both arithmetic types and strings are immutable, which means that their value cannot be changed once they are created. When a variable with an immutable type is assigned a new value, the old value and underlying storage are destroyed.

A table of type T is an associative array in which keys are of type string and values are of type T. Tables, unlike the immutable types, are objects and are mutable. This means that variables do not contain tables, but rather contain references to tables. Assigning a table to a variable results in that variable storing a reference to that table. Similarly, when tables are passed as parameters to functions or returned from functions, the respective parameters and return values are references. In each of these operations, there is no copying of internal table data, only copying of references.

HAWK uses reference counting to keep track of how many variables store references to the same table. When a table no longer has any variables referencing it, the underlying storage for the table is destroyed.

3.2.4 Automatic Conversions

3.2.4.A Promotion of Integers in Mixed Arithmetic Expressions

In mathematical binary expressions where one operand is an integer and the other operand is a double, the integer will be automatically converted to a double value. The conversion will be performed using the the built-in function `int_to_double`.

3.2.4.B String Conversion in String Concatentation Expressions

In binary addition expressions where one operand is a string, and the other operand is not a string, the non-string will be automatically converted to a string value. Tables will be converted using `table_to_string`, integers using `int_to_string`, doubles using `double_to_string`. Using `int_to_string` or `double_to_string` on a string will convert the biggest substring starting from the front of the string that is a valid int/double into an int/double. For example, `int_to_string("123f") = 123`.

3.3 Expressions

The precedence of expression operations is the same as the order of the major subjections of this section. Within each subsection, operators have the same precedence. Left or right associativity will be specified in each of the subsections for each operator.

3.3.1 Primary Expressions

Primary Expressions are identifiers, literals, strings, or expressions in parentheses.

primary expressions:

- *identifier*
- *literal*
- *string*
- *(expression)*

3.3.2 Postfix Expressions

Operators in postfix expressions group left to right.

postfix-expression:

- *primary-expression*
- *identifier[expression]*
- *identifier[expression_1]...[expression_n]*
- *identifier(argument-expression-list)* //arglist is optional

3.3.2.A Table References

A identifier followed by one or more square brackets, each containing an expression, is an expression denoting a subscripted table reference. The expressions in square brackets must be a table key of type string or an integer (which will be automatically converted to a string). The

identifier must reference a table. The whole expression is the type of value type of the table nested by the number of indices provided.

3.3.2.B Function Calls

A function call is a postfix expression (function designator) followed by parentheses containing a possibly empty, comma-separated list of expressions which constitute the arguments to the function. A declaration for the function must previously exist in the global scope. Recursive functions are permitted whenever no arguments are empty tables.

The term *argument* refers to an expression passed by a function call, the term *parameter* refers to an input object or its identifier received by the function definition.

3.3.3 Unary Operators

Expressions with unary operators group right to left.

unary expression:

- *postfix expresssion*
- *unary-operator unary-expression*

unary-operator: -

3.3.3.A Unary Minus Operator

The operand of the unary - operator must be of type int or double, and the result is the negative of its operand. An integral operand undergoes integral promotion. The type of the result is the type of the promoted operand.

3.3.4 Multiplicative Operators

The multiplicative operators *, /, and % group left-to-right.

multiplicative-expression:

- *multiplicative-expression * unary-expression*
- *multiplicative-expression / unary-expression*
- *multiplicative-expression % unary-expression*

The operands of *, /, and % must be of type integer or double. The usual arithmetic conversions are performed on the operands.

The binary * operator denotes multiplication.

The binary / operator yields the quotient, and the % operator the remainder, of the division of

the first operand by the second; if the second operand is 0, the result is undefined. Otherwise, it

is always true that (a/b)*b + a%b is equal to a. If both operands are non-negative, then the remainder is non-negative and smaller than the divisor, if not, it is guaranteed only that the absolute value of the remainder is smaller than the absolute value of the divisor.

3.3.5 Additive Operators

The additive operators + and - group left-to-right. If the operands have type int or double, the usual arithmetic conversions are performed.

additive-expression:

- *multiplicative-expression*
- *additive-expression* + *multiplicative-expression*
- *additive-expression* - *multiplicative-expression*

The result of the + operator is the sum of the operands. For strings, the sum is defined as the concatenation of the two strings.

3.3.6 Relational Operators

The relational operators group left-to-right. a<b<c is parsed as(a<b)<c, and evaluates to either 0 or 1.

relational-expression:

- *additive-expression*
- *relational-expression* < *additive-expression*
- *relational-expression* > *additive-expression*
- *relational-expression* <= *additive-expression*
- *relational-expression* >= *additive-expression*

The operators < (less), > (greater), <= (less or equal) and >= (greater or equal) all yield 0 if the specified relation is false and 1 if it is true. The type of the result is int. The usual arithmetic conversions are performed on arithmetic operands.

3.3.7 Equality Operators

equality-expression:

- *relational-expression*
- *equality-expression* == *relational-expression*

- *equality-expression != relational-expression*

The == (equal to) and the != (not equal to) operators are analogous to the relational operators except for their lower precedence. (Thus a<b == c<d is 1 whenever a<b and c<d have the

same truth-value.)

3.3.8 Logical AND Operators

logical-AND-expression:

- *equality-expression*
- *logical-AND-expression && equality-expression*

The && operator groups left-to-right. It returns 1 if both its operands compare unequal to zero, 0 otherwise.

&& guarantees left-to-right evaluation: the first operand is evaluated, including all side effects; if it is equal to 0, the value of the expression is 0. Otherwise, the right operand is evaluated, and if it is equal to 0, the expression's value is 0, otherwise 1.

The operands must be of type int or double, but don't have to be of the same type.

The result is int.

3.3.9 Logical OR Operators

logical-OR-expression:

- *logical-AND-expression*
- *logical-OR-expression* || *logical-AND-expression*

The || operator groups left-to-right. It returns 1 if either of its operands compare unequal to zero, and 0 otherwise.

|| guarantees left-to-right evaluation: the first operand is

evaluated, including all side effects; if it is unequal to 0, the value of the expression is 1. Otherwise, the right operand is evaluated, and if it is unequal to 0, the expression's value is 1, otherwise 0.

The operands must be of type int or double, but don't have to be of the same type.

The result is int.

3.3.10 Built-In Functions

HAWK includes several built-in functions that are reserved and are specially-interpreted by the

compiler. These include:

- `print("...")` : prints a strings to standard output.
- `charAt(string s, int i)` : returns the character at index i of string s
- `exists(table[i])` : checks to see if a table element exists. Returns 1 if so, otherwise 0.
- `stringEqual(string s1, string s2)` : checks if s1 and s2 are equal. Returns 1 if so, otherwise 0.
- `stringToInt(string s)`: If s can be parsed as an integer, returns that integer. Otherwise throws a runtime exception.
- `length(table)` : returns the length of a table
- `length(string)` : returns the length of a string
- `keys(table)` : returns a table containing every key of the passed table. The returned table uses the index as a the key and the return key value as the value.

3.4 Declarators

Declarations give a specific meaning to each identifier.

declaration:

- declarator

declarator:

- identifier = expression
- fun identifier(arg-list) compound-statement

For identifier = expression, the type of identifier is the result of the expression on the RHS. If this identifier is later used in an expression, it yields a type that is the same as when it was created. You must initialize a variable when you create it.

3.4.1 Function Declarators

fun identifier(arg1, arg2, ..., argn) compound-statement

The type of a function call, identifier(arg1, arg2, ..., argn), is the type of the expression in the return statement. The types of arg1, arg2,..., argn are inferred within the compound-statement. The names of arg1, arg2, ..., argn must be valid HAWK identifiers.

All arguments to a function are call-by-value. When tables are passed as

arguments, the reference to the table is passed by value. See the Syntax

Section for more information.

3.5 Statements

Statements are executed in sequence.

statement:

- *expression-statement*
- *assignment-statement*
- *compound-statement*
- *conditional-statement*
- *while-statement*
- *return-statement*
- *pattern-statement*
- *BEGIN compound-statement*
- *END compound-statement*

3.5.1 Expression Statements

expression-statement:

- *expression*

Expression statements will typically be a function call but can be any arbitrary expression.

3.5.2 Assignment Statement

assignment-statement:

- *identifier = expression*

Assignment statements are used to declare a variable or they

can be used to update the value of a variable that already exists.

3.5.3 Compound Statements

compound-statement:

- *statement-list*

statement-list:

- *{statement}*
- *{statement; statement-list}*

Compound statements are used to write several statements when

one statement is expected.

3.5.4 Conditional Statement

conditional-statement:

- `if` *(expression) compound-statement*
- `if` *(expression) compound-statement* `else` *compound-statement*

The first substatement is executed if the expression is not

equal to zero, otherwise the second statement is executed.

The second statement can either be another Conditional Statement

or a Compound Statement.

3.5.5 while Statement

while-statement:

- *while (expression) compound-statement*

The compound-statement is executed until the expression is not

equal to 0.

3.5.6 return Statement

return-statement:

- *return expression*

Return statements are only used within function bodies.

3.5.7 Pattern Statement

pattern-statement:

- *pattern-expression compound-statement*

Pattern is a pattern to match against and the compound-statement is executed

each time that the patern is matched. Refer to Expressions and Program Structure sections for additional information.

3.5.8 BEGIN Statement

BEGIN compound-statement

Refer to Expressions section for additional information.

3.5.9 END Statement

END compound-statement

Refer to Program Structure section for additional information.

3.6 Program Structure

3.6.1 General Structure

All programs must contain three sections: a begin section, a pattern section, and an end section. Programs must contain these in the order given and may not have more than one begin or end section. Following are a syntactic outline of such a structure and descriptions of each section.

```
BEGIN{

    /*action*/

}

[/*pattern*/]{

    /*action*/

}

END{

    /*action*/

}
```

3.6.1.A Begin Section

This section is executed first. Functions and variables may be declared in this section and defined and made available to other sections. In other words, variables and functions defined in this section are visible in the pattern and end sections. The section may be empty and all normal syntax previously defined should be followed.

3.6.1.B Pattern Section

This section is executed in the order and has one or more pattern blocks that begin with a CSS or Regex expression in brackets and are followed by an optional-empty action. The patterns must align with the syntax provided in the following pattern sections. The action will be executed after each instance of such pattern has been found in the given file. The action block should contain normal syntax previously defined. Functions may not be defined in this section nor can patterns be nested. Relevant information returned by the pattern match can be accessed using the `this` keyword data structure. See below for further information on access to this structure.

***this*:**

A table of relevant information returned by the pattern. In the case of a regex expression this is simply a table containing a single element of type string. For a CSS selector, on the other hand, this table is populated with the following keys and values:

- `this["id"]`: return id of the found element
- `this["class"]`: return the class(es) of the found element
- `this[/*custom attribute*/]`: return a custom attribute of the found element. This may not exist.

3.6.1.C End Section

This section is executed last after all pattern sections have been executed, this section may have include new functions and variables as well as references to previously defined (in begin or pattern sections) functions and variables. New functions and variable only are visible in the end section and normal syntax previously defined should be implemented.

3.7 Sample Programs

3.7.1 Sample 1: General Syntax

Creates a table, adds elements to this table each time a new line is encountered and prints this table. Notice that tables are mutable and how values of a table do not have to be of the same type.

```
BEGIN{

    table = {1, 2, "hello", 70, 90, 100, 100.1};

}

[\'\n'\]{

    table[length(table)] = table[length(table)-1]+10;

}

END{

    print(table);

}
```

3.7.2 Sample 2: CSS Selectors

Looks for a certain CSS pattern and populates a global table with a count derived from the pattern. Then it prints this table.

```
BEGIN{

    counts = { }

}

[@ table . wikitable > tr @]{

    state = this["title"];

    if(exists(counts[state])){

        counts[state] = counts[state]+1;

    }else{

        counts[state] = 0;

    }

}

END{

    len = length(counts) - 1;

    while(len >= 0){

        print(”STATE: ” + state + ”, COUNT: ” + counts[state]);

        len = len - 1;

    }

}
```

3.7.3 Sample 3: Regex

Looks for a regex expression, converts the output of the pattern to an integer and prints the integer if it satisifies a condition.

```
BEGIN{ }

[/ ['0'-'9']+ ft /]{

    height = int_to_string(this);

    if( height >= 14000 && height <=15000){

        print("found");

    }

}

END{ }
```

3.8 References

The following are helpful references that may have been previously refered to in above sections.

3.8.1 AWK Language Reference Manual

HAWK was inspired by AWK and, as such, the AWK LRM is a good way to start to understand the structure and syntax of HAWK programs.

http://www.gnu.org/software/gawk/manual/gawk.html

3.8.2 C Language Reference Manual

HAWK grammar and syntax is very similar to C. See the C LRM as further reference material.

See *The C Programming Language 2nd Edition* by Brian Kernighan and Dennis Ritchie

3.8.3 CSS Reference Manual

HAWK implements a limited set of CSS selectors. See the CSS reference manual for a fuller explanation of the implemented selectors.

https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

3.8.4 LUA Language Reference Manual

HAWK tables take inspiration from Lua tables, and just as in Lua tables are the only built in data structure in HAWK. See the Lua LRM as further reference material.

http://www.lua.org/manual/5.3/

3.8.5 POSIX Reference Manual

HAWK implements a limited set of Regex expressions. Regex operators are further defined in the POSIX reference manual.

https://www.gnu.org/software/guile/manual/html_node/POSIX.html

4. Project Plan

4.1 Process

4.1.1 Planning Process

Our overall plans were guided partially by Richard Townsend's advice and partially by which features we knew would be difficult or easy to implement. Our plan was to make a well-defined Language Reference Manual that would guide all language feature implementation, quickly get a scanner and parser up and running, and then implement all language features, starting with basic ones, such as integers and strings, and moving onto more complicated ones, such tables, functions, and CSS/regex.

4.1.2 Specification Process

We defined very clearly in our Language Reference Manual the syntax and semantics of the language. Although we tried our best to anticipate all potential issues in the LRM, we did have to make certain changes to our language as the project unfolded. For example, we had initially decided that tables could hold any particular value but ended up changing it so that any particular table must hold one particular type. However, the majority of the final language specification ended up being what we initially stated in the LRM.

4.1.3 Development Process

We moved through development stages in what we viewed to be a necessary as well as logical order. The first things to be completed were the lexer and parser. We next built the interface for the Semantic Abstract Syntax Tree, the semantic checker that would return an object of type SAST, as well as the beginnings of the code generation. The rest of the project was spent building up the SAST, and writing code to do semantic checking and code generation for the SAST types.

4.1.4 Testing Process

We built our initial test suite immediately after creating our "hello world" program.

All of our tests programs print out a result, which is compared to the correct output

held in a different file. We then created a program which would compile all

of our test programs and compare the results to all of the correct outputs. As we

caught bugs in our programs or thought of additional functionality that we wanted to

ensure would be supported, we added more tests to the test suite.

4.2 Team Responsibilities

We ended up splitting up work primarily based on feature implementation.

| Team Member | Feature |

|-------------|------------------|

| Graham | Regex |

| Ethan | Tables/Functions |

| George | CSS |

| Jon | Functions |

| Justin | CSS |

Everyone played a role in parsing, scanning, testing, semantic checking, and code generation

4.3 Programming Style Guide

We tried to use consistent programming style so that each segment was easy to read for each group member, although we did not have an official guide that we required everyone to use.

We wrote comments before each section to clarify what the purpose of each block of code was.

4.4 Software Development Environment

We used OCaml to write our compiler, including OCamlyacc and OCamllex. We also used shell scripts to run our test suite.

4.5 Timeline

| Date | Project Feature |

|--------------|---|

| September 30 | Language Reference Manual |

| October 18 | Scanner/Parser |

| November 16 | Hello World code generation, testing suite created |

| November 23 | SAST created and beginning of semantic checking |

| December 11 | Regex complete |

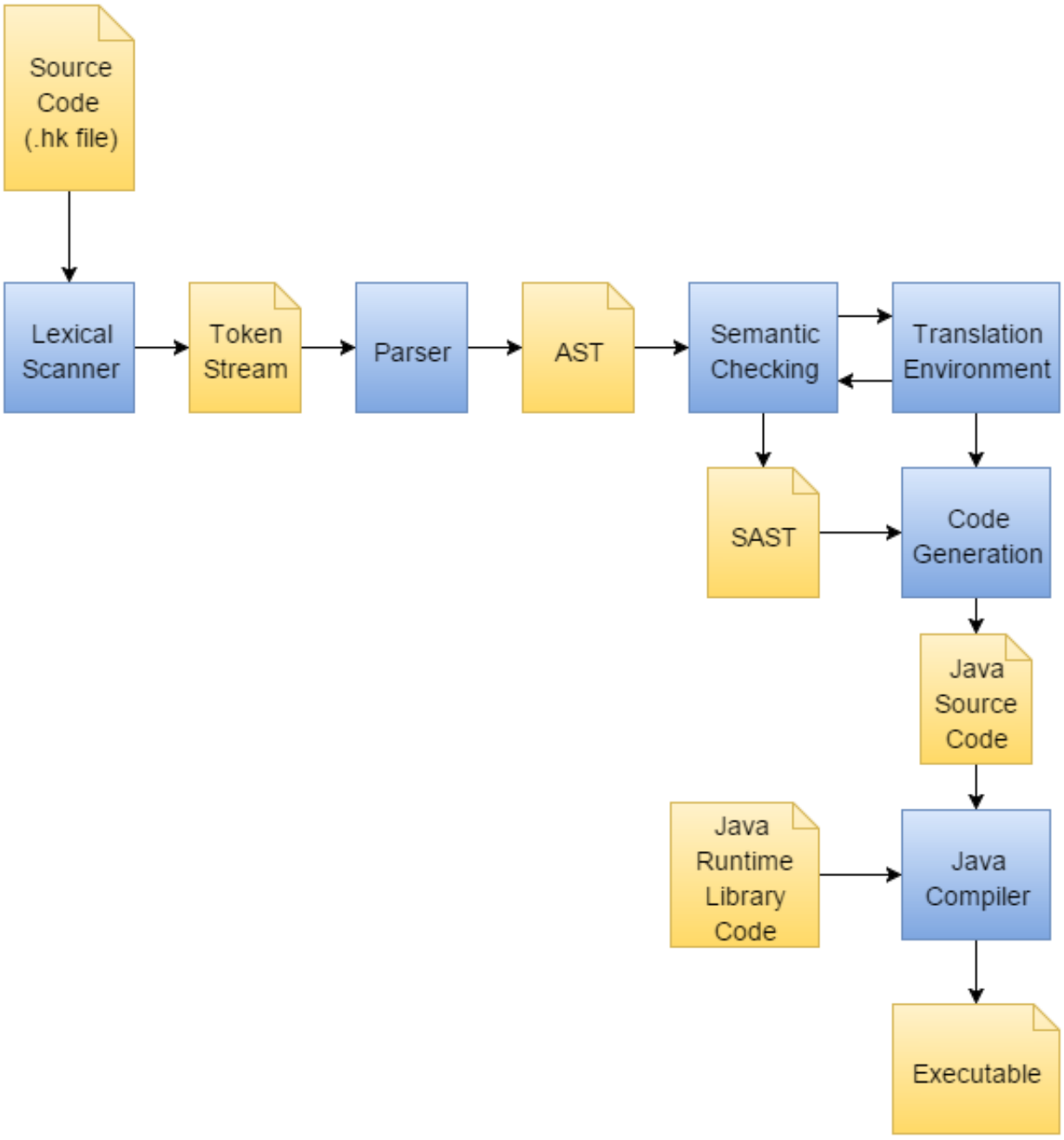
| December 17 | Type inference complete, functions complete, CSS complete |

| December 18 | All tests pass |

| December 20 | Final Report completed |

5. Architectural Design

5.1 Block Diagram



5.2 Lexical Scanner

The lexical scanner is written in OCamllexx. It takes a HAWK program as input and generates a stream of lexical tokens. It has four states of operation depending on what part of code it is reading: action code scanning, CSS pattern scanning, regex pattern scanning, and comment scanning. Action code produces a stream of identifiers, keywords, and numerical constants and operators. CSS and regex code produce a stream of pattern-relevant operators and identifiers.

