Programming Languages and Translators

COMS W4115

Prof. Stephen Edwards
Summer 2008

POSTaL

Part-of-speech Tagging Language

Language Reference Manual

Peter Nalyvayko
np2240@columbia.edu

Table of Contents

Introduction

The manual describes the language POSTaL which stands for the "**P**art-**O**f-**S**peech **Ta**gging **L**anguage". The language design combines the imperative and functional programming constructs to perform the natural language processing tasks, including but not limited to the POS tagging.

Motivation

Part-of-speech tagging is an inseparable part of the natural language processing. Without going into details about the areas of NLP the POS tagging can be used, suffice it to say that properly assigning the tags to the parts of the sentence is a step towards understanding the meaning of the sentence.
There are a few well-known NLP toolkits, among which the lingpipe (used by the POSTaL language internally for NLP specific tasks such as tagging and sentence tokenization). They target high-level imperative languages such as Java, C++, C#. The intention of the POSTaL is not to provide another toolkit, but to simplify the infrastructure and the analysis phases by offering a simple programming paradigm for performing the tagging and analysis of the tagged output while hiding the complexity of the actual POS tagging implementation.

Overview

The POSTAL is a typeless scripting language. The language constructs are best suited for list manipulation. The primary use for the language is to perform string tokenization and the part-of-speech tagging using the language's built-in facilities. The language grammar is a mixture of imperative and functional styles. While it borrows heavily from LISP for list-specific operations, the language also tries to make it easy to code the program work-flow using constructs familiar to those who are used to code in imperative languages such as C++, Java and C#.

Goals

Natural Language Processing

The language relies on the third-party API "LingPipe" for all the NLP-specific tasks. The lingpipe is a Java library for linguistic analysis of human language. Among other features, the lingpipe offers statistical models and classes to perform part-of-speech tagging and sentence tokenization.

Tokenization

The part-of-speech tagging depends on the properly tokenized sentence. The language provides built-in facilities to perform sentence chunking and tokenization.

Part-of-speech Tagging

The Part-Of-Speech tagging (POS tagging) facilities offered by the language make use of the statistical tagger based on the Hidden Markov Models. The models are a part of the language package so the users need not worry about the infrastructure or training the models which by itself is a complicated task involving training models and a training corpora (in our case, the brown corpora). The language users can perform the POS tagging having a little or no prior experience with NLP.

Ease-of-use

The language provides notational clarity somewhat lacking in functional languages intermixed with well-established syntactical constructs borrowed from the imperative languages making it easy to code and to read the program workflow.

Language Tutorial

Hello, world!

The simpliest way to get acquainted with a language is to try writing a program. The following is a simple program that outputs a “Hello, world” string to the console. Using your favorite editor, type the following program and save it into the file 'sample1.postal'.

```
function main()
{
	println(“Hello, world!”);
}
```

“function main()”

This is a function declaration. There must always be a function **main** declared in any program written in the POSTAL language. The function is used by the translator as an entry point which is invoked after the lexical, syntactical and semantic analysis are completed.

println(“Hello, world!”);

This is a statement. A statement can be a simple or a compound expression. The statement instructs the translator to invoke a built-in function **println** that prints its argument to the standard output.

Functions can optionally return a result back to the caller, however there is no need to return anything in this case because the translator ignores the results returned by the function “main”.

Running the program

Once the program is saved into a file, it can be executed by running the following command in a terminal window:

> java -jar postal.jar sample1.postal

Declare a user-defined function

The language allow declaring subroutines that are made up of statements. Such subroutines are called the user-defined functions. The function declaration starts with the **function** keyword followed by the symbolic identifier and a pair of left-right parentheses. A list of parameters is declared inside the parentheses. Parameters are optional. Functions can only be declared on the global scope. Here is a typical function declaration:

```
function gcd(a, b)
{
	println(a);
```

```
        println(b);
}
```

The above code declares a function "*gcd*" taking two arguments *a* and *b* which are printed to the standard output console using the built-in function "println".
Invoking the user-defined function is as trivial is in most imperative languages:

```
function main()
{
        gcd(1, 2);
}
```

Anonymous functions (Lambda expressions)

At the heart of the POSTaL language lies its ability to deal with the anonymous functions sometimes also referred to as 'lambda expression'. A lambda expression is declared using the keyword **lambda** followed by a pair of parentheses () enclosing the lambda expression's parameter list. The following code demonstrates a typical declaration of a lambda expression:

```
var f ;
f = lambda(x) { println(x); };
```