Comments are treated as whitespace and ignored.

5.3 Parser

The parser is written in OCamlyacc. It takes a stream of lexical tokens as input and produces a HAWK abstract syntax tree (AST) as output. If this stage succeeds, we know that the the pattern-action segments of the source program are gramatically (if not semantically) valid. We also know that any CSS or regex patterns used are valid CSS or regex patterns. This compile time checking of CSS and regex is in contrast to most programming languages where CSS selectors and regexes are passed in as strings and can result in runtime errors if malformed.

5.4 Semantic Analyzer

The semantic analyzer takes in an AST and produced a type-checked SAST which can easily be converted to code. It recursively traverses the abstract syntax tree, and for each AST node builds corresponding SAST nodes which augment underlying AST data with additional, particularly type information. While building the SAST, the semantic analyzer updates translation environment and consults the translation environment. This is particularly important for identifier resolution and type inference (see section 5.5).

Because HAWK uses type inference and has no explicit type annotation, every unique call to a function (where uniqueness is determined by function name and the types of arguments) has a different body and thus generates a different SAST function declaration.

5.5 The Translation Environment

The translation environment contains mutiple pieces of information about a program which are used for identifier resolution (variables as well as functions) and type inference.

Firstly, the translation environment contains a symbol table for every scope found by the semantic analyzer. This symbol table stores a type for each variable in a scope. In cases where a table of unresolvable type is assigned to a variable (which happens when we assign an empty table literal or nested empty table literal) we denote this with a special type called EmptyTable that is invisible to the user and only used during semantic analysis.

Additionally, the translation environment stores "update table type links" for each variable, which allow variables referring to the same underlying empty table to share information about unresolved table types and aid type inference. When the fortunes of two variables storing empty tables become linked (such as through an assignment like a=b), each adds a table type link to the other along with scope and relative nesting information. When a concrete table is finally assigned to a variable referencing an empty table (either directly or implicitly through table index assignment or function argument behavior) this information propagates through a graph of table links using depth first search and updates the types of all linked variables.

The translation environment contains mutable state. After the semantic analysis stage, the translation environment is complete and will no longer be changed. However, certain objects in

the SAST contain closures which refer to objects in the translation environment. These closures assume that the translation environment is complete and thus that all type information that can be known, is known. In the generation stage we call certain closures and therefore read from the filled out translation environment, but do not write to it.

5.6 The Code Generator

We directly generate Java code from our SAST. Many expressions are straightforward to translate, as HAWK types all have a corresponding Java type. Our parameterized table type corresponds to a generic class we define the runtime library (see Section 5.7). Care is taken to use boxed Java types rather than primitive types where it is required, particularly for generic java classes. Additionally, many HAWK constructs (such as nested table indexing, and first class CSS/regex patterns) have no direct analogue in Java. For these, we generate Java code which calls upon routines from our runtime library.

5.7 The Java Runtime Library

The Java runtime library contains implementations of built-in functions, routines to run CSS/regex pattern-action blocks, utilities for file reading, and a class for HAWK tables.

6. Test Plan

6.1 Source to Target

We use the same program used in 2.1 General Syntax

Source Program:

```
BEGIN{

    table = {1, 2, 3, 4, 5, 6};

    i = 0;

    while(i < length(table)){

        if(table[i] % 2 == 1){

                table[i] = table[i] + 1;

        }

    }

}

END{

    print(table);

}
```

Target Result:

```
import _hawk_lib.*;

public class Program {

    public static void main(String[] _args){

        _HAWKFileReader _fileReader = new _HAWKFileReader(_args);

        _HAWKRegexMatcher _regexMatcher = new
_HAWKRegexMatcher(_fileReader._getConcatFile());

        _HAWKCSSMatcher _cssMatcher = new
_HAWKCSSMatcher(_fileReader._getConcatFile());

        _HAWKTable<Integer> _table_ = (new _HAWKTable<Integer>
()).setIntIndexChained(0,1).setIntIndexChained(1,2).setIntIndexChained(2,3).setI
ntIndexChained(3,4).setIntIndexChained(4,5).setIntIndexChained(5,6);
```

```
    int _i_ = 0;
    while(_i_ < _length_(_table_)){
        if(_checkIf(_table_.getIntIndex(_i_)%2 == 1)){
        _table_.setIntIndex(_i_,_table_.getIntIndex(_i_) + 1);

    }

    }

    _print_(_table_);

}
private static void _print_(Object o){
  System.out.println(o);
}
private static String _charAt_(String s, Integer i){
  return ""+s.charAt(i);
}
private static Integer _stringEqual_(String s1, String s2){
  if(s1.equals(s2)){
    return 1;
  }else{
    return 0;
```

```
  }
}
private static Boolean _checkIf(Integer i){
  return (i != 0) ? true : false;
}
private static Boolean _checkIf(Double d){
  return (d != 0.0) ? true : false;
}
private static Boolean _checkIf(Boolean b){
  return b;
}
private static <T> int _length_(_HAWKTable<T> t){
  return t.getLength();
}

private static <T> int _exists_(T e){
  if(e == null){
    return 0;
  }
  return 1;
}

private static int _stringToInt_ (String s){
    return Integer.parseInt(s);
```

```
    }


    private static <T> _HAWKTable<String> _keys_(_HAWKTable<T> t){
      java.util.Set<String> set = t.getKeys();
      _HAWKTable<String> hawkTable = new _HAWKTable<String>();
      int i = t.getLength()-1;
      for (String s : set) {
        hawkTable.setIntIndex(i, s);
        i--;
      }
      return hawkTable;
    }


}
```

6.2 Test Suite

As we wrote the compiler, we also wrote tests for every feature we planned on including in the compiler. For each of these features, we wrote several small tests to test individual elements, such as simple operators, table literals, function calls, for loops, etc.

```
./testall.sh
-n testresult/test-cf-exists-1...
OK
-n testresult/test-cf-if-1...
OK
```

```
-n testresult/test-cf-if-2...
OK
-n testresult/test-cf-if-3...
OK
-n testresult/test-cf-keys-1...
OK
-n testresult/test-cf-length-1...
OK
-n testresult/test-cf-print-1...
OK
-n testresult/test-cf-while-1...
OK
-n testresult/test-cf-while-2...
OK
-n testresult/test-for-1...
OK
-n testresult/test-for-2...
OK
-n testresult/test-fun-1...
OK
-n testresult/test-fun-10...
OK
-n testresult/test-fun-11...
```

```
OK

-n testresult/test-fun-12...

OK

-n testresult/test-fun-13...

OK

-n testresult/test-fun-14...

OK

-n testresult/test-fun-15...

OK

-n testresult/test-fun-16...

OK

-n testresult/test-fun-2...

OK

-n testresult/test-fun-3...

OK

-n testresult/test-fun-4...

OK

-n testresult/test-fun-5...

OK

-n testresult/test-fun-6...

OK

-n testresult/test-fun-7...

OK

-n testresult/test-fun-8...
```

```
OK
-n testresult/test-fun-9...
OK
-n testresult/test-op-arith1...
OK
-n testresult/test-op-arith10...
OK
-n testresult/test-op-arith11...
OK
-n testresult/test-op-arith12...
OK
-n testresult/test-op-arith13...
OK
-n testresult/test-op-arith14...
OK
-n testresult/test-op-arith15...
OK
-n testresult/test-op-arith16...
OK
-n testresult/test-op-arith17...
OK
-n testresult/test-op-arith18...
OK
```

```
-n testresult/test-op-arith19...

OK

-n testresult/test-op-arith20...

OK

-n testresult/test-op-arith22...

OK

-n testresult/test-op-arith23...

OK

-n testresult/test-op-arith3...

OK

-n testresult/test-op-arith4...

OK

-n testresult/test-op-arith5...

OK

-n testresult/test-op-arith6...

OK

-n testresult/test-op-arith7...

OK

-n testresult/test-op-arith8...

OK

-n testresult/test-op-arith9...

OK

-n testresult/test-pattern-css1...

OK
```

```
-n testresult/test-pattern-css10...
OK
-n testresult/test-pattern-css11...
OK
-n testresult/test-pattern-css12...
OK
-n testresult/test-pattern-css13...
OK
-n testresult/test-pattern-css2...
OK
-n testresult/test-pattern-css3...
OK
-n testresult/test-pattern-css4...
OK
-n testresult/test-pattern-css5...
OK
-n testresult/test-pattern-css6...
OK
-n testresult/test-pattern-css7...
OK
-n testresult/test-pattern-css8...
OK
-n testresult/test-pattern-css9...
```

```
OK

-n testresult/test-pattern-regex1...

OK

-n testresult/test-pattern-regex10...

OK

-n testresult/test-pattern-regex11...

OK

-n testresult/test-pattern-regex2...

OK

-n testresult/test-pattern-regex3...

OK

-n testresult/test-pattern-regex4...

OK

-n testresult/test-pattern-regex5...

OK

-n testresult/test-pattern-regex6...

OK

-n testresult/test-pattern-regex7...

OK

-n testresult/test-pattern-regex8...

OK

-n testresult/test-pattern-regex9...

OK

-n testresult/test-tbl1...
```

```
OK

-n testresult/test-tbl10...

OK

-n testresult/test-tbl2...

OK

-n testresult/test-tbl3...

OK

-n testresult/test-tbl4...

OK

-n testresult/test-tbl5...

OK

-n testresult/test-tbl6...

OK

-n testresult/test-tbl7...

OK

-n testresult/test-tbl8...

OK

-n testresult/test-tbl9...

OK

successes = 82

failures = 0

total tests = 82
```

6.3 Test Automation

We had about 80 tests in our test suite, so we wrote a test script testall.sh that runs every test. It pipes the output of the program to a .i.out file, and then diffs it with a .out file that we created. If there are any differences, it will mark the test as a failure. The script will also print out the number of successes, failures, and total tests.

The tests are run with `make test`.

The testing script is included in the Appendix.

7. Lessons Learned

7.1 Graham Gobieski

- In order to motivate the group, we learned that it is best to get together for weekly five-hour hack sessions. We were also the most productive during these sessions because we could figure out together solutions any problems that arose.
- Know your LRM so that you do not have to keep referrring back to it in order to implement langauge features. Also this increases the rate at which demo programs can be written.
- Set realistic goals with the LRM. We did a great job of this, but it is still worth mentioning because of how helpful it turned out to be.
- Use branches when implementing a new feature and then merge the branch onto the master when all tests are passing. Do not commit broken code to the master.

7.2 George Yu

- Understand OCaml. Writing a compiler is already difficult; it's even more so without some understanding of the language it's being written in.
- Have weekly group meetings. Group hack sessions made sure group members could understand/catch up with any updates and also tackle and understand what to do next. Group meetings help to break down next steps and figure out how to tackle problems.
- Constant communication. Knowing when something's finished or when something works helped push development evenly through the process, so that most of the features we wanted to have in our language were eventually implemented.

7.3 Ethan Benjamin

- OCaml is practical even in a team full of functional programming novices, and the unique

benefits of functional programming become more and more apparent as the size of a codebase increases. Before starting my Master's I worked in industry for four years and have seen many of the problems that arise in large software projects. In my experience, fixing bugs consumes most of a developer's time, and many bugs in traditional imperative languages come from code that fails to account for a particular object taking a particular state. OCaml's default purity, discriminated unions and lack of nullable types, combined with pattern matching, greatly reduce the risk of encountering exceptional behavior that was unaccounted for. I found that the pure functions we wrote rarely ever had to be rewritten, and if they did, it was due to a genuine change in feature requirements rather than due to an unexpected bug. To a novice like me OCaml can feel frustratingly strict in that it makes you explicity account for every use case, but this turns out to be a form tough love that encourages you to get things right immediately and yields great benefits down the line.

- An effective leader has an outsize effect in improving team productivity. Our project manager did an outstanding job of identifying outstanding issues and assigning responsbilities. He portioned tasks in a way that maximized our ability to work independently and was always quick to address problems which blocked progress. Because he took a birds eye view of the project he was also effective in helping team mates working on different components effectively communicate with each other.

7.4 Justin Chang

- Start early
- Set deadlines by understanding work involved, when assigning/discussing next steps, actually talk through and figure out what has to be done
- Work together! More efficient to bounce ideas off of each other, work in bursts as a group if possible
- Don't push untested code and don't push without being aware of what other work is being done

7.5 Jon Adelson

- While a seemingly obvious statement, knowing OCaml well helps a lot when writing a compiler in OCaml. Familiarity with OCaml List functions makes code much clearer and likely error free. Additionally, getting used to making mistakes in OCaml is an important part of the development process. Initially, I would get frustrated because I believed that OCaml was incorrectly complaining about a seemingly valid program, only to find that I had in fact passed in the incorrect type, forgotten an argument to a function, or made one of many other possible mistakes. Learning to read OCaml error messages and

understanding where the real error exists is an important skill.

- It is important to know what other people have already completed because you do not want two people writing the same pieces of code and you can easily leverage what others have written to accomplish new tasks. For example, when incorporating functions into the language, Graham had already written most of the code to semantically check statements, so it was reasonably straightforward to check all of the statements within the body of a function declaration.
- Although it is more interesting to think about the functionality and syntax of your language, you do need to take time to consider what is legal in your target language. For example, there were a couple scenarios we had to guard against when returning values from functions. Java does not like if a non-void function is not guaranteed to return a value, such as if the return statement is within an if statement that does not also have an else clause. Additionally, Java does not like when you have unreachable code, such as when there are additional statements after a return statement that is guaranteed to execute.

8. Appendix

8.1 Scanner

```
(*Setup type to check if we are scanning a regex pattern block*)

{

    open Parser

    open Util

    type is_pat = REGEX | CSS | NO

    let state_ref = ref NO

}



(*Some standard character classes*)

let digits = ['0' - '9']+

let signed_int = ['+' '-']? digits
```

```
let decimal = ['+' '-']? (digits '.' ['0'-'9']* | '.' digits) (['e' 'E']
signed_int)?

let id = ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*

let css_id = ['a'-'z' 'A'-'Z'] ['a'-'z' 'A'-'Z' '0'-'9' '_' '-' ]*



(*Rules for converting buf to tokens*)

rule token pat = parse

    [' ' '\t' '\r' '\n'] {token pat lexbuf}

    | "/*"     { comment pat lexbuf }

    | '[' {LBRACK} | ']' {RBRACK}

    | '{' {LBRACE} | '}' {RBRACE}

    | '(' {LPAREN} | ')' {RPAREN}

    | ';' {SEMI} | ':' {COLON}

    | ',' {COMMA}

    | "fun" {FUN}

    | "in" {IN}

    | "BEGIN" {BEGIN} | "END" {END}

    | "else" {ELSE} | "if" {IF}

    | "while" {WHILE} | "for" {FOR}

    | "this" {THIS}

    | "return" {RETURN}

    | '<' {LT} | '>' {GT} | "==" {EQ} | "!=" {NEQ} | ">=" {GEQ} | "<=" {LEQ} |
"&&" {AND} | "||" {OR}

    | '+' {PLUS} | '-' {MINUS} | '*' {TIMES} | '/' {DIVIDES} | '=' {ASSIGN} |
'%' {MOD}
```

```
    | "[/" {pat := REGEX ; LBRACK_FSLASH}

    | "[@" {pat:= CSS; LBRACK_AMP}

    | "*=" {TIMES_EQ} | "^=" {XOR_EQ} | "$=" {DOLLAR_EQ} | "~=" {TILDE_EQ}

    | id as lxm {ID(wrap_id lxm) }

    | digits as lxm {INT(int_of_string lxm)}

    | decimal as lxm {DOUBLE(float_of_string lxm)}

    | '"' [^ '"']* '"' as lxm {STRING(lxm)}

    | eof {EOF}


(*Switch to this rule when a comment is encountered*)

    and comment pat = parse

      "*/" { token pat lexbuf }

    | _    { comment pat lexbuf }


(*Switch to this rule when a regex pattern block is encountered. This is
required

because regex expressions are more general than ID and are not strings.*)

    and regex_scan pat = parse

        "/]" {pat := NO ; FSLASH_RBRACK}

        | ['\\']['"' '.' '?' '|' '^' ']' '[' '(' ')' '-' '\\' '$' '*' 'n' 't'
'r'] as lxm{REGEX_STRING(lxm)}

        | '.' {PERIOD} | '?' {QUEST} | '^' {CARROT} | '|' {VERT} | '-' {MINUS} |
'*' {TIMES}

        | '[' {LBRACK} | ']' {RBRACK}

        | '(' {LPAREN} | ')' {RPAREN}
```

```
        | [^'"' '/' '.' '?' '|' '^' ']' '[' '(' ')' '-' '\\' '$' '*'] as lxm
{REGEX_STRING((Char.escaped lxm))}



and css_scan pat = parse

    [' ' '\t' '\r' '\n'] {css_scan pat lexbuf}

    | "@]" {pat:= NO; AMP_RBRACK}

    | '(' {LPAREN} | ')' {RPAREN}

    | id as lxm { ID(lxm)}

    | css_id as lxm { CSSID(lxm) }

    | '[' {LBRACK} | ']' {RBRACK}

    | '(' {LPAREN} | ')' {RPAREN}

    | '<' {LT} | '>' {GT}

    | '~' {TILDE}

    | '#' {HASH}

    | '+' {PLUS} | '*' {TIMES} | '=' {ASSIGN}

    | "*=" {TIMES_EQ} | "^=" {XOR_EQ} | "$=" {DOLLAR_EQ} | "~=" {TILDE_EQ}

    | '"' [^ '"']+ '"' as lxm {STRING(lxm)}

    | '.' {PERIOD}

    | ',' {COMMA}



(*The function to get the next token; checks to see if it is scanning a regex
pattern block*)

{

    let next_token lexbuf = match !state_ref with

        | NO -> token state_ref lexbuf
```

```
        | REGEX -> regex_scan state_ref lexbuf

        | CSS -> css_scan state_ref lexbuf

}
```

8.2 Parser

```
/*NOTE: much of the parsing code for statements/expressions is directly adapted
from MicroC*/



%{ open Ast

   open Util

%}



%token LPAREN RPAREN LBRACK RBRACK LBRACE RBRACE

%token PLUS MINUS TIMES DIVIDES MOD

%token LT GT LEQ GEQ EQ NEQ AND OR

%token PERIOD ASSIGN HASH TILDE COMMA COLON PERIOD QUEST CARROT VERT

%token FUN

%token SEMI

%token LBRACK_AMP

%token AMP_RBRACK

%token LBRACK_FSLASH

%token FSLASH_RBRACK

%token EOF
```

```
%token <string> STRING

%token <string> REGEX_STRING

%token <int> INT

%token <float> DOUBLE

%token <string> ID

%token <string> CSSID


/*Program structure*/

%token BEGIN END

/*other keywords*/

%token ELSE IF RETURN THIS WHILE FOR IN


/* the attribute selectors */

%token TIMES_EQ XOR_EQ DOLLAR_EQ TILDE_EQ


/*Precedence and associativity*/

%nonassoc NOELSE

%nonassoc ELSE

%right ASSIGN

%left AND

%left OR

%left EQ NEQ

%left LT GT LEQ GEQ

%left PLUS MINUS
```

```
%left TIMES DIVIDES MOD VERT QUEST

%nonassoc UMINUS


%start program

%type <Ast.program> program


%%


/*Program structure */

program:

    begin_stmt pattern_action_list end_stmt EOF { {begin_stmt = $1;

                                                   pattern_actions = $2;

                                                   end_stmt = $3} }


begin_stmt:

    BEGIN LBRACE stmt_list RBRACE {Block($3)}

end_stmt:

    END LBRACE stmt_list RBRACE {Block($3)}


pattern_action_list:

    /* */ {[]}

    | pattern_action pattern_action_list {$1 :: $2}
```

```
pattern_action:
    pattern LBRACE stmt_list RBRACE { ($1,Block($3)) }

/*End of program structure*/


/*Statements and expressions*/
stmt_list:
    /* */ {[]}
    | stmt stmt_list {$1 :: $2}

stmt:
    expr_no_brace SEMI {Expr($1)}
    | RETURN expr SEMI {Return($2)}
    | LBRACE stmt_list RBRACE {Block($2)}
    | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3,$5, Block([]))}
    | IF LPAREN expr RPAREN stmt ELSE stmt { If($3,$5,$7) }
    | WHILE LPAREN expr RPAREN stmt { While($3,$5) }
    | FOR LPAREN ID IN ID RPAREN stmt { For($3,$5,$7) }
    | FOR LPAREN ID IN THIS RPAREN stmt { ForThis($3,$7) }
    | FUN func_decl {Func($2)}
    | SEMI {Empty}

expr:
```

```
    table_literal {TableLiteral($1)}

    /*below fixes S/R error. Basically, do all arithmetic before this reduction
*/

    | expr_no_brace {$1} %prec ASSIGN


expr_no_brace:

    ID {Id($1)}

    | literal {Literal($1)}

    | expr_no_brace TIMES expr_no_brace {Binop($1,Times,$3)}

    | expr_no_brace DIVIDES expr_no_brace {Binop($1,Divides,$3)}

    | expr_no_brace MOD expr_no_brace {Binop($1,Mod,$3)}

    | expr_no_brace MINUS expr_no_brace {Binop($1,Minus,$3)}

    | expr_no_brace PLUS expr_no_brace {Binop($1,Plus,$3)}

    | expr_no_brace EQ expr_no_brace {Binop($1,Equal,$3)}

    | expr_no_brace LT expr_no_brace {Binop($1,Less,$3)}

    | expr_no_brace GT expr_no_brace {Binop($1,Greater,$3)}

    | expr_no_brace LEQ expr_no_brace {Binop($1,LessEqual,$3)}

    | expr_no_brace GEQ expr_no_brace {Binop($1,GreaterEqual,$3)}

    | expr_no_brace NEQ expr_no_brace {Binop($1,NotEqual,$3)}

    | expr_no_brace AND expr_no_brace {Binop($1,BAnd,$3)}

    | expr_no_brace OR expr_no_brace {Binop($1,BOr,$3)}

    | ID ASSIGN expr {Assign($1,$3)}

    | ID LPAREN expr_list RPAREN {Call($1,$3)}

    | ID LPAREN  RPAREN {Call($1,[])}
```

```
    | ID bracket_expr_list {TableAccess($1,$2)}
    | THIS LBRACK expr RBRACK {ThisAccess($3)}
    | ID bracket_expr_list ASSIGN expr {TableAssign($1,$2,$4)}
    | LPAREN expr RPAREN {$2}
    | MINUS expr_no_brace %prec UMINUS {Uminus($2)}

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

bracket_expr_list:
    | LBRACK expr RBRACK {[$2]}
    | LBRACK expr RBRACK bracket_expr_list { $2 :: $4}

literal:
    INT {IntLiteral($1)}
    |STRING {StringLiteral($1)}
    |DOUBLE {DoubleLiteral($1)}
  |THIS {This}

table_literal:
    array_literal {$1}
    |keyvalue_literal {KeyValueLiteral($1)}
```

```
array_literal:
    LBRACE RBRACE { EmptyTable }
    |LBRACE expr_list RBRACE {ArrayLiteral($2)}

keyvalue_literal:
    LBRACE keyvalue_list RBRACE {$2}

keyvalue_list:
    keyvalue { [$1] }
    |keyvalue_list COMMA keyvalue { $3 :: $1}

keyvalue:
    key COLON expr  { ($1,$3) }

key:
    INT {IntKey($1)}
    |STRING {StringKey($1)}

func_decl:
    ID LPAREN params_list RPAREN LBRACE stmt_list RBRACE { {fname=$1;
                                                            params=$3;
                                                            body=$6;
                                                          }}
```

```
params_list:
    /* */ { [] }
    | ID { [$1] }
    | ID COMMA params_list {$1::$3}

pattern:
    LBRACK_AMP css_selector AMP_RBRACK {CssPattern($2)}
    | LBRACK_FSLASH regex_sequence FSLASH_RBRACK {RegexPattern($2)}

/*End of statements and expressions*/

/*Start of Regex*/

regex:
    REGEX_STRING {RegexString($1)}
    | PERIOD {RegexAnyChar}
    | LPAREN regex_sequence RPAREN {RegexNested($2)}
    | LBRACK regex_set_sequence RBRACK {RegexSet($2)}
    | regex QUEST {RegexUnOp($1,Optional)}
    | regex PLUS {RegexUnOp($1,KleenePlus)}
    | regex TIMES {RegexUnOp($1,KleeneTimes)}
    | regex VERT regex {RegexBinOp($1,Or,$3)}
```

```
regex_sequence:

    regex {[$1]}

    | regex regex_sequence {$1 :: $2}


regex_set:

    REGEX_STRING {RegexStringSet($1)}

    | REGEX_STRING MINUS REGEX_STRING{RegexRangeSet($1,$3)}

    | uARROT regex_set {RegexComplementSet($2)}

    | LBRACK regex_set_sequence RBRACK {RegexNestedSet($2)}


regex_set_sequence:

    regex_set {[$1]}

    | regex_set regex_set_sequence {$1 :: $2}


/*End of Regex stuff*/


/*Start of CSS Selector stuff*/

css_selector:

    simple_selector_seq {SingleSelector($1)}

    | css_selector PLUS simple_selector_seq
{ChainedSelectors($1,DirectSibling,$3)}

    | css_selector GT simple_selector_seq {ChainedSelectors($1,Descendent,$3)}

    | css_selector TILDE simple_selector_seq
{ChainedSelectors($1,AnySibling,$3)}
```

```
    | css_selector typed_simple_selector_seq
{ChainedSelectors($1,DirectChild,$2)}



simple_selector_seq:

    typed_simple_selector_seq {$1}

    | property_selector_list {(NoType,$1)}



typed_simple_selector_seq:

    type_selector {($1,[])}

    | type_selector property_selector_list {($1,$2)}



type_selector:

    TIMES {Universal}

    |ID {Elt($1)}



property_selector_list:

    property_selector { [$1]}

    | property_selector property_selector_list { $1::$2}



css_id:

    CSSID {$1}

    | ID {$1}



property_selector:
```

```
    PERIOD css_id {ClassMatch($2)}
    | HASH css_id {IdMatch($2)}
    | LBRACK css_id RBRACK {AttributeExists($2)}
    | LBRACK css_id ASSIGN STRING RBRACK {AttributeEquals($2,$4)}
    | LBRACK css_id TIMES_EQ STRING RBRACK {AttributeContains($2,$4)}
    | LBRACK css_id XOR_EQ STRING RBRACK {AttributeBeginsWith($2,$4)}
    | LBRACK css_id DOLLAR_EQ STRING RBRACK {AttributeEndsWith($2,$4)}
    | LBRACK css_id TILDE_EQ STRING RBRACK {AttributeWhitespaceContains($2,$4)}
/*end of CSS selector stuff */
```

8.3 AST

```
(* NOTE: much of the AST code for statements/expressions is closely adapted from
MicroC *)


(* CSS patterns *)
type css_combinator = DirectChild | Descendent | DirectSibling | AnySibling


type property_selector =
    ClassMatch of string
    | IdMatch of string
    | AttributeExists of string
    | AttributeEquals of string * string
    | AttributeContains of string * string
```

```
    | AttributeBeginsWith of string * string

    | AttributeEndsWith of string * string

    | AttributeWhitespaceContains of string * string


type type_selector =

    Elt of string

    | Universal

    | NoType


type simple_selector_sequence =

    type_selector * (property_selector list)


type css_selector =

    SingleSelector of simple_selector_sequence

    | ChainedSelectors of css_selector * css_combinator *
simple_selector_sequence


(* Regex patterns *)

type regex_op = Or | Optional | KleenePlus | KleeneTimes


type regex_set =

    RegexStringSet of string

    | RegexRangeSet of string * string

    | RegexComplementSet of regex_set
```

```
    | RegexNestedSet of regex_set_sequence
and regex_set_sequence = regex_set list


type regex =
        RegexString of string
        | RegexAnyChar
        | RegexNested of regex_sequence
        | RegexSet of regex_set_sequence
        | RegexUnOp of regex * regex_op
        | RegexBinOp of regex * regex_op * regex
and regex_sequence = regex list



type pattern =
    CssPattern of css_selector
    | RegexPattern of regex_sequence

(* Arithmetic Expressions *)
type op = Plus | Minus | Divides | Times | Equal
            | Less | Greater | LessEqual | GreaterEqual
            | NotEqual | Mod | BAnd | BOr

(* Table Literals *)
type key_literal =
```

```
    IntKey of int
    |StringKey of string


type literal =
    IntLiteral of int
    |StringLiteral of string
    |DoubleLiteral of float
    |This
and table_literal =
    EmptyTable
    |ArrayLiteral of expr list
    |KeyValueLiteral of (key_literal * expr) list
and expr =
    Id of string
    |Literal of literal
    |TableLiteral of table_literal
    |Assign of string * expr
    |Binop of expr * op * expr
    |Uminus of expr
    |Call of string * expr list
    |TableAccess of string * (expr list)(* e.g t[1] or (t[2][3])[4]  *)
    |ThisAccess of expr
    |TableAssign of string * (expr list) * expr
```

```
(*Program Structure and Action Syntax*)

type stmt =

    Block of stmt list

    | Expr of expr

    | Func of func_decl

    | Return of expr

    | If of expr * stmt * stmt

    | While of expr * stmt

    | For of string * string * stmt

    | ForThis of string * stmt

    | Empty

and

func_decl = {

    fname : string;

    params : string list;

    body : stmt list;

}


type pattern_action = pattern * stmt


type program = {

    begin_stmt : stmt;

    pattern_actions : pattern_action list;
```

```
    end_stmt : stmt;
}
```