The above code declares an uninitialized variable 'f' and assigns it a lambda expression. The lambda expression is not evaluated at the time of declaration and simply is a set of instructions which can be executed by the interpreter. To evaluate a lambda expression, the language provides a keyword **apply:**

```
var f ;
f = lambda(x) { println(x); };
apply(f, 1);
```

The **apply** takes a lambda expression as its first argument followed by the argument list. The arguments are evaluated and then are used to invoke the lambda expression. Few words must be said about the scoping and the lambda expressions. While the user-defined functions as well as built-in functions all reside at the global scope level, the lambda expressions reside at the level of the caller, allowing the lambda expressions to access the local variables declared within the caller's scope. The following code demonstrates the concept:

```
var b = 10;
function gcd(f)
{
        var b = 5;
        println("Locally defined b:" + b);
        println("A value returned by the lambda expression:" + apply(f));
}

function main()
{
        gcd(lambda() { return b; });
}
```

What is the output of the program? Correct, "5"!

Examples

The following section presents the implementations of several most popular algorithms that have been used in the functional languages such as Lisp. At the end of the section there is an example to

demonstrate the part-of-speech tagging functionality built into the language.

Implement a high-order function 'fold'

High-order 'fold' function takes a list and a lambda expression as an input, and 'folds' the results of the evaluating of the lambda expression in between the elements of the list.

Example:

Original list:

1	2	3	4	5

fold(list ('1', '2', '3', '4', '5'), '0', lambda(x) { return '+'; });

The result of the fold function:

0	+	1	+	2	+	3	+	4	+	5

```
// Fold takes a function and "folds" it in
// between the elements of the list.
function fold(l, f, init)
{
	var result = list();
	dolist(l, lambda(x) { cons(x, result); cons(apply(f, x), result); result =
tail(tail(result)); });
	return cons(init, head(result));
}
function main()
{
	fold(words(“2 3 4 5 6”), lambda(x) { return "+"; }, "1");
}
```

The output of the program is:

```
(1 + 2 + 3 + 4 + 5 + 6)
```

The line “function fold(l, f, init) declares a function that takes three parameters. The implementation of the function is not trivial so let's walk through the details of the fold function. The first line:

```
var result = list();
```

Is a declaration of a variable 'result' which is initialized to an empty list. The list() function is a built-in function that creates an empty list object. The next line is where the folding happens:

```
dolist(l, lambda(x) { ... });
```

It reads: “for each element in the list 'l' execute the lambda expression with a single argument. When invoking the lambda expression, the interpreter passes a current element of the list to it as the only argument, so it is important to remember that when using lambda expressions in conjuncture with the dolist statement, the lambda expression must always be declared to take a single argument to avoid run-time errors.

Let's move on to the implementation of the lambda expression used to perform the folding.

```
lambda(x)
```

```
{
        cons(x, result);
        cons(apply(f, x), result);
        result = tail(tail(result));
}
```

First statement “cons(x, result)” adds an element represented by the variable “x” to the list represented by the variable “result”. “cons” is built-in function used to add elements to an existing list. The element specified as the first argument to the “cons” function becomes the head of the list specified by the second argument. The next line “cons(apply(f, x), result)” invokes the function “f” passed as an argument to the “fold” function. The “apply” is a built-in function to invoke lambda expressions. The first argument to the “apply” is a lambda expression declared inline or a variable referencing a lambda expression. Following the lambda expression is a list of expressions used as arguments to invoke the lambda expression. The number of parameters must match the number of formal arguments of the aforementioned lambda expression. For example,

```
 apply(lambda(x, y) {println(“(” + x + “,” + y + “)”);}, 1, 2);
```

means “invoke the lambda expression declared as an anonymous function that takes two parameters. Pass '1' as the parameter 'x', and '2' as the parameter 'y'”. The output of the above fragment is :

```
                        (1,2)
```

The last statement in the lambda expression declaration:

```
result = tail(tail(result));
```

Simply shifts the head element pointer of the list “result” two elements forward. “tail” is a built-in function which takes a list as an input and returns a list following the first element of the specified list.

Implementing a high-order function “map”

“Map” function is a high-order function that takes a list and a function and applies the function to each element of the list. The result of the function is a new list.