8.4 Semantic Checker

```
open Sast

let is_table = function
    Table(_) | EmptyTable -> true
    | _ -> false

let rec type_to_str = function
  Int -> "int"
  | Double -> "double"
  | Table(value_type) as t -> "Table(" ^ (type_to_str value_type) ^ ")"
  | String -> "String"
    | Void -> "void"
    | EmptyTable -> "ET"
    | UnknownReturn -> "UR"

let get_binop_type t1 op t2 =
  match op with
  Ast.Plus -> (
```

```
    match t1, t2 with

    _, UnknownReturn | UnknownReturn,_ -> UnknownReturn

    |x, y when x = y && not (is_table x) ->  x

    | x, y when x = String || y = String ->  String

    | Int, Double -> Double

    | Double, Int -> Double

    | _ , _ -> raise (Failure("binary operation type mismatch"))

    )

  | _ -> (

    match t1, t2 with

    _, UnknownReturn | UnknownReturn,_ -> UnknownReturn

    |x, y when x = y && not (is_table x) && x != String ->  x

    | Int, Double ->  Double

    | Double, Int ->  Double

    | _ , _ -> raise (Failure("binary operation type mismatch or operation does
not support these types"))

    )



type b_arg_types = BAny | BTable | BString | BInt



(*Built-in functions and their types*)

let built_in = [("_print_", [BAny], Int); ("_exists_", [BAny], Int);

                ("_length_", [BAny], Int); ("_keys_", [BTable], Table(String));

                ("_children_", [BTable], String); ("_inner_html_", [BTable],
```

```
String);

                ("_charAt_", [BString; BInt], String); ("_stringEqual_",
[BString; BString], Int);

                ("_stringToInt_", [BString],Int) ]



let rec find_var_and_scope (scope : symbol_table) name = try

  (List.find (fun (s, _) -> s = name) scope.variables),scope with Not_found ->

  match scope.parent with

    Some(parent) -> find_var_and_scope parent name

    | _ -> raise Not_found



let rec is_empty_table_container = function

    EmptyTable -> true

    | Table(t) -> (is_empty_table_container t)

    | _ -> false



let rec find (scope : symbol_table) name =

    fst (find_var_and_scope scope name )



let assert_not_void typ err =

    if typ = Void then

        raise (Failure err)



let get_sig_return_type global_env func_sig =
```

```
    List.assoc func_sig global_env.func_signatures



(*

let get_existing_func_decl global_env func_signature =

    let matches_signature func_decl =

        let param_type_promises = List.map snd func_decl.params in

        let param_types = List.map (fun promise -> promise ())
param_type_promises in

        func_decl.fname = (fst func_signature) && param_types = (snd
func_signature)

    in

    try

        let fdecl = List.find matches_signature global_env.funcs in

        let return_type = (fdecl.return_type_promise ()) in

        Some (fdecl,return_type)

    with Not_found -> None

*)



let add_initial_func_signature global_env func_signature =

    ignore (global_env.func_signatures<- (func_signature,UnknownReturn)::
(global_env.func_signatures))



let get_existing_func_sig global_env func_signature =

    try

        let entry = List.find (fun entry -> (fst entry) = func_signature)
global_env.func_signatures in
```

```
        Some (func_signature,(snd entry))
    with Not_found -> None


(*
if a variable is of type t1, can it be assigned to a variable of type t2?
obviously if:


a=3
then you can do
a=10 (both are Int)


but also if:
a = {}
then
a = {{3}} is allowable too because EmptyTable can possibly match any table
(and vice versa)
*)
let rec can_assign t1 t2 =
    match t1,t2 with
        t1,t2 when t1=t2 -> true
        | EmptyTable, Table(t) -> true
        | Table(t), EmptyTable -> true
        | Table(s), Table(t) -> (can_assign s t)
```

```
        | _ -> false




(*Find a built in function by name  *)

let rec find_built_in name = try

  List.find (fun (s, t, r) -> s = name) built_in with Not_found -> raise
Not_found



(*Update a variable type of a variable within a given symbol table,

going up parent links until matching variable is found

This function has side effects*)

let rec update_variable_type sym_t var_id new_type =

    try

        let _ = (find sym_t var_id) in (* this will raise exception if variable
doesn't exist *)

        let new_variable_list =

            List.map (fun (v_id,typ) -> if v_id=var_id then (v_id,new_type) else
(v_id,typ)) sym_t.variables

        in

        ignore (sym_t.variables <- new_variable_list)

    with Not_found ->

        match sym_t.parent with

            None -> raise (Failure "Couldn't find variable to update")

            | Some(parent) -> update_variable_type parent var_id new_type
```

```
(* Eliminate links for updating table types between two variables

this function has side effects

 *)

let remove_update_table_link table_id sym_tab link_id link_scope =

    (* match on value equality for link id and reference equality for link scope
*)

    let keep_entry (t_id,update_link) =

        not (t_id=table_id && update_link.id=link_id && update_link.assign_scope
== link_scope)

    in

    sym_tab.update_table_links<- List.filter keep_entry
sym_tab.update_table_links



let remove_mutual_update_table_links table_id sym_tab link_id link_scope =

    remove_update_table_link table_id sym_tab link_id link_scope;

    remove_update_table_link link_id link_scope table_id sym_tab



let add_update_table_link table_id sym_tab link_id link_scope nesting =

    let new_update_link = {id=link_id;assign_scope=link_scope;nesting=nesting}
in

    sym_tab.update_table_links <-
(table_id,new_update_link)::sym_tab.update_table_links



(* imagine:

t = { {} }
```

```
s = t[0]



s is linked to t with nesting level 1

t is linked to s with nesting level -1



if you do:

t[0][0] = 3, then t goes from Table(EmptyTable) to Table(Table(Int)), and s
should go from EmptyTable to Table(Int)



if you do:

s[0] = 3, the end result should be the same

*)

let add_mutual_update_table_link table_id sym_tab link_id link_scope nesting =

    remove_mutual_update_table_links table_id sym_tab link_id link_scope;

    add_update_table_link table_id sym_tab link_id link_scope nesting;

    add_update_table_link link_id link_scope table_id sym_tab (-nesting)




(*nest or unnest a type with additional tables

e.g.

apply_nesting Int 1 = Table(Int)

apply_nesting Table(Table(Int)) -1 = Table(Int)

*)

let rec apply_nesting = function
```

```
    t, 0 -> t
    | t, n when n>0 -> Table (apply_nesting (t,n-1))
    | Table(t),n when n<0 -> (apply_nesting (t,n+1))
    | _ -> raise (Failure "Attempting to unnest non-table.")



let is_identifier_expr = function
    | Id(_) -> true
    | TableAccess(_) -> true
    | Assign(_) -> true
    | TableAssign(_) -> true
    | _ -> false


(* return the identifier and nesting level for
identifier based expressions


None for non-identifier based expressions
 *)
let get_identifier_expr_info = function
    Id(id) -> Some (id,0)
    | TableAccess(id,ind_list) -> Some (id,(List.length ind_list))
    | Assign(id,_) -> Some (id,0)
    | TableAssign(id,ind_list,_) -> Some (id, (List.length ind_list))
    | _ -> None
```

```
let get_var_type_promise scope id =

    let promise () =

        let (_,t) = (find scope id) in

        t

    in

    promise



let get_id_based_expr_promise id scope nesting expr =

    let promise () =

        let (_,t) = (find scope id) in

        expr, (apply_nesting (t,(-nesting)))

    in

    promise

(*

imagine:

t = {3:{},10:{}}

if we later find out the type of t to be Table(Table(Int)) or something even
more nested, then we can now determine the

types of the nested empty tables in this table literal

*)

let rec retype_empty_table_literal table_literal new_table_type =

    match table_literal with

        | [] -> []
```

```
        | (key, (old_e, old_t))::tail ->

            let inner_t = apply_nesting (new_table_type,-1) in

            let new_e = (match old_e with

                |TableLiteral(tl) -> TableLiteral (retype_empty_table_literal tl
inner_t)

                | _ -> old_e)

            in

            (key, (new_e,inner_t))::(retype_empty_table_literal tail
new_table_type)



let get_table_literal_promise assigner tl =

    let promise () =

        let (_,new_t) = (find assigner.assign_scope assigner.id ) in

        let nested_t = apply_nesting (new_t,(-assigner.nesting)) in

        let new_tl = retype_empty_table_literal tl nested_t in

        TableLiteral(new_tl),nested_t

    in promise



(*

Return a promise which will return

a SAST expression given our current knowledge of symbol tables

Once the semantics stage has completed this promise will give us our best

possible understanding of an expression
```

```
These promises are used anytime an l-value or potential l-value is involved.
This includes:



1. assignment

2. table assignment

3. return statements



Uncertainty arises from:



1. Empty table literals (EmptyTable)

2. Recursive calls  (UnknownReturn)



But can always be resolved when syntactic analysis is over

*)

let rec get_expression_promise assigner global_env assignee_e assignee_type
assignee_scope =

    let is_et = (is_empty_table_container assignee_type) in

    let id_info = get_identifier_expr_info assignee_e in

    let noop_promise = (fun () -> assignee_e,assignee_type) in

    let checked_promise promise =

        (fun () ->

            match assigner with

                None -> (promise ())

                | Some(assgn) ->

                    let (_,assgn_t) = (find assgn.assign_scope assgn.id) in
```

```
                    let assgn_t = apply_nesting (assgn_t,-assgn.nesting) in

                    let (_,new_t) as result = (promise ()) in

                    if assgn_t<>new_t then

                        raise

                        (Failure ("Trying to assign variable " ^ assgn.id ^

                        " of type " ^ (type_to_str assgn_t) ^ " with
incompatible type " ^ (type_to_str new_t)))

                    else

                        result

        )

    in

    let promise = (match assignee_e with

        Binop(e1,op,e2) ->

            let (e1_e,e1_type) = e1 in

            let (e2_e,e2_type) = e2 in

            let e1_promise = get_expression_promise None global_env e1_e e1_type
assignee_scope in

            let e2_promise = get_expression_promise None global_env e2_e e2_type
assignee_scope in

            ( fun () ->

                let (e1_e,e1_type) as e1 = (e1_promise ()) in

                let (e2_e,e2_type) as e2 = (e2_promise ()) in

                Binop(e1,op,e2), (get_binop_type e1_type op e2_type)

            )

        | Uminus(e1) ->
```

```
            let (e1_e,e1_type) = e1 in

            let e1_promise = get_expression_promise None global_env e1_e e1_type
assignee_scope in

            ( fun () ->

                let (e1_e,e1_type) as e1 = (e1_promise ()) in

                Uminus(e1), e1_type

            )



        |CallStub(signature,_) ->

            (fun () ->

                let return_t = get_sig_return_type global_env signature in

                assignee_e,return_t )

        | _ when (not is_et) ->

            (* If we're not dealing with an empty table, the expression and type
should not change *)

            noop_promise

        | _ when id_info <> None ->

            (match id_info with

                (*TODO: use assignee_scope instead *)

                Some(id,nesting) -> get_id_based_expr_promise id assignee_scope
nesting assignee_e

                | None -> raise (Failure "We shouldn't be here")

            )

        | TableLiteral(tl) ->

            (
```

```
            match assigner with

                (* At this point we're out of luck, but since its a truly empty
table with no references

                to non-empties, it shouldn't matter *)

                None -> noop_promise

                | Some (assigner) -> get_table_literal_promise assigner tl

            )

        | Call(fdecl,_) -> (fun () -> assignee_e, (fdecl.return_type_promise ())
)

        | _  -> raise (Failure "This type of expression should not yield an
empty table.")

    )

    in checked_promise promise

(*  Consider a statement like

a = {}

this should be deferred... we don't know how to construct 'a' until we know it's
type



however, once we have:

"a[0] = 3", or "a = {5}" or "b=a; b[3] = 3;"



We can make fully informed assignment because we know the type of a



this promise will return the correct type of expressions of this sort,

assuming the symbol table has been filled out properly
```

```
*)

let get_assignment_expression_promise assigner global_env assignee_e
assignee_type =

    get_expression_promise (Some assigner) global_env assignee_e assignee_type
assigner.assign_scope




(* Update the type of a table variable within a given symbol scope

Need to ensure that table update links are respected *)

let rec update_table_type sym_tab table_id new_type =

    (* update all table links in depth first search style *)

    let rec update_linked_table_types table_id sym_tab new_type visited =

        (*First do the neccesary mutation and update our variable type *)

        update_variable_type sym_tab table_id new_type;

        let already_visited t_id sym_tab visited =

            List.exists (fun (t,s) -> t=t_id && s==sym_tab) visited

        in

        let visited = (table_id,sym_tab)::visited in

        let neighbors = List.map snd (List.filter (fun (t_id,_) ->
table_id=t_id) sym_tab.update_table_links) in

        let unvisited_neighbors = List.filter (fun n -> not (already_visited
n.id n.assign_scope visited)) neighbors in

        let new_neighbor_types = List.map (fun n-> apply_nesting
(new_type,n.nesting)) unvisited_neighbors in

        let folder = (fun visited (neighb,neighb_t) -> update_linked_table_types
neighb.id neighb.assign_scope neighb_t visited) in

        List.fold_left folder visited (List.combine unvisited_neighbors
```

```
new_neighbor_types)


    in

    let (_, table_sym_tab) = find_var_and_scope sym_tab table_id in

    ignore (update_linked_table_types table_id table_sym_tab new_type [])



(*Find all return types of a statement

if a block, recursively search through sub-statements *)

let rec find_all_return_types = function

    Return(promise) -> [(snd (promise () ))]

    | Block(sl,_) -> List.concat (List.map find_all_return_types sl)

    | If(_,s1,s2) ->

        (* For if *)

        (find_all_return_types s1) @ (find_all_return_types s2)

    | While(_, stmt) -> (find_all_return_types stmt)

    | For(_,_,stmt) -> (find_all_return_types stmt)

    | _ -> []


(*Find the return type of a statement

if a block, recursively search through sub-statements and

mache sure return types are consistent*)

let find_return_type func_body =

    let all_return_types = find_all_return_types func_body in
```

```
    match all_return_types with

        [] -> Void

        | type_list ->

            if (Util.all_the_same type_list) then

                (List.hd type_list)

            else

                raise (Failure "Inconsistent return types in user defined
function.")




(*Find all return types of a statement

if a block, recursively search through sub-statements *)

let rec is_guaranteed_if_return valid = function

    If(_,Return(_),Return(_)) -> true

    | If(_,Return(_),Block(sl,_))  -> is_guaranteed_block_return valid sl

    | If(_,Block (sl,_), Return(_)) -> is_guaranteed_block_return valid sl

    | If(_,Return(_),If(a,b,c)) -> is_guaranteed_if_return valid (If(a,b,c))

    | If(_,If(a,b,c) ,Return(_)) -> is_guaranteed_if_return valid (If(a,b,c))

    | If(_,If (a,b,c),If (d,e,f)) -> is_guaranteed_if_return valid (If(a,b,c))
&& is_guaranteed_if_return valid (If(d,e,f))

    | If(_,Block (sl,_),Block (sl',_)) -> is_guaranteed_block_return valid sl &&
is_guaranteed_block_return valid sl'

    | If(_,If(a,b,c),Block (sl,_)) -> is_guaranteed_if_return valid (If(a,b,c))
&& is_guaranteed_block_return valid sl

    | If(_,Block (sl,_),If(a,b,c)) -> is_guaranteed_if_return valid (If(a,b,c))
&& is_guaranteed_block_return valid sl
```

```
    | If(_,_,_) -> false
and is_guaranteed_block_return valid = function
    [] -> valid
    | hd::tl ->
        match hd with
            Return(_) -> (match tl with
                        [] -> true
                        | _ -> raise (Failure "Unreachable code"))
            | If (a,b,c) ->
                let valid_if = is_guaranteed_if_return valid (If(a,b,c)) in
                if valid_if then (match tl with
                                    [] -> true
                                  | _ -> raise (Failure "Uncreachable code"))
                else is_guaranteed_block_return valid tl
            | Block(sl, _) ->
                let valid_block = (is_guaranteed_block_return valid sl) in
                if valid_block then (match tl with
                                      [] -> true
                                     | _ -> raise (Failure "Uncreachable code"))
                else is_guaranteed_block_return valid tl
            | _ -> is_guaranteed_block_return valid tl
```

```
(*

Just as with assignment, we may not know the return type of a function in
advance due to empty tables.

Assuming scope is available, this function will give you the proper return type



force_concrete = do we tolerate UnknownReturn types for this promise?

*)

let get_return_type_promise_checked func_body env force_concrete =

    let stmt_list = match func_body with Block(sl, _) -> sl in

    let all_return_type_promises = !(env.returns) in

    let get_return_type () =

        let all_return_types = List.map (fun f -> f () )
all_return_type_promises in

        match all_return_types with

            [] -> Void

            | type_list ->

                let search_list = if force_concrete then

                                    type_list

                                else

                                    List.filter (fun typ -> typ<>UnknownReturn )
type_list

                in

                if (Util.all_the_same search_list) then

                    if (is_guaranteed_block_return false stmt_list) then

                        (List.hd search_list)
```

```
                    else raise (Failure "No valid return statements")

                else

                    raise (Failure "Inconsistent return types in user defined
function.")

    in get_return_type


let get_return_type_promise func_body env =

    get_return_type_promise_checked func_body env true


let get_return_type func_body env =

    let promise = get_return_type_promise_checked func_body env false in

    (promise ())




let valid_table_index_type = function

    Int | String -> true

    | _ -> false


(*Get the value type of a table type going n_indices levels of nesting

e.g. if table_t = Table(int) and n_indices=1 then return Some int

if table_t = Table(Table(Table(string))) and n_indices=2, return Some
Table(string)

if table_t is not a table, return none *)

let rec get_table_access_type table_t n_indices =
```

```
    match table_t,n_indices with
        Table(val_type), 1 -> Some val_type
        | Table(val_type), n ->
            (match (get_table_access_type val_type (n-1)) with
                None -> None
                | x -> x)
        | _ -> None



(* Used to tie the types of empty table variables for type inference
when performing assignment or passing arguments into a function
var_id = variable name
var_nesting = nesting of var_id variable relative to assignee.
    For instance, if we have t = s[1][4] then nesting is 2
sym_t = symbol table that var_id resides in
asignee = expression being assigned to var_id
lookup_scope = scope to search for names in assignee
    When doing type inference through function calls, this will not be the same
as sym_t,
    whose bodies do not have access to names used in passed in parameters
*)
let create_linkage_if_applicable var_id var_nesting sym_t assignee lookup_scope
=
```

```
    let other_info = match assignee with

        Id(other_id) ->

            let ((_,other_type), other_scope) = find_var_and_scope lookup_scope
other_id in

            Some (other_id,other_type,other_scope, -var_nesting)

        | TableAccess(other_id, indices) ->

            let ((_,outer_type), other_scope) = find_var_and_scope lookup_scope
other_id in

            let other_nesting = List.length indices in

            let other_type = (apply_nesting (outer_type,-(other_nesting))) in

            Some (other_id,other_type,other_scope,other_nesting-var_nesting)

        | _ -> None

    in match other_info with

        | Some(other_id,other_type,other_scope,other_nesting) when
(is_empty_table_container other_type) ->

            add_mutual_update_table_link var_id sym_t other_id other_scope
other_nesting

        | _ -> ()



(* All variables same as above *)

let create_assignment_linkage_if_applicable var_id var_nesting sym_t assignee =

    (*The lookup scope is the same as the variable we're assigning to *)

    create_linkage_if_applicable var_id var_nesting sym_t assignee sym_t



(* add func_decl to the global environment

   IF there is not already a function with the same name
```

```
   and same number of parameters (where all the parameters have the same type)
*)
let add_func_to_global_env global_env func_decl =
    (*the unique signature of a function *)
    let func_decl_id fd =
        let param_types = List.map snd fd.params in
        (fd.fname, param_types)
    in
    let new_id = func_decl_id func_decl in
    let existing_ids = List.map func_decl_id global_env.funcs in
    if List.mem new_id existing_ids then
        ()
    else
        ignore (global_env.funcs <- func_decl::(global_env.funcs))


let add_func_sig_to_global_env global_env func_signature return_type =
    let rec replace_func_sig = function
        [] -> []
        | (other_sig,_)::tl when other_sig=func_signature ->
            (other_sig,return_type)::(replace_func_sig tl)
        | hd::tl -> hd::(replace_func_sig tl)
    in
    ignore (global_env.func_signatures <- (replace_func_sig
global_env.func_signatures))
```

```
(*

let func_signature_exists global_env func_signature =

    let same_function_name =

        List.filter (fun func_sig -> ((fst func_sig) = (fst func_signature)))
global_env.func_signatures

    in

    let same_signature_types = List.filter (fun func_sig -> (snd func_sig) =
(snd func_signature)) same_function_name

    in

    (match same_signature_types with

        [] -> false

        | _ -> true)



let add_func_signature_to_global_env global_env func_signature =

    ignore (global_env.func_signatures <- (func_signature::
(global_env.func_signatures)))

*)




(*

Convert an AST expression to an SAST expression!

*)

let rec check_expr env global_env = function

  Ast.TableLiteral(tl) -> check_table_literal env global_env tl
```

```
  | Ast.Literal(l) ->(

    match l with

     Ast.IntLiteral(v) -> Literal(l), Int

     | Ast.StringLiteral(v) -> Literal(l), String

     | Ast.DoubleLiteral(v) -> Literal(l), Double

     | Ast.This -> if env.is_pattern then Literal(l), Table(String) else raise
(Failure("This is a reserved word and is not defined in this context."))

     )

  | Ast.Id(v) ->

    let vdecl = try

      find env.scope v

    with Not_found ->

      raise (Failure("undeclared identifier " ^ v)) in

    let (v, typ) = vdecl in

    Id(v), typ

  | Ast.TableAssign(table_id, index_list, e) -> (*TODO: MAKE THIS SHIT MORE LIKE
ASSIGN WITH TABLE LINX AND SHIT *)

    let nesting = (List.length index_list) in

    let assignee_env = match e with

        Call(_) -> {env with return_assigner=(Some
{id=table_id;assign_scope=env.scope;nesting=nesting}) }

        | _ -> env

    in

    let (assignee_e, assignee_type) as assignee = check_expr env global_env e in

    assert_not_void assignee_type "Can't assign void to a table (or anything for
```

```
that matter).";

    let indices_sast = check_table_indices env global_env index_list in

    let (_,table_t) = try

      find env.scope table_id

    with Not_found ->

      raise (Failure("undeclared table identifier " ^ table_id)) in

    let assign_info = {id=table_id;assign_scope=env.scope;nesting=nesting} in

    let expr_promise = get_assignment_expression_promise assign_info global_env
assignee_e assignee_type in

    (* Most nested part of *)

    let final_table_t = apply_nesting (table_t,-(nesting - 1)) in

    let rec update_empty_table_container_type old_table_t new_type =

        match old_table_t with

            Table(t) -> (Table (update_empty_table_container_type t new_type))

            | EmptyTable -> Table(new_type)

            | t -> t

    in

    (*let new_table_type = (update_empty_table_container_type table_t
assignee_type) in*)

    let new_table_type = apply_nesting (assignee_type,nesting) in

    (*

    print_string ("new type of " ^ table_id ^ " is " ^ (type_to_str
new_table_type) ^"\n");

    *)

    update_table_type env.scope table_id new_table_type;
```

```
    (match table_t with

        Table(_) | EmptyTable ->

            let vdecl = match final_table_t with

                EmptyTable -> (* Going from a nested empty table to a different
level of nesting, update *)

                    TableAssign (table_id,
indices_sast,expr_promise),assignee_type

                |Table(val_type) ->

                    if (can_assign val_type assignee_type) then (*
val_type=assignee_type then *)

                        TableAssign
(table_id,indices_sast,expr_promise),assignee_type

                    else

                        raise (Failure "Trying to assign value to table with
incompatible type.")

                | _ -> raise (Failure "check_expr TableAssign: Shouldn't be
here. ")

            in

            (if (is_table assignee_type) then

                create_assignment_linkage_if_applicable table_id nesting
env.scope assignee_e

            );

            vdecl



        | _ -> raise (Failure "Cannot do table assignment for a non-table"))

  | Ast.Assign(v, assignee) ->

    let assign_info = {id=v;assign_scope=env.scope;nesting=0} in
```

```
    let assignee_env = match assignee with

        Call(_) -> {env with return_assigner= (Some assign_info) }

        | _ -> env

    in

    let (assignee_e, assignee_type) as assignee = check_expr assignee_env
global_env assignee in

    let expr_promise = get_assignment_expression_promise assign_info global_env
assignee_e assignee_type in

    assert_not_void assignee_type "Can't assign void to a variable.";

    let (new_e,new_type) as vdecl =

    try (*Reassigning a variable to a different type is okay because assigment =
declaration*)

      let (_,prev_typ) = find env.scope v in (*Add it in the symbol table if its
a different type*)

      if (not (can_assign prev_typ assignee_type)) then

        raise (Failure ("identifier type cannot be assigned to previously
declared type " ^ v))

      else

        Assign (v, expr_promise), assignee_type

    with Not_found -> (*Declaring/Defining a new variable*)

      let decl = (v, assignee_type) in env.scope.variables <- (decl ::
env.scope.variables) ;

      VAssign (v, expr_promise), assignee_type

    in

    (if (is_table new_type) then

        create_assignment_linkage_if_applicable v 0 env.scope assignee_e;

        (*
```

```
        print_string ("new type of " ^ v ^ " is " ^ (type_to_str new_type)
^"\n");

        *)

        update_table_type env.scope v new_type);

    vdecl

  | Ast.Binop(e1, op, e2) ->

    let e1 = check_expr env global_env e1

    and e2 = check_expr env global_env e2 in



    let _, t1 = e1

    and _, t2 = e2 in

    (*Operators come in*)

    let return_type = get_binop_type t1 op t2 in

    Binop(e1,op,e2),return_type

  | Ast.Uminus(e) ->

    let (e, typ) = check_expr env global_env e in

    (*Check for int or double*)

    if typ != Int && typ != Double && typ != UnknownReturn then raise
(Failure("unary minus operation does not support this type")) ;

    Uminus((e, typ)), typ

  | Ast.Call(v, el) ->

    let func_decl =

        try Some (List.assoc v env.func_decls)

        with Not_found -> None
```

```
    in

    (

    match func_decl with

        (* Function declaration found among user functions*)

        Some(func_decl) ->

            let el_typed = List.map (fun e -> (check_expr env global_env e)) el
in

            let (arg_exprs,arg_types) = List.split el_typed in

            let func_signature = (v, arg_types) in



            (* See if the function signature eixsts *)

            (

            match get_existing_func_sig global_env func_signature with

                Some(signature, t) -> CallStub (signature, el_typed), t

                | None ->



                (* Add function signature before proceeding *)

                add_initial_func_signature global_env func_signature;



                let typed_args = List.combine func_decl.params arg_types in

                let func_env = {env with scope = {parent = None; variables =
typed_args; update_table_links = []};

                                returns = ref [];

                                } in

                let link_argument (arg,assignee) =
```

```
                    create_linkage_if_applicable arg 0 func_env.scope assignee
env.scope

                in

                (*Make sure that if any empty tables are passed in, proper type
inference is done with them *)

                List.iter link_argument (List.combine func_decl.params
arg_exprs);

                let func_body = check_stmt func_env global_env (Ast.Block
func_decl.body) in

                let return_type_promise = get_return_type_promise func_body
func_env in (*func_env.scope func_body in*)

                (*TODO: find some way to link this with assignment as well *)

                let initial_return_type = get_return_type func_body func_env in

                (

                match func_body with

                    Block(stmt_list,_) ->

                        let func_body_list = stmt_list in

                        let arg_type_promises = List.map (get_var_type_promise
func_env.scope) func_decl.params in

                        let params = List.combine func_decl.params
arg_type_promises in

                        let func_decl_typed = { fname = v; params = params;

                                                body = func_body_list;

                                                return_type_promise =
return_type_promise } in

                        add_func_to_global_env global_env func_decl_typed;

                        add_func_sig_to_global_env global_env func_signature
initial_return_type;
```

```
                        let typed_func_call = Call(func_decl_typed, el_typed),
initial_return_type in

                        typed_func_call

                    | _ -> raise (Failure "Function body must be a Block.")

                )

            )

        (* No user function with this name defined. Check built-ins *)

        | None ->

            let el = List.map (fun e -> (check_expr env global_env e)) el in

            let (built_in_name, built_in_typs, return_typ) = find_built_in v in

            let check = try

                if (List.length el) = (List.length built_in_typs) then

                  List.map2 (fun (_, typ) j -> match typ, j with

                    String, BString -> true

                    | Table(_), BTable -> true

                    | Int, BInt -> true

                    | _, BAny -> true

                    | _,_ -> raise (Failure("Parameter types do not match for
function with name " ^ v))

                  ) el built_in_typs

                else

                  raise (Failure("Length of parameter lists do not match for
function with name " ^ v))

            with Not_found -> raise (Failure("Function with name " ^ v ^ " is
not defined in any space.")) in
```

```
            BCall(v, el), return_typ
    )
  | Ast.TableAccess(table_id,index_exprs) ->
    (*First, get table variable if it exists *)
    let (_,table_t) = try
      find env.scope table_id
    with Not_found ->
      raise (Failure("undeclared identifier " ^ table_id)) in
    (*next ensure that its a table*)
    if (is_table table_t) then
        let indices_sast = check_table_indices env global_env index_exprs in
        let nesting = (List.length index_exprs) in
        let final_table_t = apply_nesting (table_t,-(nesting)) in
        TableAccess (table_id,indices_sast),final_table_t
    else
        raise (Failure ("Attempting to index non-table"))
    | Ast.ThisAccess(expr) ->
        let ind_expr = check_expr env global_env expr in
        ThisAccess(ind_expr), String
and check_table_indices env global_env index_expr_lst =
    let index_sast = (List.map (check_expr env global_env) index_expr_lst) in
    let index_types = (List.map snd index_sast) in
    if not (List.for_all valid_table_index_type index_types) then
        raise (Failure("All table indices must be string or int expressions"))
```

```
    else
        index_sast
and check_table_literal env global_env tl =
    let get_unique_elt lst =
        if (Util.all_the_same lst) && (List.length lst)>0 then
            (List.hd lst)
        else
            raise (Failure("Array literal values must be the same types"))
    in
    let check_keys_exprs env global_env keys exprs =
        let values_sast = (List.map (check_expr env global_env) exprs) in
        let value_type = (get_unique_elt (List.map snd values_sast)) in
        (TableLiteral (List.combine keys values_sast)), (Table value_type)
    in
    match tl with
        Ast.EmptyTable -> (TableLiteral []),EmptyTable
        | Ast.ArrayLiteral(exprs) ->
            let keys = List.map (fun i -> (Ast.IntKey i)) (Util.range 0
(List.length exprs ))  in
            check_keys_exprs env global_env keys exprs
        | Ast.KeyValueLiteral(kv_list) ->
            let keys = (List.map fst kv_list) in
            let exprs = (List.map snd kv_list) in
            check_keys_exprs env global_env keys exprs
```

```
and check_stmt env global_env = function

  Ast.Block(sl) ->

    let scopeT = { parent = Some(env.scope); variables = []; update_table_links=
[] } in

    let envT = { env with scope = scopeT} in

    let sl = List.map (fun s -> (check_stmt envT global_env s)) sl in
envT.scope.variables <- List.rev scopeT.variables;

    Block (sl, envT)

  | Ast.Expr(e) ->

        (match e with

        Assign(_) | TableAssign(_) | Call(_) -> Expr(check_expr env global_env
e)

        | _ -> raise (Failure("Expression is not statement in Java")))

  | Ast.Empty -> Empty

  | Ast.Func(f) ->(

    try (*Test to see if user is trying to overwrite built-in function*)

      ignore(find_built_in f.Ast.fname) ;

      raise (Failure("function is overwrites built-in function " ^ f.Ast.fname))

    with Not_found -> (*valid function*)

      (*We handle func generation elsewhere so can make this a no-op *)

      Block([], env )

    )

  | Ast.Return(e) ->

    let (return_e,return_t) as return_expr = check_expr env global_env e in
```

```
    let expr_promise = match env.return_assigner with

        None ->

            get_expression_promise None global_env return_e return_t env.scope

        | Some(assigner) as assgn ->

            create_linkage_if_applicable assigner.id assigner.nesting
assigner.assign_scope return_e env.scope;

            get_expression_promise assgn global_env return_e return_t env.scope

    in

    let return_type_promise = fun () -> (snd (expr_promise ())) in

    env.returns:= return_type_promise::!(env.returns);

    Return(expr_promise)

  | Ast.If(e, s1, s2) ->

    let (e, typ) = check_expr env global_env e in

    if typ != Int && typ != Double then raise (Failure("If statement does not
support this type")) ;

    If((e, typ), check_stmt env global_env s1, check_stmt env global_env s2)

  | Ast.While(e, s) ->

    let (e, typ) = check_expr env global_env e in

    if typ != Int && typ != Double then raise (Failure("unary minus operation
does not support this type")) ;

    While((e, typ), check_stmt env global_env s)

  | Ast.For(key_id, table_id, stmt) ->

    let scopeT = { parent = Some(env.scope); variables = [(key_id,String)];
update_table_links=[] } in

    let envT = { env with scope = scopeT} in
```

```
    let stmt = check_stmt envT global_env stmt in

    let (_,table_t) = try

      find env.scope table_id

    with Not_found ->

      raise (Failure("Undeclared table identifier in for statement:" ^
table_id))

    in

    if is_table table_t then

        For(key_id,table_id,stmt)

    else

        raise (Failure("Cannot do for statement on non-table " ^ table_id))

    | Ast.ForThis(key_id, stmt) ->

        let scopeT = { parent = Some(env.scope); variables = [(key_id,String)];
update_table_links=[] } in

        let envT = { env with scope = scopeT} in

        let stmt = check_stmt envT global_env stmt in

        ForThis(key_id,stmt)



let check_pattern env global_env a = check_stmt env global_env a



let get_func_decls_stmt stmt =

    let rec get_func_decls_stmt_unchecked stmt=

        match stmt with

            Ast.Block(stmt_list) -> List.concat (List.map
get_func_decls_stmt_unchecked stmt_list)
```

```
            | Ast.Func(fdecl) -> [fdecl.fname,fdecl]
            | _ -> []
    in
    let func_decls = get_func_decls_stmt_unchecked stmt in
    (*Make sure that there are no duplicates*)
    let names = List.map fst func_decls in
    if (Util.have_duplicates String.compare names) then
        raise (Failure "Duplicate function names declared!")
    else
        func_decls



let check_program p =
    let func_decls = get_func_decls_stmt p.Ast.begin_stmt in
    let init_scope = {
      parent = None;
      variables = [];
      update_table_links = []} in
    let init_env = { scope = init_scope;
                     return = None;
                     func_decls = func_decls;
                     is_pattern = false;
                     return_assigner = None;
                     returns = ref [] } in
```

```
    let global_env = { funcs = []; func_signatures = []} in

    let (begin_block, env) = match check_stmt init_env global_env
p.Ast.begin_stmt with

                                Block(begin_block, env) -> begin_block, env

                                | _ -> raise (Failure("begin is not a block"))
in

  let env = {env with is_pattern = true} in

    let pattern_actions = List.map (fun (pattern, action) -> pattern,
(check_pattern env global_env action)) p.Ast.pattern_actions in

  let env = {env with is_pattern = false} in

  let (end_block, env) = match check_stmt env global_env p.Ast.end_stmt with

                                Block(end_block, env) -> end_block, env

                                | _ -> raise (Failure("end is not a block")) in

    {concrete_funcs = global_env.funcs;

    begin_stmt = Block(begin_block, env);

    pattern_actions = pattern_actions;

    end_stmt = Block(end_block, env);}
```

8.5 SAST

```
type t = Int | String | Double | Table of t | EmptyTable | Void | UnknownReturn


type type_promise = unit -> t


type func_signature = string * (t list)
```

```
(*update_table_link represents table variables that a given variable is
implicity attached to

consider this code:

t = {}

s = t

at that point, you will add {id:"t", nesting:0}

when the type of s updated from EmptyTable to Table(something), type of t will
become Table(something)



Sometimes you need more nesting. Consider:

t = {}

t[3] = {}

s = t[3]

at that point you will add {id:"t", nesting:1}

when type of s is updated from EmptyTable to Table(something), type of t will
become Table(Table(something))

*)

type assigner_info = {id: string; assign_scope: symbol_table; nesting:int}



and symbol_table = {

  parent: symbol_table option;

  mutable variables: (string*t) list;

  (*When an EmptyTable variable in this list has its type updated, must also
update linked variables *)

  mutable update_table_links: (string * assigner_info) list
```

```
}


type translation_environment = {
  func_decls: (string * Ast.func_decl) list;
  scope: symbol_table;
  is_pattern: bool;
  return: t option;
  (* Keeps track of all return statement types
     useful for assuring consistency of function call returns
  made as a reference so that children can update it
  *)
  return_assigner: assigner_info option;
  returns: (type_promise list) ref;

}


(* Represents a lazy or delayed computation for an expression
 used because we don't always initially know the type or form an expression
takes
 due to empty tables
*)


and table_literal = (Ast.key_literal * expr_t) list
and expr_det =
```

```
  Id of string

  |Literal of Ast.literal

  |TableLiteral of table_literal (*Every table literal, at the end of the day,
is keys and values (possibly none) *)

  |VAssign of string * expr_t_promise

  |Assign of string *  expr_t_promise

  |TableAssign of string * (expr_t list) * expr_t_promise

  |Binop of expr_t * Ast.op * expr_t

  |Uminus of expr_t

  |Call of func_decl_t * (expr_t list) (* function, parameters *)

  |CallStub of func_signature * (expr_t list)

  |BCall of string * (expr_t list) (* built-in function name, parameters *)

  |TableAccess of string * (expr_t list)

  |ThisAccess of expr_t

and expr_t = expr_det * t

(* Sometimes the accurate type of an expression is not known in advance

(when empty tables are involved)

, so we defer type determination til later

by using a closure

*)

and expr_t_promise = unit -> expr_t

and stmt_t =

  Block of stmt_t list * translation_environment

  | Expr of expr_t
```

```
  | Func of func_decl_t
  | Return of expr_t_promise
  | If of expr_t * stmt_t * stmt_t
  | While of expr_t * stmt_t
  | For of string * string * stmt_t
  | ForThis of string * stmt_t
  | Empty
  and
  func_decl_t = {
    fname : string;
    params : (string * type_promise) list;
    body : stmt_t list;
    return_type_promise : type_promise
  }

type global_environment = {
  mutable funcs: func_decl_t list;
  mutable func_signatures: (func_signature * t) list;
}

type pattern_action_t = Ast.pattern * stmt_t

type program_t = {
```

```
    concrete_funcs: func_decl_t list;

    begin_stmt : stmt_t;

    pattern_actions : pattern_action_t list;

    end_stmt : stmt_t;

}
```