Syntax

map (expression, lambda_expression | identifier)

```
// Map takes a list and a function and applies the function to each element of the list.
function map(l, f)
{
        var result = list();
        dolist(l,
                lambda(x)
                {
                        cons(apply(f, x), result);
                        result = tail(result);
                }
        );
        return head(result);
}
function main()
{
        print(map(words(“1 2 3 4 5 6”), lambda(x) { return “” + numeric(x) * 4;}));
}
```

The output of the program:

```
                              (4 8 12 16 20 24)
```

The function implementation is similar in spirit to that of the “fold” function. Notice the use of the numeric conversion utility “numeric”. “numeric” is a built-function which purpose is to convert the input into a numerical output.

Part-of-speech (POS) tagging example

The following example demonstrates how to perform the POS tagging using the language's built-in facilities and how the output of the tagging can be used to further analyze the input data.

```
function main()
{
	var sentence = words("Nascar's most popular driver announced that his future would not include
racing for Dale Earnhardt Inc.");
	var result = nbest(sentence);
	dolist(result,
		lambda(x) {
			if(length(x) > 1,
				apply(
				lambda(x)
					{
						if (nth(1, x) == "np",
							println(nth(0, x))
						);
					},
				x)
			)
		}
	);
}
```

The above program “POS-tags” the input sentence and then prints the proper nouns found in the “sentence”.
The first line declares the program's main entry point. The second line:

```
	var sentence = words("Nascar's most popular driver anno...”);
```

Declares a variable “sentence” and initializes it to a list of tokens. The “words” function takes a string as an input and tokenizes it by breaking the string onto individual tokens. The tokenization rules are preset to Indo-European dictionary and at this time cannot be changed by the user. The following is the list of tokens produced by the “words” function:

```
(Nascar ' s most popular driver announced that his future would not include racing for Dale Earnhardt
Inc .)
```

Notice the punctuation marks such as **'** and **.** have become independent tokens. Next step to do the tagging.

```
	var result = nbest(sentence);
```

“nbest” is a built-in function which does the actual part-of-speech tagging using pre-trained Hidden Markov Model along with the statistical classifier. The model is a part of the core of the language and was generated using the Brown Corpus as the training data. The output of the “nbest” function is shown below:

```
((Nascar np) (' ') (s vbz) (most ql) (popular jj) (driver nn) (announced vbd) (that cs) (his pp$)
(future nn) (would md) (not *) (include vb) (racing vbg) (for in) (Dale np) (Earnhardt np) (Inc np) (.
.))
```

The tagged list is a list of lists where the nested lists contain two elements: the original token and a part-of-speech tag associated with the token. See the appendix for the list of currently supported tags.
The remaining part of the program filters the tagged sentence and prints out only proper nouns. Notice the use of the built-function “nth” which provides a way to glance at a specific element of a list using

its index without iterating over the list. At last, the output of the entire program looks like this:

```
Nascar
Dale
Earnhardt
Inc
```

Lexical Conventions

Tokens

The following are supported classes of tokens (the text which compiler does not break down onto components): operators, identifiers, keywords, constants, string literals and punctuation characters. The available punctuation characters are square brackets “[]”, figure brackets “{}”, parentheses “()”, semi-colon “;”, comma “,” .

White-space

The white space characters such as space between words are excluded during tokenization.

Comments

Comments are similar to the C++ comments. There are two types of comments supported by the language: multi-line comments starting with /* and terminating with */ and single-line comments starting with //; the end-of-line is used as a terminator for the single line comments.

Keywords

The following table contains the keywords reserved by the language. Keywords cannot be used as identifier names:

dolist	function	if
lambda	nil	numeric
return	true	var
while	apply	

Identifiers

Identifiers are names used to declare and refer to the variables and functions in the program. Identifiers are created when a variable or a function are declared. For example,

Syntax
identifier:
nondigit
identifier nondigit
identifier digit

Allowable nondigits

```
'a'..'z', 'A'..'Z','_', '0'..'9'
```

Identifier must start with a non-digit or underscore. The grammar treats identifiers in a case-insensitive manner.

Constants

Constants are numbers, characters or character strings that can be used in the program. Constants are used to represent an integer, floating-point or character strings.

String literals

String literal is a sequence of characters enclosed in the double quotes. Double quotes or backslash '\' in a string literal are escaped with backslash '\' .

Punctuation

Punctuation is used to organize the parts of the program and for other purposes. Available punctuation characters are: '[]' '{ }' () ',' ';' '=' '>=' '<=' '>' '<' '!=' '==' '#' '||' '&&'

Types

The POSTaL is dynamically typed and the type checking is performed at run-time. Internally, the language defines two basic types: lists and atoms. To language provides a set of functions to perform the run-time type checking.