8.6 Code Generator

```
open Sast


let strip_quotes str =

  match String.length str with

  | 0 | 1 | 2 -> ""

  | len -> String.sub str 1 (len - 2)


let rec type_to_str = function

  Int -> "int"

  | Double -> "double"

  | Table(value_type) as t -> (type_to_boxed_str t)

  | String -> "String"

    | Void -> "void"

    | EmptyTable -> "wtf" (*raise (Failure "EmptyTable should never be
generated. semantics messed up") *)


and type_to_boxed_str = function
```

```
    Int -> "Integer"
    | Double -> "Double"
    | Table(value_type) -> "_HAWKTable<" ^ (type_to_boxed_str value_type) ^ ">"
    | t -> type_to_str t


let rec repeat str n =
    match str,n with
        |s,0 -> ""
        |s,1 -> s
        |s,n -> s ^ (repeat str (n-1))


let string_for_indent indent =
    repeat "\t" indent


let rec string_of_regex_set = function
  Ast.RegexStringSet(str) -> str
  | Ast.RegexRangeSet(str1, str2) -> str1 ^ "-" ^ str2
  | Ast.RegexComplementSet(set) -> "^" ^ (string_of_regex_set set)
  | Ast.RegexNestedSet(set) -> (string_of_regex_set_sequence set)


and string_of_regex_set_sequence seq =
    let all_together = String.concat "" (List.map string_of_regex_set seq) in
    "[" ^ all_together ^ "]"
```

```
let string_of_regex_op op =

    match op with

        | Ast.Or -> "|"

        | Ast.Optional -> "?"

        | Ast.KleenePlus -> "+"

        | Ast.KleeneTimes -> "*"



let rec string_of_regex regex = match regex with

  Ast.RegexString(str) -> str

  | Ast.RegexAnyChar -> "."

  | Ast.RegexNested(sequence) -> "(" ^ (string_of_regex_sequence sequence) ^ ")"

  | Ast.RegexSet(sequence) -> string_of_regex_set_sequence sequence

  | Ast.RegexUnOp(re,op) -> (string_of_regex re) ^ (string_of_regex_op op)

  | Ast.RegexBinOp(re1,op,re2) -> (string_of_regex re1) ^ (string_of_regex_op
op) ^ (string_of_regex re1)



and string_of_regex_sequence seq = String.concat "" (List.map string_of_regex
seq)



let string_of_property prop = match prop with

    |Ast.ClassMatch(s) -> "." ^ s

    |Ast.IdMatch(s) -> "#" ^s

    | Ast.AttributeExists(str) -> "[" ^ (strip_quotes str) ^ "]"

    | Ast.AttributeEquals(attr, str) -> "[" ^ attr ^ "=" ^ (strip_quotes str) ^
```

```
"]"

    | Ast.AttributeContains(attr,str) -> "[" ^ attr ^ "*=" ^ (strip_quotes str)
^ "]"

    | Ast.AttributeBeginsWith(attr,str) -> "[" ^ attr ^ "^=" ^ (strip_quotes
str) ^ "]"

    | Ast.AttributeEndsWith(attr,str) -> "[" ^ attr ^ "$=" ^ (strip_quotes str)
^ "]"

    | Ast.AttributeWhitespaceContains(attr,str) -> "[" ^ attr ^ "~=" ^
(strip_quotes str) ^ "]"



let string_of_type_selector = function

    |Ast.Elt(str) -> str

    | Ast.Universal -> "*"

    | Ast.NoType -> ""



let string_of_simple_selector_seq (type_selector,props) =

    let prop_string = List.fold_left (fun acc prop -> acc ^ (string_of_property
prop)) "" props in

    (string_of_type_selector type_selector) ^ prop_string



let string_of_combinator = function

    | Ast.DirectChild -> " "

    | Ast.Descendent -> ">"

    | Ast.DirectSibling -> "+"

    | Ast.AnySibling -> "~"
```

```
let rec string_of_css_selector css_selector = match css_selector with

    | Ast.SingleSelector(seq) -> (string_of_simple_selector_seq seq)

    | Ast.ChainedSelectors(selector,comb,seq) -> (string_of_css_selector
selector) ^ (string_of_combinator comb) ^ (string_of_simple_selector_seq seq)



let string_of_op = function

    Ast.Plus ->  " + "

    | Ast.Minus ->  " - "

    | Ast.Times ->  " * "

    | Ast.Divides ->  " / "

  | Ast.Mod -> "%"

    | Ast.Equal ->  " == "

    | Ast.NotEqual ->  " != "

    | Ast.Less ->  " < "

    | Ast.LessEqual ->  " <= "

    | Ast.Greater ->  " > "

    | Ast.GreaterEqual ->  " >= "

  | Ast.BAnd -> " && "

  | Ast.BOr -> " || "



let string_of_key_literal = function

    Ast.IntKey(key) -> string_of_int key

    | Ast.StringKey(key) -> key
```

```
(*TODO: these don't need to all be mutually recursive*)

let rec string_of_table_literal kv_list table_t =

    let string_of_kv = function

        | Ast.IntKey(i), expr -> ".setIntIndexChained(" ^ (string_of_int i) ^
"," ^ (string_of_expr expr) ^ ")"

        | Ast.StringKey(s), expr -> ".setStringIndexChained(" ^ s ^ "," ^
(string_of_expr expr) ^ ")"

    in

    let kv_part = String.concat "" (List.map string_of_kv kv_list) in

    "(new " ^ (type_to_str (Table (table_t))) ^ "())" ^ kv_part

and

string_of_literal = function

    Ast.IntLiteral(x), _ -> string_of_int x

    | Ast.StringLiteral(str), _ -> str

    | Ast.DoubleLiteral(dbl), _ -> string_of_float dbl

    | Ast.This, _-> "_this"

and string_of_expr_list = function

    [] -> ""

    | [hd] -> string_of_expr hd

    | hd::tl -> (string_of_expr hd) ^ ", " ^ string_of_expr_list tl

and string_of_get_index_expr = function

    ind_e,Int as e -> ".getIntIndex(" ^ (string_of_expr e) ^ ")"

    | ind_e,String as e -> ".getStringIndex(" ^ (string_of_expr e) ^ ")"

    | _ -> raise (Failure "This type of table indexing should not happen.
```

```
Semantic stage must have failed.")

and string_of_set_index_expr ind value =

    string_of_set_index_expr_with_value_str ind (string_of_expr value)

and string_of_set_index_expr_with_value_str ind value_str =

    let inner = ((string_of_expr ind) ^ "," ^ value_str) in

    match ind with

        (_,Int) -> ".setIntIndex(" ^ inner ^ ")"

        | (_,String)  -> ".setStringIndex(" ^ inner ^ ")"

        | _ -> raise (Failure "This type of table indexing should not happen.
Semantic stage must have failed.")

and

string_of_expr = function

    Id(id), _ -> id

    | Literal(lit), t -> string_of_literal (lit,t)

    (* Assigning to something that stays empty for the duration of the program
is a no-op *)

    | TableLiteral(kv_list), Table(val_type) -> string_of_table_literal kv_list
val_type

    | VAssign(id, expr_promise), t ->

        let (_,typ) as expr = expr_promise () in

        if (Semantics.is_empty_table_container typ) then

            ""

        else

            (type_to_str typ) ^ " " ^ id ^ " = " ^ (string_of_expr expr)

    | Assign(id, expr_promise), t ->
```

```
        let (_,typ) as expr = expr_promise () in

        if (Semantics.is_empty_table_container typ) then

            ""

        else

            id ^ " = " ^ (string_of_expr expr)

    | Binop(expr1, op, expr2), _ -> (string_of_expr expr1) ^ (string_of_op op) ^
(string_of_expr expr2)

    | Uminus(expr), _ -> "-" ^ (string_of_expr expr)

    | BCall(id, expr_list), _ -> id ^ "(" ^ string_of_expr_list expr_list ^ ")"

    | Call(fdecl, expr_list), _ -> fdecl.fname ^ "(" ^ string_of_expr_list
expr_list ^ ")"

    | CallStub(func_sig,expr_list),_ -> (fst func_sig) ^ "(" ^
string_of_expr_list expr_list ^ ")"

    | TableAccess(table_id, ind_list), _ ->

        table_id ^ (String.concat "" (List.map string_of_get_index_expr
ind_list))

    | ThisAccess(expr),_ ->

        string_of_expr ((TableAccess ("_this",[expr])),String)

    | TableAssign(table_id,ind_list,expr_promise), t ->

        let (_,typ) as expr = expr_promise () in

        if (Semantics.is_empty_table_container typ) then

            ""

        else

            let nesting_level = (List.length ind_list) in

            let nestings = (Util.range 1 (nesting_level+1)) in

            let enum_ind_list = (List.combine ind_list nestings) in
```

```
            let value_str =  string_of_expr expr

            in

            (*

            a[1][2][3] = 4 gets an inner table, which gets an inner table, which
then sets index 3 to 4

            *)

            let ind_to_string (ind_expr,nesting) =

                match ind_expr,nesting with

                    ind_e,n when n=nesting_level ->
string_of_set_index_expr_with_value_str ind_e value_str

                    |ind_e,_ -> string_of_get_index_expr ind_e

            in

            table_id ^ (String.concat "" (List.map ind_to_string enum_ind_list))

    | _ -> raise (Failure "We shouldn't be here.")

and string_of_func_decl func_decl  =

    let param_names = (List.map fst func_decl.params) in

    func_decl.fname ^ "(" ^ (String.concat "," param_names) ^ ")" ^
(string_of_stmt_list func_decl.body 0)

and

string_of_stmt_list stmt_list nested = match stmt_list with

    [] -> ""

    | hd::tl -> (string_of_stmt hd nested) ^ "\n" ^ (string_of_stmt_list tl
nested)

and string_of_stmt stmt nested = match stmt with

    Block(stmt_list, _) -> "{\n" ^ (string_of_stmt_list stmt_list nested) ^ "\n"
^ (string_for_indent (nested - 1)) ^ "}"
```

```
    | Expr(expr) -> (string_for_indent nested) ^ (string_of_expr expr) ^ ";"

    | Func(func_decl) -> ""

    | Return(expr_promise) ->

        let expr = (expr_promise ()) in

        (string_for_indent nested) ^ "return " ^ (string_of_expr expr) ^ ";"

    | If(expr, stmt1, stmt2) -> (match stmt2 with

        | Block([], _)  -> (string_for_indent nested) ^ "if(_checkIf(" ^
(string_of_expr expr) ^ "))" ^ (string_of_stmt stmt1 nested)

        | _ -> (string_for_indent nested) ^ "if(_checkIf(" ^ (string_of_expr
expr) ^ "))" ^ (string_of_stmt stmt1 nested) ^ "else" ^ (string_of_stmt stmt2
nested))

    | While(expr, stmt) -> (string_for_indent nested) ^ "while(" ^
(string_of_expr expr) ^ ")" ^ (string_of_stmt stmt (nested + 1))

  | Empty -> ""

    | For(key_id, table_id, stmt) ->

        (string_for_indent nested) ^ "for(String " ^ key_id ^ " : " ^ table_id ^
".getKeys())" ^ (string_of_stmt stmt nested)

    | ForThis(key_id, stmt) ->

        string_of_stmt (For (key_id,"_this",stmt)) nested



let string_of_begin_end block nested= match block with

  Block(stmt_list, _) -> string_of_stmt_list stmt_list nested

  | _ -> ""



let string_of_pattern pat = match pat with

        Ast.CssPattern(css_selector) -> "for(_HAWKTable<String> _this :
```

```
_cssMatcher._match(\"" ^ (string_of_css_selector css_selector) ^"\"))"

        | Ast.RegexPattern(regex_seq) -> "for(_HAWKTable<String> _this :
_regexMatcher._match(\""^string_of_regex_sequence regex_seq^"\"))"



let string_of_pattern_action nested (pattern,action) =

    (string_of_pattern pattern) ^ (string_of_stmt action nested)



let string_of_file file nested =

  let file_string = ref "" in

  let ic = open_in file in

    try

      while true; do

        file_string :=  !file_string ^ (string_for_indent nested) ^ (input_line
ic) ^ "\n";

      done; !file_string

    with End_of_file ->

      close_in ic;

  !file_string



 let string_of_typed_param (name,type_promise) =

    let type_str = type_to_str (type_promise ()) in

    type_str ^ " " ^ name



 let string_of_user_func nested func_decl =

    (string_for_indent nested) ^ "public static " ^ (type_to_str
```

```
(func_decl.return_type_promise ()))

    ^ " " ^ func_decl.fname ^ "(" ^ String.concat "," (List.map
string_of_typed_param func_decl.params) ^ ")"

    ^ "{\n" ^  (string_of_stmt_list func_decl.body (nested + 1)) ^  "\n" ^
(string_for_indent nested) ^ "}"



 let string_of_user_funcs func_decls nested =

    String.concat "\n\n" (List.map (string_of_user_func nested) func_decls)



let string_of_program prog =

  (string_of_file "Imports.java" 0)

    ^  "public class Program {\n"

    ^ (string_for_indent 1) ^ "public static void main(String[] _args){" ^ "\n"

  ^  (string_of_file "Setup.java" 2)

    ^ (string_of_begin_end prog.begin_stmt 2)

    ^ (String.concat "\n" (List.map (string_of_pattern_action 2)
prog.pattern_actions)) ^"\n"

    ^ (string_of_begin_end prog.end_stmt 2) ^ "\n" ^ (string_for_indent 1) ^
"}\n"

  ^ (string_of_file "BuiltIn.java" 1)

  ^ (string_of_user_funcs prog.concrete_funcs 1) ^ "\n"

  ^ "}"
```

8.7 Hawk Lib and Built-In Functions

8.7.1 _HAWKFileReader.java

```
package _hawk_lib;



public class _HAWKFileReader{

  public _HAWKFileReader(String[] files){

    concatFile = "";

    if(files.length > 0){

      for(String s : files){

        try {

            concatFile += org.jsoup.Jsoup.connect(s).get().toString();

        } catch (Exception e) {

            try{

              byte[] encoded =
java.nio.file.Files.readAllBytes(java.nio.file.Paths.get(s));

              concatFile += new String(encoded,
java.nio.charset.StandardCharsets.UTF_8);

            }catch(java.io.IOException e1){

              System.out.println(s+" does not seem to be valid and/or exist");

            }

        }

      }

    }

  }

  public String _getConcatFile(){

    return concatFile;

  }
```

```
  private String concatFile;

}
```

8.7.2 _HAWKRegexMatcher.java

```
package _hawk_lib;



public class _HAWKRegexMatcher{

  public _HAWKRegexMatcher(String aInput){

    input = aInput;

  }

  public _HAWKTable<_HAWKTable<String>> _match(String pat){

      matcher = java.util.regex.Pattern.compile(pat).matcher(input);

      _HAWKTable<_HAWKTable<String>> allMatches = new
_HAWKTable<_HAWKTable<String>>();

      int i = 0;

      while(matcher.find()){

        _HAWKTable<String> temp = new _HAWKTable<String>();

        temp.setIntIndex(0, matcher.group());

        allMatches.setIntIndex(i, temp);

        i++;

      }

      return allMatches;

  }

  java.util.regex.Matcher matcher;

  String input;

}
```

8.7.3 _HAWKCSSMatcher.java

```
package _hawk_lib;


public class _HAWKCSSMatcher{

    public _HAWKCSSMatcher(String aInput){

        input = aInput;

    }

    public _HAWKTable<_HAWKTable<String>> _match(String pat){

        _HAWKTable<_HAWKTable<String>> _allMatches = new _HAWKTable<>();

        org.jsoup.nodes.Document doc = org.jsoup.Jsoup.parse(input);

        int ctr = 0;

        for(org.jsoup.nodes.Element e : doc.select(pat)){

            _HAWKTable<String> _matches = new _HAWKTable<>();

            // inserts empty string if no id

            _matches.setStringIndex("id", e.id());

            // insert attributes as key-val pairs

            org.jsoup.nodes.Attributes elementAttributes = e.attributes();

            for(org.jsoup.nodes.Attribute a : elementAttributes){

                _matches.setStringIndex(a.getKey(), a.getValue());

            }


            try{

                _matches.setStringIndex("nextChild", e.child(0).id());
```

```
            }
            catch(IndexOutOfBoundsException err){
                _matches.setStringIndex("nextChild", "");
            }

            // insert innerhtml into table
            _matches.setStringIndex("innerhtml", e.html());
            _allMatches.setIntIndex(ctr++, _matches);
        }

        return _allMatches;
    }

    String input;
}
```

8.7.4 _HAWKTable.java

```
package _hawk_lib;

public class _HAWKTable<T> implements java.lang.Iterable<T>{

    private java.util.Hashtable<String,T> storage;
```

```
    public _HAWKTable(){

        storage = new java.util.Hashtable<String,T>();

    }



    //ensure that HAWK Tables have a consistent mechanism of converting ints to
strings and back

    private static String intToString(int i){

        return i + "";

    }



  public T getIntIndex(int i){

      return storage.get(intToString(i));

  }



  public T getStringIndex(String s){

      return storage.get(s);

  }



  public _HAWKTable setIntIndexChained(int i, T t){

      storage.put(intToString(i),t);

      return this;

  }



  public _HAWKTable setStringIndexChained(String s, T t){
```

```
    storage.put(s,t);
    return this;
}

public void setIntIndex(int i, T t){
    storage.put(intToString(i),t);
}

public void setStringIndex(String s, T t){
    storage.put(s,t);
}

  public java.util.Set<String> getKeys(){
      return storage.keySet();
  }

  public int getLength(){
      return storage.size();
  }

  public java.util.Iterator<T> iterator() {
      return new MyIterator();
  }
```

```
  class MyIterator implements java.util.Iterator<T> {


      private String[] keys;

      private int i;


      public MyIterator(){

          keys = storage.keySet().toArray(new String[storage.size()]);

          i = 0;

      }

      public boolean hasNext() {

           return i < keys.length;

      }

      public T next() {

           return storage.get(keys[i++]);

      }

      public void remove() {

           ;

      }

  }


public String toString(){

    String temp = "";

    for( java.util.Map.Entry<String, T> entry : storage.entrySet() ){
```

```
          temp += entry.toString()+"\n";
      }
            temp = temp.substring(0, temp.length() - 1);
            // temp = new java.lang.StringBuilder(temp).reverse().toString();
      return temp;
  }
}
```

8.7.5 BuiltIn.java

```
private static void _print_(Object o){
  System.out.println(o);
}
private static String _charAt_(String s, Integer i){
  return ""+s.charAt(i);
}
private static Integer _stringEqual_(String s1, String s2){
  if(s1.equals(s2)){
    return 1;
  }else{
    return 0;
  }
}
private static Boolean _checkIf(Integer i){
```

```
  return (i != 0) ? true : false;
}
private static Boolean _checkIf(Double d){
  return (d != 0.0) ? true : false;
}
private static Boolean _checkIf(Boolean b){
  return b;
}
private static int _length_(String s){
  return s.length();
}
private static <T> int _length_(_HAWKTable<T> t){
  return t.getLength();
}

private static <T> int _exists_(T e){
  if(e == null){
    return 0;
  }
  return 1;
}

private static int _stringToInt_ (String s){
```

```
    return Integer.parseInt(s);
}


private static <T> _HAWKTable<String> _keys_(_HAWKTable<T> t){
  java.util.Set<String> set = t.getKeys();
  _HAWKTable<String> hawkTable = new _HAWKTable<String>();
  int i = t.getLength()-1;
  for (String s : set) {
    hawkTable.setIntIndex(i, s);
    i--;
  }
  return hawkTable;
}
```

8.8 Makefile and Test Script

8.8.1 Makefile

```
OBJS = util.cmo parser.cmo scanner.cmo prettyprint.cmo semantics.cmo
generator.cmo main.cmo


prog : $(OBJS) ./_hawk_lib/*.java
    OCamlc -o prog $(OBJS)
    javac -cp ./_hawk_lib/jsoup.jar ./_hawk_lib/*.java


.PHONY : test
```

```
test : prog testall.sh
    ./testall.sh

scanner.ml : scanner.mll
    OCamllex scanner.mll

parser.ml parser.mli : parser.mly
    OCamlyacc -v parser.mly

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

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

.PHONY : clean
clean :
    rm -f prog parser.ml parser.mli scanner.ml testall.log \
    *.cmo *.cmi *.out *.cmx *.diff *.exe testresult/*.out \
    testresult/*.diff Program.java *.class \
    parser.output \
    _hawk_lib/*.class
```

```
# Generated by OCamldep *.ml *.mli

prog.cmo:  util.cmo scanner.cmo parser.cmi ast.cmi sast.cmi prettyprint.cmo
semantics.cmo generator.cmo

prog.cmx:  scanner.cmx parser.cmx semantics.cmx generator.cmx sast.cmi ast.cmi

parser.cmo: ast.cmi parser.cmi

parser.cmx: ast.cmi parser.cmi

scanner.cmo: parser.cmi

scanner.cmx: parser.cmx

semantics.cmo: sast.cmi

parser.cmi: ast.cmi
```

8.8.2 Test Script

```
#!/bin/sh


PROG="./prog"


# Set time limit for all operations

ulimit -t 30


globallog=testall.log

rm -f $globallog

error=0

globalerror=0
```

```
keep=0


successes=0

failures=0

totalTests=0


Usage() {

    echo "Usage: testall.sh [options] [.hk files]"

    echo "-k    Keep intermediate files"

    echo "-h    Print this help"

    exit 1

}


SignalError() {

    if [ $error -eq 0 ] ; then

    echo "FAILED"

    error=1

    fi

    echo "  $1"

}


# Compare <outfile> <reffile> <difffile>

# Compares the outfile with reffile.  Differences, if any, written to difffile
```

```
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
    SignalError "$1 differs"
    echo "FAILED $1 differs from $2" 1>&2
    }
}

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

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                              s/.hk//'`
    basename="testresult/$basename"
```

```
    reffile=`echo $1 | sed 's/.hk$//'`

    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."



    echo -n "$basename..."



    echo 1>&2

    echo "###### Testing $basename" 1>&2



    generatedfiles=""



    generatedfiles="$generatedfiles ${basename}.i.out" &&

    Run "$PROG" $1 ">" "Program.java && javac Program.java && java -cp
./_hawk_lib/jsoup.jar: Program ./tests/testinput.txt
http://www.wikicu.com/Srikar_Varadaraj >" ${basename}.i.out &&

    Compare ${basename}.i.out ${reffile}.out ${basename}.i.diff



    # Report the status and clean up the generated files



    if [ $error -eq 0 ] ; then

    if [ $keep -eq 0 ] ; then

        rm -f $generatedfiles

    fi

    echo "OK"

    echo "###### SUCCESS" 1>&2
```

```
    ((successes++))
    ((totalTests++))
    else
    echo "###### FAILED" 1>&2
    globalerror=$error
    ((failures++))
    ((totalTests++))
    fi
}

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

shift `expr $OPTIND - 1`

if [ $# -ge 1 ]
```

```
then
    files=$@
else
    files="tests/fail-*.hk tests/test-*.hk"
fi

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

echo "successes = $successes"
```

```
echo "failures = $failures"

echo "total tests = $totalTests"



exit $globalerror
```

8.9 Tests

hello.hk

```
BEGIN{

    print("hello world");

}

END{}
```

test-cf-exists-1.hk

```
BEGIN{
    table = {0,1,2};
    if(exists(table[0])){
      print("exists");
    }
    if(exists(table[4]) == 0){
      print("nope");
    }
}
END{}
```

test-cf-exists-1.out

```
exists
nope
```

test-cf-if-1.hk

```
BEGIN{
    i = 0;
    if(i < 0){
        baz = 1 + 1;
        print(baz);
    }
    print("I should be here");
}
END{}
```

test-cf-if-1.out

```
I should be here
```

test-cf-if-2.hk

```
BEGIN{
    i = 0;
    if(i < 0){


    }
    else{
        baz = "hello";
        print(baz);
    }
}
END{}
```

test-cf-if-2.out

```
hello
```

test-cf-if-3.hk

```
BEGIN{
    i = -1;
    if(i < 0){
        i = 2;
    }
    else{
        ;
    }
    print(i);
}
END{}
```

test-cf-if-3.out

```
2
```

test-cf-keys-1.hk

```
BEGIN{
  table = {0,1,2};
  print(table);
  print("--------");
  print(keys(table));
}
END{}
```

test-cf-keys-1.out

```
2=2
1=1
0=0
--------
2=2
1=1
0=0
```

test-cf-length-1.hk

```
BEGIN{
    table = {0,1};
    x = length(table);
    print(x);
}
END{}
```

test-cf-length-1.out

```
2
```

test-cf-print-1.hk

```
BEGIN{
    print("Hello world!");
}
END{}
```

test-cf-print-1.out

```
Hello world!
```

test-cf-while-1.hk

```
BEGIN{
    i = 0;
    while(i < 1){
        i = i + 1;
    }
    print(i);
}
END{}
```

test-cf-while-1.out

```
1
```

test-cf-while-2.hk

```
BEGIN{
    i = 0;
    while(i < 1){
        i = 2;
    }
    print(i);
}
END{}
```

test-cf-while-2.out

```
2
```

test-for-1.hk

```
BEGIN {
    t = {1,2,3,4,5};
    sum = 0;
    for (key in t){
        sum = sum + t[key];
    }
    print(sum);
}

END {}
```

test-for-1.out

```
15
```

test-for-2.hk

```
BEGIN {
    t = {{1,2},{3,4}};
    sum = 0;
    for (key in t){
        inner_table = t[key];
        for (key2 in inner_table){
            sum = sum + inner_table[key2];
        }
    }
    print(sum);
}

END {}
```

test-for-2.out

```
10
```

test-fun-1.hk

```
BEGIN {
    fun f(a){
        return a;
    }
    x = f(3);
    print(x);
}
END {


}
```

test-fun-1.out

```
3
```

test-fun-10.hk

```
BEGIN {
    fun f(b,c){

        b[3] = 42;
        return b;
    }

    a = {};
    d = f(a,a);
    print(d[3]);

}
END {

}
```

test-fun-10.out

```
42
```

test-fun-11.hk

```
BEGIN {
    fun f(x){
        return x;
    }

    x = f(42);
    print(x);
}
END {

}
```

test-fun-11.out

```
42
```

test-fun-12.hk

```
BEGIN {
    fun f(){
        y = {};
        return y;

    }

    x = f();
    x[0] = 42;
    print(x[0]);



}
END {

}
```

test-fun-12.out

```
42
```

test-fun-13.hk

```
BEGIN {
    fun f(){
        y = {{}};
        return y;

    }

    x = f();
    x[0][0] = 42;
    print(x[0][0]);



}
END {

}
```

test-fun-13.out

```
42
```

test-fun-14.hk

```
BEGIN {
    fun f(a){
        if(a == 1){
            return 1;
        }
        return f(a - 1);
    }

    print(f(3));



}
END {


}
```

test-fun-14.out

```
1
```

test-fun-15.hk

```
BEGIN {


    fun f1(x){
        return f2(34);
    }
    fun f2(a){
        return "dog";
    }


    x = f1(10);
    print(x);


}
END {


}
```

test-fun-15.out

```
dog
```

test-fun-16.hk

```
BEGIN {


    fun fac(x){
        if(x==1){
            return 1;
        }


        return fac(x-1)*x;
    }


    x = fac(3);
    print(x);


}
END {


}
```

test-fun-16.out

```
6
```

test-fun-2.hk

```
BEGIN {

    fun f(a){

        return a;

    }

    x = f("hello");

    print(x);

}

END {


}
```

test-fun-2.out

```
hello
```

test-fun-3.hk

```
BEGIN {
    fun f(a){
        return a;
    }
    x = f(4.5);
    print(x);
}
END {


}
```

test-fun-3.out

```
4.5
```

test-fun-4.hk

```
BEGIN {

    fun f(a){

        return a;

    }

    x = f(4.5);

    print(x);

}

END {



}
```

test-fun-4.out

```
4.5
```

test-fun-5.hk

```
BEGIN {

    fun f(a,b){

        return a + b;

    }

    x = f(1.5, 1);

    print(x);

}

END {


}
```

test-fun-5.out

```
2.5
```

test-fun-6.hk

```
BEGIN {

    fun f(a,b){

        return a + b;

    }

    x = f("hello", 1);

    print(x);

}

END {


}
```

test-fun-6.out

```
hello1
```

test-fun-7.hk

```
BEGIN {
    fun f(a,b){
        y = 2;
        z = 4;
        return a + b;
    }
    x = f("hello", 1);
    print(x);
}
END {

}
```

test-fun-7.out

```
hello1
```

test-fun-8.hk

```
BEGIN {
    fun f(a,b){
        if(a==2){
            return a + b;
        }
        else{
            return 3;
        }
    }
    x = f(2, 4);
    print(x);
}
END {

}
```

test-fun-8.out

```
6
```

test-fun-9.hk

```
BEGIN {
    fun f(a){
        print(a);
    }
    f(42);
}
END {

}
```

test-fun-9.out

```
42
```

test-op-arith1.hk

```
BEGIN{
  i = 1+1;
  print(i);
}
END{}
```

test-op-arith1.out

```
2
```

test-op-arith10.hk

```
BEGIN{
  print(1 - 1.0);
}
END{}
```

test-op-arith10.out

```
0.0
```

test-op-arith11.hk

```
BEGIN{
  print(1.0 - 1.0);
}
END{}
```

test-op-arith11.out

```
0.0
```

test-op-arith12.hk

```
BEGIN{
  print(1 * 1);
}
END{}
```

test-op-arith12.out

```
1
```

test-op-arith13.hk

```
BEGIN{
  print(1 * 1.0);
}
END{}
```

test-op-arith13.out

```
1.0
```

test-op-arith14.hk

```
BEGIN{
  print(1.0 * 1.0);
}
END{}
```

test-op-arith14.out

```
1.0
```

test-op-arith15.hk

```
BEGIN{
  print(1 / 1);
}
END{}
```

test-op-arith15.out

```
1
```

test-op-arith16.hk

```
BEGIN{
  print(1 / 1.0);
}
END{}
```

test-op-arith16.out

```
1.0
```

test-op-arith17.hk

```
BEGIN{
  print(1.0 / 1.0);
}
END{}
```

test-op-arith17.out

```
1.0
```

test-op-arith18.hk

```
BEGIN{
  print(1 % 1);
}
END{}
```

test-op-arith18.out

```
0
```

test-op-arith19.hk

```
BEGIN{
  print(1 % 1.0);
}
END{}
```

test-op-arith19.out

```
0.0
```

test-op-arith20.hk

```
BEGIN{
  print(1.0 % 1.0);
}
END{}
```

test-op-arith20.out

```
0.0
```

test-op-arith22.hk

```
BEGIN{
  print(-1);
}
END{}
```

test-op-arith22.out

```
-1
```

test-op-arith23.hk

```
BEGIN{
  print(-1.0);
}
END{}
```

test-op-arith23.out

```
-1.0
```

test-op-arith24.hk

```
BEGIN{
    if(1!=1){
        print("Dog");
    }
    else{
        print(42);
    }
}
END{}
```

test-op-arith24.out

```
42
```

test-op-arith3.hk

```
BEGIN{
  print(1 + 1);
}
END{}
```

test-op-arith3.out

```
2
```

test-op-arith4.hk

```
BEGIN{
  print(1 + 1.0);
}
END{}
```

test-op-arith4.out

```
2.0
```

test-op-arith5.hk

```
BEGIN{
  print(1 + "test");
}
END{}
```

test-op-arith5.out

```
1test
```

test-op-arith6.hk

```
BEGIN{
  print(1.0 + "test");
}
END{}
```

test-op-arith6.out

```
1.0test
```

test-op-arith7.hk

```
BEGIN{
  print("test" + "test");
}
END{}
```

test-op-arith7.out

```
testtest
```

test-op-arith8.hk

```
BEGIN{
 print(1.0 + 1.0);
}
END{}
```

test-op-arith8.out

```
2.0
```

test-op-arith9.hk

```
BEGIN{
  print(1 - 1);
}
END{}
```

test-op-arith9.out

```
0
```

test-pattern-css1.hk

```
/* passing http://www.wikicu.com/Srikar_Varadaraj */
BEGIN{}
[@#Basketball_Scouting_Report@]{
    print(this);
}
END{}
```

test-pattern-css1.out

```
innerhtml=Basketball Scouting Report
nextChild=
class=mw-headline
id=Basketball_Scouting_Report
```

test-pattern-css10.hk

```
BEGIN{}
[@a[href^="http://www.wikicu.com/Main_Page"]@]{
    print(this);
}
END{}
```

test-pattern-css10.out

```
rel=nofollow
href=http://www.wikicu.com/Main_Page
innerhtml=Main page
nextChild=
id=
```

test-pattern-css11.hk

```
BEGIN{}
[@a[href~="http://www.wikicu.com/Main_Page"]@]{
    print(this);
}
END{}
```

test-pattern-css11.out

```
rel=nofollow

href=http://www.wikicu.com/Main_Page

innerhtml=Main page

nextChild=

id=
```

test-pattern-css12.hk

```
BEGIN{}

[@h2 *@]{

  print(this);

}

END{}
```

test-pattern-css12.out

```
innerhtml=Attractiveness

nextChild=

class=mw-headline

id=Attractiveness

innerhtml=Mating Calls

nextChild=

class=mw-headline

id=Mating_Calls

innerhtml=Basketball Scouting Report

nextChild=

class=mw-headline

id=Basketball_Scouting_Report

innerhtml=Academic Success

nextChild=

class=mw-headline

id=Academic_Success

innerhtml=History

nextChild=

class=mw-headline

id=History
```

test-pattern-css13.hk

```
BEGIN{}
[@ .graham-dash @]{
}

END{print(5);}
```

test-pattern-css13.out

```
5
```

test-pattern-css2.hk

```
BEGIN{}
[@#catlinks@]{
  print(this);
}
END{}
```

test-pattern-css2.out

```
innerhtml=<div id="mw-normal-catlinks">

 <a href="/Special:Categories" title="Special:Categories">Category</a>:

 <ul>

  <li><a href="/Category:Miscellaneous_people" title="Category:Miscellaneous
people">Miscellaneous people</a></li>

 </ul>

</div>

nextChild=mw-normal-catlinks

class=catlinks

id=catlinks
```

test-pattern-css3.hk

```
BEGIN{}

[@p@]{

  print(this);

}

END{}
```

test-pattern-css3.out

innerhtml=Pokemon Type: Wood

nextChild=

id=

innerhtml=Known as one of the most devilishly attractive Indian males, he has been known to ensnare women with his long lashes. Ways to repel this predator: Weak Against: Clean Laundry, Soap, Detergent Strong Against: Dirty clothes, garbage, rotten food

nextChild=

id=

innerhtml=Standing at 5'9, Srikar was ranked the #1 shooting guard in India. Known for a deadly jumpshot but absolutely no speed, stamina, or muscle, he was widely regarded as a shooting threat that would add greatly to Columbia's limited offensive skillset. However, after coming to Columbia to play on the basketball team, Srikar sadly found out that American basketball is drastically different from Indian basketball. He sadly couldn't even make the girls intramural team. After a year of dedication and training, he was allowed to play with the Columbia Under-12 basketball team.

nextChild=

id=

innerhtml=A prestigious Rabi Scholar, Srikar attempted to take 9 classes but was ultimately shut down by Dean Simon Bird. Often found in the Hartley Computer lab, Srikar is known for working furiously into the night and disappearing during the day.

nextChild=

id=

innerhtml=Known as the world's smartest chocolate labrador, Srikar was born in India, lived in India, and raised in India. Fell in love with a whale but as we know a dog and whale love story is bound to end in failure.

nextChild=

id=

test-pattern-css4.hk

```
BEGIN{}

[@div p@]{

  print(this);

}

END{}
```

test-pattern-css4.out

innerhtml=Pokemon Type: Wood

nextChild=

id=

innerhtml=Known as one of the most devilishly attractive Indian males, he has been known to ensnare women with his long lashes. Ways to repel this predator: Weak Against: Clean Laundry, Soap, Detergent Strong Against: Dirty clothes, garbage, rotten food

nextChild=

id=

innerhtml=Standing at 5'9, Srikar was ranked the #1 shooting guard in India. Known for a deadly jumpshot but absolutely no speed, stamina, or muscle, he was widely regarded as a shooting threat that would add greatly to Columbia's limited offensive skillset. However, after coming to Columbia to play on the basketball team, Srikar sadly found out that American basketball is drastically different from Indian basketball. He sadly couldn't even make the girls intramural team. After a year of dedication and training, he was allowed to play with the Columbia Under-12 basketball team.

nextChild=

id=

innerhtml=A prestigious Rabi Scholar, Srikar attempted to take 9 classes but was ultimately shut down by Dean Simon Bird. Often found in the Hartley Computer lab, Srikar is known for working furiously into the night and disappearing during the day.

nextChild=

id=

innerhtml=Known as the world's smartest chocolate labrador, Srikar was born in India, lived in India, and raised in India. Fell in love with a whale but as we know a dog and whale love story is bound to end in failure.

nextChild=

id=

test-pattern-css5.hk

```
BEGIN{}

[@div > p@]{

  print(this);

}

END{}
```

test-pattern-css5.out

innerhtml=Pokemon Type: Wood

nextChild=

id=

innerhtml=Known as one of the most devilishly attractive Indian males, he has been known to ensnare women with his long lashes. Ways to repel this predator: Weak Against: Clean Laundry, Soap, Detergent Strong Against: Dirty clothes, garbage, rotten food

nextChild=

id=

innerhtml=Standing at 5'9, Srikar was ranked the #1 shooting guard in India. Known for a deadly jumpshot but absolutely no speed, stamina, or muscle, he was widely regarded as a shooting threat that would add greatly to Columbia's limited offensive skillset. However, after coming to Columbia to play on the basketball team, Srikar sadly found out that American basketball is drastically different from Indian basketball. He sadly couldn't even make the girls intramural team. After a year of dedication and training, he was allowed to play with the Columbia Under-12 basketball team.

nextChild=

id=

innerhtml=A prestigious Rabi Scholar, Srikar attempted to take 9 classes but was ultimately shut down by Dean Simon Bird. Often found in the Hartley Computer lab, Srikar is known for working furiously into the night and disappearing during the day.

nextChild=

id=

innerhtml=Known as the world's smartest chocolate labrador, Srikar was born in India, lived in India, and raised in India. Fell in love with a whale but as we know a dog and whale love story is bound to end in failure.

nextChild=

id=

test-pattern-css6.hk

```
BEGIN{}

[@div ~ p@]{

  print(this);

}

END{}
```

test-pattern-css6.out

innerhtml=Pokemon Type: Wood

nextChild=

id=

innerhtml=Known as one of the most devilishly attractive Indian males, he has been known to ensnare women with his long lashes. Ways to repel this predator: Weak Against: Clean Laundry, Soap, Detergent Strong Against: Dirty clothes, garbage, rotten food

nextChild=

id=

innerhtml=Standing at 5'9, Srikar was ranked the #1 shooting guard in India. Known for a deadly jumpshot but absolutely no speed, stamina, or muscle, he was widely regarded as a shooting threat that would add greatly to Columbia's limited offensive skillset. However, after coming to Columbia to play on the basketball team, Srikar sadly found out that American basketball is drastically different from Indian basketball. He sadly couldn't even make the girls intramural team. After a year of dedication and training, he was allowed to play with the Columbia Under-12 basketball team.

nextChild=

id=

innerhtml=A prestigious Rabi Scholar, Srikar attempted to take 9 classes but was ultimately shut down by Dean Simon Bird. Often found in the Hartley Computer lab, Srikar is known for working furiously into the night and disappearing during the day.

nextChild=

id=

innerhtml=Known as the world's smartest chocolate labrador, Srikar was born in India, lived in India, and raised in India. Fell in love with a whale but as we know a dog and whale love story is bound to end in failure.

nextChild=

id=

test-pattern-css7.hk

```
BEGIN{}

[@li + li@]{

  print(this);

}

END{}
```

test-pattern-css7.out

```
innerhtml=<a href="http://www.mediawiki.org/"><img
src="/skins/common/images/poweredby_mediawiki_88x31.png" alt="Powered by
MediaWiki" width="88" height="31"></a>

nextChild=

id=footer-poweredbyico

innerhtml=<a href="/WikiCU:General_disclaimer" title="WikiCU:General
disclaimer">Disclaimers</a>

nextChild=

id=footer-places-disclaimer

innerhtml=<a href="/WikiCU:About" title="WikiCU:About">About WikiCU</a>

nextChild=

id=footer-places-about

innerhtml=Content is available under <a class="external"
href="http://www.gnu.org/copyleft/fdl.html">GNU Free Documentation License 1.3
or later</a>.

nextChild=

id=footer-info-copyright
```

innerhtml=This page has been accessed 157 times.