Variable declarations

'Declaration' specifies a new variable or function. Variables can be declared globally (program scope) or locally (function scope). Variables declared inside the function are only visible within that function. Variables that are declared outside of any function are said to be defined on the file scope which makes them accessible and visible to the entire program. To declare a variable use a keyword var followed by a symbolic identifier followed by optional variable initialization.
Example:

```
var a;
var b = a;
var c = 1;
var d = 10 * 3;
```

Function declarations

Functions are always declared on a global scope. Nested function declarations are not supported. Functions can be declared inline using the 'lambda-expression'. See discussion on lambda expression later in the document.

Scopes

Scope is a sequence of statements enclosed with '{}' (curly braces). There are compound statement scopes, function scopes and nested scopes. Compound statement scope is a scope that is composed of one or more statements. Function scope is also known as function definition or function body is always associated with a function declaration. The scopes within other scopes are nested scopes. The language supports a static scoping to make it easy to reason about the code. Consider the following code fragment:

```
var a = 2;
function gcd()
{
        println(“variable:” + a);
}
function main()
{
        var a = 10;
        gcd();
}
```

In the above code, the output will be “variable: 2” because the gcd function refers to the variable “a” declared in the top-level scope, and not to local variable declared in the calling function “main”.

L-values and R-values

L-values are expressions that can appear on the left side of the assignment operator. L-values can be variables or variables followed by a subscript operator. R-values are expressions that can appear on the right side of the assignment. Unless otherwise specified, all expressions qualify as r-values.

Program structure

The source code resides in a single file. Variables and functions declared outside the program file are not visible and not accessible to the code.

Expressions

Primary expressions

Primary expressions are literals and identifiers. Literal is a constant which can be a string, a character, an integer or a float-point number.

Postfix Expressions

Postfix expressions are primary expressions or expressions followed by one of the following operators:

Operator Name	**Operator Notation**
Function call operator	()

Function-call Results

The function is always evaluated into r-value.

Unary expressions

Unary operators are applied to one operand in the expression.

Operators

Operator Name	**Operator Notation**
Assignment operator	'='
Greater-or-equal operator	>=
Less-or-equal operator	<=
Greater-than operator	>
Less-than operator	<
Non-equal operator	!=
Equality operator	'=='
Multiplication operator	*
Division operator	/
Additive operator	+
Subtraction operator	-
Disjunction operator	&&
Conjunction operator	||

All binary operations are left-associative.

Relational operators '>', '<', '<=', '>=', '==', '!='

Relational operators can compare lists, numerical or string values. When a string value appear on the left side and the numerical value appears on the right side, the numerical value is converted into a string and then strings are compared. Lists and numerics cannot be compared. When comparing a string and a list, either a string gets tokenized into a list or a list gets concatenated to a string. The actual transform depends which side of the comparison the operands appear. Lastly, relational operators cannot be “chained”.

var x = 1 < 3 < 4; // parser error!

Additive operators '+' and '-'

Additive operators are defined for numerics and strings. Operators cannot be used with lists.

Multiplicative operators '*' and '/'

Multiplicative operators are defined for numeric literals only.

Assignment operator '='

Assignment operator stores the value on the right into the object on the left. The left operand is referred to as 'l-value' and can either be an identifier or an identifier followed by a subscript. The assignment operator returns a value assigned to the left operand.

Statements

Selection statements

The If Statement

Enables the conditional branching. The expressions enclosed in the square brackets '[]' represent an optional part of the statement.

ifStatement : “if” '(' expression ',' expression [',' expression])

Where the “expression” is evaluated to “true” or “nil”. If the value of the expression is evaluated to “true” then the statement1 is evaluated, otherwise the statement2 (if provided) is evaluated. The statement2 is optional and does not have to be specified.

Iteration Statements

The While statement

Executes the statement until the expression is evaluated to nil.

whileStatement : “while” '(' expression ',' expression ')'

The keywords 'break' and 'continue' may only appear with the statement body inside the while statement. The 'break' stop the execution of the inner-most while loop immediately. The 'continue' interrupts the current iteration of the inner-most loop and moves on to the next iteration.