nextChild=

id=footer-info-viewcount

innerhtml=<a href="/index.php5?title=Srikar_Varadaraj&oldid=54050"
title="Permanent link to this revision of the page">Permanent link

nextChild=

id=t-permalink

innerhtml=<a href="/index.php5?title=Srikar_Varadaraj&printable=yes"
rel="alternate">Printable version

nextChild=

id=

innerhtml=<a href="/Special:SpecialPages" title="A list of all special pages
[q]" accesskey="q">Special pages

nextChild=

id=t-specialpages

innerhtml=<a href="/Special:RecentChangesLinked/Srikar_Varadaraj" title="Recent
changes in pages linked from this page [k]" accesskey="k">Related changes

nextChild=

id=t-recentchangeslinked

innerhtml=<a href="http://www.wikicu.com/Help:Contents" rel="nofollow"
title="The place to find out">Help

nextChild=

id=n-Help

innerhtml=Random article

nextChild=

id=n-Random-article

```
innerhtml=<a href="/Special:RecentChanges">Recent changes</a>

nextChild=

id=n-Recent-changes

innerhtml=<a href="/WikiCU:Community_portal">Community portal</a>

nextChild=

id=n-Community-portal

innerhtml=<span><a href="/index.php5?title=Srikar_Varadaraj&amp;action=history"
title="Past revisions of this page [h]" accesskey="h">View history</a></span>

nextChild=

class=collapsible

id=ca-history

innerhtml=<span><a href="/index.php5?title=Srikar_Varadaraj&amp;action=edit"
title="This page is protected.

You can view its source [e]" accesskey="e">View source</a></span>

nextChild=

id=ca-viewsource

innerhtml=<span><a href="/Talk:Srikar_Varadaraj" title="Discussion about the
content page [t]" accesskey="t">Discussion</a></span>

nextChild=

id=ca-talk

innerhtml=<a href="#Attractiveness"><span class="tocnumber">5</span> <span
class="toctext">Attractiveness</span></a>

nextChild=

class=toclevel-1 tocsection-5

id=
```

```
innerhtml=<a href="#Mating_Calls"><span class="tocnumber">4</span> <span
class="toctext">Mating Calls</span></a>

nextChild=

class=toclevel-1 tocsection-4

id=

innerhtml=<a href="#Basketball_Scouting_Report"><span class="tocnumber">3</span>
<span class="toctext">Basketball Scouting Report</span></a>

nextChild=

class=toclevel-1 tocsection-3

id=

innerhtml=<a href="#Academic_Success"><span class="tocnumber">2</span> <span
class="toctext">Academic Success</span></a>

nextChild=

class=toclevel-1 tocsection-2

id=
```

test-pattern-css8.hk

```
BEGIN{}

[@a[href*="http://www.wikicu.com/Main_Page"]@]{

    print(this);

}

END{}
```

test-pattern-css8.out

```
rel=nofollow

href=http://www.wikicu.com/Main_Page

innerhtml=Main page

nextChild=

id=
```

test-pattern-css9.hk

```
BEGIN{}

[@a[href$="http://www.wikicu.com/Main_Page"]@]{

    print(this);

}

END{}
```

test-pattern-css9.out

```
rel=nofollow

href=http://www.wikicu.com/Main_Page

innerhtml=Main page

nextChild=

id=
```

test-pattern-regex1.hk

```
BEGIN{}
[/c/]{print("Found c");}
END{}
```

test-pattern-regex1.out

```
Found c
Found c
Found c
Found c
Found c
Found c
Found c
Found c
Found c
...
```

test-pattern-regex10.hk

```
BEGIN{}
[/[a]hey/]{print("Found [a]hey");}
END{}
```

test-pattern-regex10.out

```
Found [a]hey
```

test-pattern-regex11.hk

```
BEGIN{
  a = 0;
}
[/[a]|hey/]{
  a = 6;
}
END{
  print(a);
}
```

test-pattern-regex11.out

```
6
```

test-pattern-regex2.hk

```
BEGIN{}
[/~/]{print("Found ~");}
END{}
```

test-pattern-regex2.out

```
Found ~
```

test-pattern-regex3.hk

```
BEGIN{
a = 0;
}
[/./]{
  a = 6;
}
END{
  print(a);
}
```

test-pattern-regex3.out

```
6
```

test-pattern-regex4.hk

```
BEGIN{
  a = 8;
}
[/[a-z]/]{
  a = 9;
}
END{
  print(a);
}
```

test-pattern-regex4.out

```
9
```

test-pattern-regex5.hk

```
BEGIN{
  a = 0;
}
[/[az]/]{
  a = 9;
}
END{
  print(a);
}
```

test-pattern-regex5.out

```
9
```

test-pattern-regex6.hk

```
BEGIN{
  a = 8;
}
[/[^a]/]{
  a = 9;
}
END{
  print(a);
}
```

test-pattern-regex6.out

```
9
```

test-pattern-regex7.hk

```
BEGIN{}
[/(hello)where/]{
  print("Found hello where");
}
END{}
```

test-pattern-regex7.out

```
Found hello where
```

test-pattern-regex8.hk

```
BEGIN{}
[/[a]+/]{
  print("Found seq of a's");
}
END{}
```

test-pattern-regex8.out

```
Found seq of a's

Found seq of a's

Found seq of a's

Found seq of a's

Found seq of a's

Found seq of a's

Found seq of a's

Found seq of a's

Found seq of a's

Found seq of a's

Found seq of a's

...
```

test-pattern-regex9.hk

```
BEGIN{
  a = 0;
}
[/a[a]?/]{
  a = 6;
}
END{
  print(a);
}
```

test-pattern-regex9.out

```
6
```

test-tbl1.hk

```
BEGIN {
    t = {5};
    print(t[0]);
}
END {}
```

test-tbl1.out

```
5
```

test-tbl10.hk

```
BEGIN {
    t = {3};
    t[0] = 4;
    t[1] = t[0];
    print(t[1]);
}
END {}
```

test-tbl10.out

```
4
```

test-tbl2.hk

```
BEGIN {
    t = {};
    t[0] = 5;
    print(t[0]);
}
END {}
```

test-tbl2.out

```
5
```

test-tbl3.hk

```
BEGIN {
    t = {};
    t[0] = 5;
    s = t;
    print(s[0]);
}
END {}
```

test-tbl3.out

```
5
```

test-tbl4.hk

```
BEGIN {
    t = {};
    t[0] = "dog";
    t[1] = "cat";
    t["fish"] = t[0]+t[1];
    print(t["fish"]);
}
END {}
```

test-tbl4.out

```
dogcat
```

test-tbl5.hk

```
BEGIN {
    t = {};
    t[0] = 5;
    t = {};
    t[0] = 6;
    print(t[0]);
}
END {}
```

test-tbl5.out

```
6
```

test-tbl6.hk

```
BEGIN {
    t = {"dog":"cat"+"mouse"};
    print(t["dog"]);
}
END {}
```

test-tbl6.out

```
catmouse
```

test-tbl7.hk

```
BEGIN {
    t = {};
    t[0] = {{"dog":3}};
    print(t[0][0]["dog"]);
}
END {}
```

test-tbl7.out

```
3
```

test-tbl8.hk

```
BEGIN {
    t = {};
    t[0] = {3};
    t[1] = {4,5};
    print(t[0][0] + t[1][0]+t[1][1]);
}
END {}
```

test-tbl8.out

```
12
```

test-tbl9.hk

```
BEGIN {
    t = {1:{2:{}}};
    t[1][2][3] = {4:5};
    print(t[1][2][3][4]);
}
END {}
```

test-tbl9.out

```
5
```

testinput.txt

```
ccccc
ahey~
89
hellowhere
aa
```

8.10 Miscellaneous Code

8.10.1 util.ml

```
(*Closed-open range from a to b, e.g. range 1 5 = [1;2;3;4] *)

let rec range a b =

    if a=b-1 then

        [a]

    else

        a::(range (a+1) b)



let have_duplicates compare lst =

    let sorted = List.sort compare lst in

    match sorted with

        [] -> false

        | hd::tl -> fst (List.fold_left (fun (b,last) next -> (b ||
last=next),next) (false,hd) tl)



(*Are all the elements of a list the same? *)

let all_the_same = function

    | [] -> true

    | lst ->

        (let hd = (List.hd lst) in

        List.for_all ((=) hd) lst)



let wrap_id str = "_" ^ str ^ "_"

let unwrap_id s = String.sub s 1 ( (String.length s) - 2)
```

8.10.2 prettyprint.ml

```
open Ast


let rec repeat str n =

    match str,n with

        |s,1 -> s

        |s,n -> s ^ (repeat str (n-1))


let string_for_indent indent =

    repeat "\t" indent


let rec string_of_regex_set = function

    RegexStringSet(str) -> str

    | RegexRangeSet(str1, str2) -> str1 ^ "-" ^ str2

    | RegexComplementSet(set) -> "^" ^ (string_of_regex_set set)

    | RegexNestedSet(set) -> (string_of_regex_set_sequence set)


and string_of_regex_set_sequence seq =

    let all_together = String.concat " " (List.map string_of_regex_set seq) in

    "[" ^ all_together ^ "]"


let string_of_regex_op op =

    match op with
```

```
        | Or -> "|"

        | Optional -> "?"

        | KleenePlus -> "+"

        | KleeneTimes -> "*"



let rec string_of_regex regex = match regex with

  RegexString(str) -> str

  | RegexAnyChar -> "."

  | RegexNested(sequence) -> "(" ^ (string_of_regex_sequence sequence) ^ ")"

  | RegexSet(sequence) -> string_of_regex_set_sequence sequence

  | RegexUnOp(re,op) -> (string_of_regex re) ^ (string_of_regex_op op)

  | RegexBinOp(re1,op,re2) -> (string_of_regex re1) ^ (string_of_regex_op op) ^
(string_of_regex re1)



and string_of_regex_sequence seq = String.concat "" (List.map string_of_regex
seq)




let string_of_property prop = match prop with

    |ClassMatch(s) -> "." ^ s

    |IdMatch(s) -> "#" ^s

    | AttributeExists(str) -> "[" ^ str ^ "]"

    | AttributeEquals(attr,str) -> "[" ^ attr ^ "=" ^ str ^ "]"

    | AttributeContains(attr,str) -> "[" ^ attr ^ "*=" ^ str ^ "]"
```

```
    | AttributeBeginsWith(attr,str) -> "[" ^ attr ^ "^=" ^ str ^ "]"

    | AttributeEndsWith(attr,str) -> "[" ^ attr ^ "$=" ^ str ^ "]"

    | AttributeWhitespaceContains(attr,str) -> "[" ^ attr ^ "~=" ^ str ^ "]"



let string_of_type_selector = function

    | Elt(str) -> str

    | Universal -> "*"

    | NoType -> ""



let string_of_simple_selector_seq (type_selector,props) =

    let prop_string = List.fold_left (fun acc prop -> acc ^ (string_of_property
prop)) "" props in

    (string_of_type_selector type_selector) ^ prop_string



let string_of_combinator = function

    | DirectChild -> " "

    | Descendent -> ">"

    | DirectSibling -> "+"

    | AnySibling -> "~"



let rec string_of_css_selector css_selector = match css_selector with

    | SingleSelector(seq) -> (string_of_simple_selector_seq seq)

    | ChainedSelectors(selector,comb,seq) -> (string_of_css_selector selector) ^
(string_of_combinator comb) ^ (string_of_simple_selector_seq seq)
```

```
let string_of_op = function
    Plus ->  " + "
    | Minus ->  " - "
    | Times ->  " * "
    | Divides ->  " / "
    | Equal ->  " = "
    | Mod -> "%"
    | NotEqual ->  " != "
    | Less ->  " < "
    | LessEqual ->  " <= "
    | Greater ->  " > "
    | GreaterEqual ->  " >= "
    | BAnd -> " && "
    | BOr -> " || "

let string_of_key_literal = function
    IntKey(key) -> string_of_int key
    | StringKey(key) -> key

(*TODO: these don't need to all be mutually recursive*)
let rec string_of_table_literal table_lit =
    let inner_part = match table_lit with
```

```
        EmptyTable -> "{}"

        | ArrayLiteral(expr_list) ->  (String.concat "," (List.map
string_of_expr expr_list))

        | KeyValueLiteral(keyval_list) -> (String.concat "," (List.map
string_of_keyval_literal keyval_list))

    in "{" ^ inner_part ^ "}"

and

string_of_literal = function

    IntLiteral(x) -> string_of_int x

    | StringLiteral(str) -> str

    | DoubleLiteral(dbl) -> string_of_float dbl

    | This -> "This"



and

string_of_keyval_literal (key,v) =

    (string_of_key_literal key) ^ ":" ^ (string_of_expr v)

and string_of_expr_list = function

    [] -> ""

    | [hd] -> string_of_expr hd

    | hd::tl -> (string_of_expr hd) ^ ", " ^ string_of_expr_list tl

and

string_of_table_indices indices =

    let e_strings =  (List.map (fun e -> "[" ^ (string_of_expr e) ^ "]")
indices) in

    (String.concat "" e_strings)
```

```
and

string_of_expr = function

    Id(id) -> id

    | Literal(lit) -> string_of_literal lit

    | TableLiteral(tbl_lit) -> string_of_table_literal tbl_lit

    | Assign(id, expr) -> id ^ " = " ^ (string_of_expr expr)

    | Binop(expr1, op, expr2) -> (string_of_expr expr1) ^ (string_of_op op) ^
(string_of_expr expr2)

    | Uminus(expr) -> "-" ^ (string_of_expr expr)

    | Call(id, expr_list) -> id ^ "(" ^ string_of_expr_list expr_list ^ ")"

    | TableAccess(t, e_list) ->

        let bracket_part = string_of_table_indices e_list in

        t ^ bracket_part

    | TableAssign(table_id, e_list, e_assignee) ->

        let bracket_part = string_of_table_indices e_list in

        table_id ^ bracket_part ^ "=" ^ (string_of_expr e_assignee)




let rec string_of_func_decl func_decl  =

    func_decl.fname ^ "(" ^ (String.concat "," func_decl.params) ^ ")" ^
(string_of_stmt_list func_decl.body)

and

string_of_stmt_list = function

    [] -> ""

    | hd::tl -> (string_of_stmt hd) ^ "\n" ^ (string_of_stmt_list tl)
```

```
and string_of_stmt = function

    Block(stmt_list) -> "{\n" ^ (string_of_stmt_list stmt_list) ^ "\n}"

    | Expr(expr) -> (string_of_expr expr) ^ ";"

    | Func(func_decl) -> string_of_func_decl func_decl

    | Return(expr) -> "Return " ^ (string_of_expr expr) ^ ";"

    | If(expr, stmt1, stmt2) -> "if(" ^ (string_of_expr expr) ^ ")" ^
(string_of_stmt stmt1) ^ "else" ^ (string_of_stmt stmt2)

    | While(expr, stmt) -> "while(" ^ (string_of_expr expr) ^ ")" ^
(string_of_stmt stmt)

    | For(str1, str2, stmt) -> "for(" ^ str1 ^ " in " ^ str2 ^ ")" ^
(string_of_stmt stmt)

    | Empty -> ""



let string_of_pattern pat =

    let inner_pat = match pat with

        CssPattern(css_selector) -> "@" ^ (string_of_css_selector css_selector)
^ "@"

        | RegexPattern(regex_seq) -> "/" ^ (string_of_regex_sequence regex_seq)
^ "/"

    in "[" ^ inner_pat ^ "]"



let string_of_pattern_action (pattern,action) =

    (string_of_pattern pattern) ^ (string_of_stmt action)



let string_of_program prog =

    "BEGIN " ^ (string_of_stmt prog.begin_stmt) ^ "\n"
```

```
    ^ (String.concat "\n" (List.map string_of_pattern_action
prog.pattern_actions)) ^"\n"

    ^ "END" ^ (string_of_stmt prog.end_stmt)
```

8.10.3 Setup.java

```
_HAWKFileReader _fileReader = new _HAWKFileReader(_args);

_HAWKRegexMatcher _regexMatcher = new
_HAWKRegexMatcher(_fileReader._getConcatFile());

_HAWKCSSMatcher _cssMatcher = new _HAWKCSSMatcher(_fileReader._getConcatFile());
```

8.10.4 Import.java

```
import _hawk_lib.*;
```

8.10.5 Demos

demo1.hk

```
BEGIN{

  primes = {};

  fun toInt(s){

    n = {};

    index = 0;

    val = 0;

    isChar = 0;
```

```
if(length(s) == 0){
  return -1;
}
while(index < length(s)){
 if(stringEqual(charAt(s, index),"0")){
   val = val + 0;
 }else{isChar = isChar + 1;}
 if(stringEqual(charAt(s, index),"1")){
   val = val + 1;
 }else{isChar = isChar + 1;}
 if(stringEqual(charAt(s, index),"2")){
   val = val + 2;
 }else{isChar = isChar + 1;}
 if(stringEqual(charAt(s, index),"3")){
   val = val +  3;
 }else{isChar = isChar + 1;}
 if(stringEqual(charAt(s, index),"4")){
   val = val + 4;
 }else{isChar = isChar + 1;}
 if(stringEqual(charAt(s, index),"5")){
   val = val + 5;
 }else{isChar = isChar + 1;}
 if(stringEqual(charAt(s, index),"6")){
   val = val + 6;
```

```
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"7")){
      val = val + 7;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"8")){
      val = val + 8;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"9")){
      val = val + 9;
    }else{isChar = isChar + 1;}
    if(isChar >= 10){
      if(index == 0){
        return -1;
      }else{
        return val/10;
      }
    }
    val = val * 10;
    index = index + 1;
    isChar = 0;
  }
  return val/10;
}
```

```
  fun isPrime(i){
    if(i <= 1){
      return 0;
    }
    if(i <= 3){
      return 1;
    }
    if(i % 2 == 0 || i % 3 == 0){
      return 0;
    }
    n = 5;
    while(n*n <= i){
        if (i % n == 0){
            return 0;
        }
        n = n + 1;
    }
    return 1;
  }
}
[/[0-9]+/]{
  if(exists(this[0])){
    val = toInt(this[0]);
    if(val == -1){
```

```
    }else{
      if(isPrime(val)){
        primes[val] = 1;
      }
    }
  }
}
END{
  print(keys(primes));
  print("Number of Primes: "+ length(primes));
}
```

demo1.sh

```
./prog ./demos/demo1.hk > Program.java && javac Program.java && java -cp
./_hawk_lib/jsoup.jar: Program ./demos/regexdemo.txt
```

demo2.hk

```
BEGIN{
avg = 0.0;
ctr = 0;
fun toInt(s){
    n = {};
    index = 0;
```

```
val = 0;
isChar = 0;
if(length(s) == 0){
    return -1;
}
while(index < length(s)){
    if(stringEqual(charAt(s, index),"0")){
     val = val + 0;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"1")){
     val = val + 1;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"2")){
     val = val + 2;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"3")){
     val = val +  3;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"4")){
     val = val + 4;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"5")){
     val = val + 5;
```

```
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"6")){
     val = val + 6;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"7")){
     val = val + 7;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"8")){
     val = val + 8;
    }else{isChar = isChar + 1;}
    if(stringEqual(charAt(s, index),"9")){
     val = val + 9;
    }else{isChar = isChar + 1;}
    if(isChar >= 10){
     if(index == 0){
       return -1;
     }else{
       return val/10;
     }
    }
    val = val * 10;
    index = index + 1;
    isChar = 0;
}
```

```
    return val/10;
}
}
[@.score .unvoted@]{
    votes = toInt(this["innerhtml"]);
    if(votes >= 0){
        avg = avg + votes;
        ctr = ctr + 1;
    }
}
END{
    avg = avg/ctr;
    print(avg);
}
```

demo2.sh

```
./prog ./demos/demo2.hk > Program.java && javac Program.java && java -cp
./_hawk_lib/jsoup.jar: Program http://www.reddit.com
```

demo3.hk

```
BEGIN{
  fun getSub(s,c){
    i = 0;
```

```
    sub = "";
    while(i < length(s)){
      char = charAt(s, i);
      if(stringEqual(char, c)){
        return sub;
      }
      sub = sub + char;
      i = i + 1;
    }
    return sub;
  }
}
[@.city-nav-header@]{
  print(getSub(this["innerhtml"],"<"));
}
[@#curTemp > span > .wx-value@]{
  print("Temperature: "+this["innerhtml"]+"F");
}
[@#curCond > span@]{
  print("Current Conditions: "+this["innerhtml"]);
}
END{}
```

demo3.sh

```
args="$*"

q=${args// /'%20'}

./prog ./demos/demo3.hk > Program.java && javac Program.java && java -cp
./_hawk_lib/jsoup.jar: Program http://www.wunderground.com/cgi-
bin/findweather/getForecast?query=$q
```

demo4.hk

```
BEGIN{

  lineCount = 0;

}

[/[\n]/]{

  lineCount = lineCount + 1;

}

END{

  print(lineCount);

}
```

demo4.sh

```
./prog ./demos/demo4.hk > Program.java && javac Program.java && java -cp
./_hawk_lib/jsoup.jar: Program $*
```

demo5.hk

```
BEGIN{

    fun getSingletonTable(x){
        return {x};
    }

    fun modifyTable(t){
        t[0] = getSingletonTable(42);
    }

    table = {};
    table[0] = {};
    nestedTable = table[0];

    modifyTable(nestedTable);

    nestedNestedTable = table[0][0];
    value = nestedNestedTable[0];
    print(value);

}

END{
```

```
}
```

demo5.sh

```
./prog ./demos/demo5.hk > Program.java && javac Program.java && java -cp
./_hawk_lib/jsoup.jar: Program $*
```

regexdemo.txt

```
1

2

3

4

5

6

7

8

9

10
```