The dolist Statement

Iterates the list specified by the expression and for each element of the list executes the lambda expression or invokes the function specified by its reference using the Function Reference Operator.

dolistStatement: “dolist” '(' expression ',' “lambda” '(' arglist ')' '{' statement '}' '|' identifier ')'

Built-in functions

The language defines several categories of built-in functions to support list manipulations, debugging

and IO operations.

The length function

Returns a length of the expression. The result is an integer value. If the expression cannot be evaluated into a list, the return value will be zero.

Syntax:

length (expression)

The list function

Creates an empty list. The result of the evaluation of the list function is an empty list.

Syntax:

list()

The hashtable function

Creates an empty hash table. The elements of the hash table are key-value pairs. The hash tables are used to perform quick lookups using the The result of the evaluation is an empty hash table.

Syntax:

hashtable()

The pop function

Removes the element of the top of the list or hash table and returns the first element of the list or the hash table.

Syntax:

pop(expression)

Where expression must resolve into a list object.

The tail function

Evaluates the expression, and, if the expression evaluates into a list, returns a list that follows immediately after the first element. The function returns an error if the expression does not represent a list. The function returns nil if the expression represents an empty list.

Syntax

tail (expression)

The nbest function

The nbest is pivotal to the language. The function takes a list of tokens and performs stochastic part-of-speech tagging using the Hidden Markov Model. The function returns a copy of the original list where

each element of the original list is expanded into a list with the first element representing the original token and the remaining elements are possible POS tags.

Syntax

nbest(expression)

Example

var list = words(“The movie starts at seven.”);

println(list); // prints ('The' 'movie' 'starts' 'at' 'seven' '.')

var copy = nbest(list);

println(copy);

The output of the above program:

((The dt) (movie nn) (starts vb) (at in) (seven cd) (. .))

The dup function

Returns a deep copy of r-value specified by the expression.

Syntax

dup(expression)

Example:

var original = words(“1 2 3 4”);

var reference = original; // ← the variable 'reference' refers to the variable 'original' without coping its contents

var copyof = dup(original); // ← the variable 'copyof' contains exact copy of the contents stored in the variable 'original'

The openf function

I/O operation. The function loads the file contents, automatically tokenize the loaded sentences and returns a list.

Syntax

openf (filepath)

filepath	- a name of the file to store the list contents

The savef function

I/O operation. Stores the contents of the list into a file using the specified name. The function creates a file if it does not exist.

Syntax

savef(list, filepath)

list	– a list object
filepath	- a name of the file to store the list contents

The nth function

Allows random access to the list elements.

Syntax

nth (index, list)

Where

index - is an numerical value; it must be less than the total number of elements in the list.
list – a list object.

The cons function

A fundamental function for list construction.

Syntax

cons(elem, list)

Example:

```
var l = words(“B C D E F”);
println (l); // prints (B C D E F)
println( cons(“A”, l) ); // prints (A B C D E F)
```

The print and println functions

Prints the input expression to the standard output. The println function prints the input and terminates the line.

Syntax

print (expression)

println (expression)

The head function

Rewinds the list to the first element. The list's current head can be changed by using the 'tail' function. The head function moves the pointer to the first list element back to its original location.

Syntax

head (list)

Example:

```
var l = words(“A B C D”);
```

```
l = tail(l);
println(l); // prints (B C D)
l  = head(l);
println(l); // prints (A B C D)
```

The isliteral function

Tests whether the specified expression reprents a string or a character.

Syntax

```
isliteral (expression)
```

The isnumeric function

Tests whether the specified expression represents a numeric value.

Syntax

```
isnumeric(expression)
```

Example

```
println (isnumeric(1)); // prints true
println(isnumeric("A")); // prints nil
```

The islist function

Tests whether the specified expression represents a list object.

Syntax

```
islist(expression)
```

The assert function

Evaluates the expression and generates an assertion if the evaluation returned false. The function can be used to detect incorrect or unexpected results when debugging the program.

Syntax

```
assert (expression)
```

The islambda function

Tests whether the specified expression is a lambda expression.

Syntax

```
islambda (expression)
```

The assoc function

Looks up the hashtable and returns a value associated with the given key.

Syntax

assoc(hash, key)

Example

```
var hash = hashtable();
println (assoc(hash, “key1”));
```

The sassoc function

Associates the given key and the value and adds the association to the specified hashtable.

Syntax

sassoc (hash, key, value)

Example

```
var hash = hashtable();
sassoc(hash, “key1”, 1);
sassoc(hash, “key2”, 2);
```

The words function

NLP specific function. Parses the input string into a list of tokens.

Syntax

words(string)

Example

```
var tokens = words (“A red fox jumped over the fence.”);
println (tokens); // prints '(A red fox jumped over the fence .)'
```

Project Plan

Because the design, implementation, testing and the final reporting were all done by a single person's team, the project planning was easier and at the same more difficult in certain aspects than when working in a team of several members. The project schedule has been set up to allocate most of the time to the grammar design and the implementation of the language translator. The testing has been an integral part of the development process and not a separate phase. This was a conscious decision drawn from the past experience. As the developing was nearing the final phase, more time was spent on the documentation and writing the unit tests.

Domain analysis phase

Investigation of current state of the linguistic analysis with respect to the part-of-the speech tagging. Several tagger algorithms have been reviewed and analyzed. NLP course taken last semester played a role in selecting the statistical tagging methods for part-of-speech tagging.

Requirement analysis

The description for the course project contained the requirements which constrained, and thus simplified, the choice of the language, development platform as well the requirements for the compiler generator toolkit ANTLR. Due to a limited amount of resources the decision was made to implement an interpreter instead of a full-blown compiler.

Project specification

During the phase the important items were identified. For NLP specific tasks, the lingpipe API was chosen as it provided the most comprehensive set of classes and interfaces to deal with the part-of-speech tagging. During this phase, the development environment specification was devised identifying the primary operating system, the language of choice, the time line and the source control system.

The development environment

- Linux/ubuntu operating system
- Java SDK 1.5
- Eclipse
- ANTLR plugin for Eclipse
- ANTLR Java API
- Perforce as a source control system

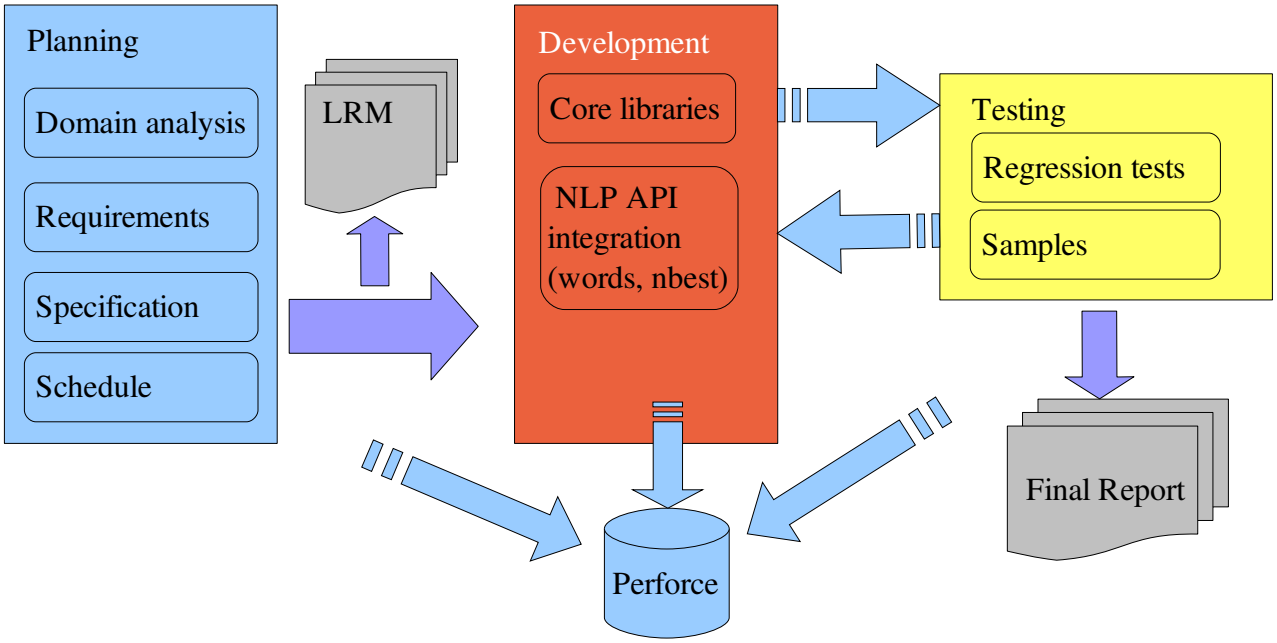
Development and testing phases

During the development phase, the core language libraries have been developed to support the language type system, control flow, function invocation and the integration with the NLP API. The testing phase has been an integral part of the development. The goal of the testing phase has been to product a suite

of regression tests to verify all major parts of the language: all types of supported declarations, consistency of the results produced by the arithmetic operations and comparison operations, control flow operations and NLP specific functionality.

Project development

The development was done in a way to ensure the timely development and testing of the core functionality. The bells and whistles of the language have been added later. The foremost importance was given to the grammar to ensure its consistency and ease-of-use. The unit tests provided an invaluable source of information enabling to discover the shortcomings of the initial design and helping to revise the original grammar and corresponding parts of the core libraries.



Testing

Testing was an integral part of the project planning. The continuous testing (continuous regression tests) was carried out during the development to verify the newly added features. As the development was mostly done, the testing phase moved into the next phase – examples and documentation. During that phase new unit tests were developed to implement well-known algorithms such as “fold”, “map” and “gcd”.

Programming style guidelines

Category	Description
Variable naming conventions	The language is case-sensitive, some choose the names carefully. The preferred naming convention is to use lower case for variable names. Variable names cannot start with numbers.
Function naming conventions	Use the lower camel case for function names.

	Method names cannot start with numbers.
Lambda expressions	Keep your lambda expression concise by keeping its definition small.
Function definitions	Use the 'return' statement to explicitly return a result from a user-defined function or a lambda expression. Omitting the return statement, while not an error, will lower the readability of the program. To improve reuse, keep your function footprint small.
Program structure	Do not put everything into the main function.
Assertions	Use the 'assert' built-in function to validate the results to ensure your program's proper behavior.

Project Time-line

Date	Description
06/04/2008	Language proposal
06/20/2008	Language reference manual
08/10/2008	Deployment strategy identified.
07/08/2008	Lexer and Parser complete
07/31/2008	Translator initial implementation
08/02/2008	Part-of-speech tagging added
08/03/2008	More built-in functions
08/08/2008	Translator complete.
08/10/2008	Final report submitted.

Project Log

The project log is a progress report generated by extracting the history of submissions to the perforce.

Date	Description
06/06/2008	Project has begun
06/08/2008	Development environment is configured
06/09/2008	Source control system is configured
06/10/2008	Initial draft of the lexer and parser
06/15/2008	Begun working on the language reference manual
06/18/2008	Language reference manual first draft

06/20/2008	LRM is submitted
06/26/2008	Begun working on the translator
07/13/2008	submitting the initial (cleaned-up) grammar, examples and documentation; the tree walker needs work; new operators are needed to implement sort, map and fold operations
07/27/2008	Classes to support compiler type system
08/02/2008	Implemented some built-in functions like nbest, words, pop, length, etc.; added the lingpipe specific functionality to do the actual POS tagging. Improvements to the unit test suite to provide a better error reporting by including the column and line numbers with the assertion.
08/06/2008	added more built-in functions (assert, openf, print); the "main" user-defined function without parameters is treated as an entry point and is automatically invoked once the program is parsed and the semantic analysis phase is complete. More regression tests.
08/10/2008	Lists are now assigned by reference, i.e. the list does not get copied when assigned to another list - to copy the list contents, the function "dup" shall be used; ability to report the line and column numbers when throwing an assert. Added unit tests to validate the new functionality.
08/10/2008	Compiled and 'jar'-ed the compiled translator to prepare for deployment.
08/11/2008	Submitted the final report.

Software development environment

The language was developed using Java and the Eclipse IDE. All supporting classes are written in Java.

Operating system

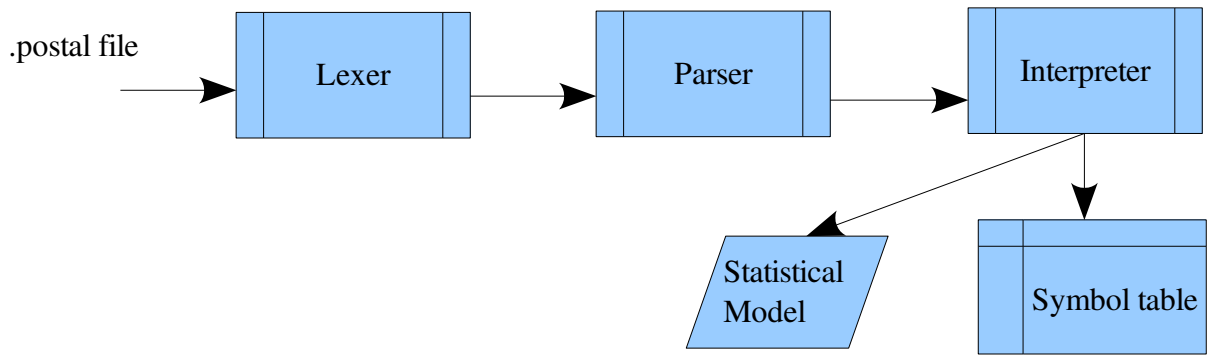
The development was done on the Linux/ubuntu codename “hardy-heron”. At times the Windows/cygwin and X-windows remoting was used to access the development machine. The source control system “Perforce” was locally installed on the development machine to provide the version control and progress tracking system. The following packages were used:

- Java 1.5
- ANTLR 2.7.6
- LINGPIPE-3.5.0

Architectural Design

Architecture

The POSTaL translator consists of several main blocks: lexer, parser, symbol tables, the program interpreter, run-time environment, a set of core libraries and the statistical model used to perform the POS tagging. The component diagram shows the blocks and their relationships:



- Lexer (Postal.g)
  - The role of the lexer component is to recognize the input and translate it into the sequence of tokens.
- Parser (POSTALParser.g)

The role of the parser is to take the sequence of tokens produced during the tokenization by the lexer component, perform the syntax analysis and check the grammar errors. The output of the parser is an abstract syntax tree (AST) representing the semantical relationships between the syntactical elements.

- Interpreter (POSTALWalker.g)

  The role of the interpreter is to perform the semantic analysis by walking the AST tree generated by the parser and to generate the output by evaluating the AST nodes. During the evaluation the semantics are checked and errors may be produced to indicate semantic inconsistencies.
- Symbol table
  - The symbol table keep track of variable, lambda expression and function declarations. The second very important role of the symbol table is to impose the scoping rules.

System File Diagram

The following list represents the directory structure of all files pertaining to the POSTaL language:

/

Main.java

MANIFEST

WS4115/Projects/Postal/AST:

AdvancedCommonAST.java

WS4115/Projects/Postal/brown:

HMMTagger.java

WS4115/Projects/Postal/builtin:

BuiltinFunctionProvider.java

TextFileHelper.java

WS4115/Projects/Postal/data:

pos-en-general-brown.HiddenMarkovModel

WS4115/Projects/Postal/examples:

algorithms.postal	arithmetics.postal
assignment.postal	builtin.postal
containers.postal	controlflow.postal
equal.postal	greaterorequal.postal
greaterthan.postal	lambda.postal
lessorequal.postal	lessthan.postal
notequal.postal	tagging.postal

WS4115/Projects/Postal/Grammar:

POSTAL.g	POSTALParser.g	POSTALWalker.g

WS4115/Projects/Postal/Types:

AdditionOperations.java	ANDOperations.java
AssertionException.java	AssignOperations.java

BooleanVariable.java	DivideOperations.java
DoubleVariable.java	EQOperations.java
ExpressionResolver.java	GTEOperations.java
GTOperations.java	IBinaryOp.java
IByRef.java	IntegerVariable.java
IntrinsicLinkValue.java	IPostalFunctionStrategy.java
IPostalLinkResolver.java	IPostalListStrategy.java
IPostalSetStrategy.java	IPostalType.java
IPostalVariable.java	IUnaryOp.java
LTEOperations.java	LTOperations.java
MulOperations.java	NEQOperations.java
OpClass.java	OROperations.java
PostalEnv.java	PostalError.java
PostalFunctionInterpreter.java	PostalFunction.java
PostalHashtableStrategy.java	PostalLinkedListStrategy.java
PostalLink.java	PostalList.java
PostalNilPtr.java	PostalOpCodes.java
PostalOperations.java	PostalScopeFlags.java
PostalScope.java	PostalSet.java
PostalTypeFactory.java	PostalTypeKind.java
PostalVariable.java	ReflectedFunction.java
StringVariable.java	SubtractionOperations.java

Test Plan

Overview

The testing has been integral part of the development phase. The unit tests have been split onto two levels: regression testing and unit testing. The regression tests encompassed the entire feature set provided by the language and are meant to ensure the integrity of the entire product, while the unit tests target the specific feature supported by the language.

Examples

The deployment package includes the directory examples/ which contains some sample programs demonstrating the part-of-speech tagging. To run the examples type:

> java -jar postal.jar examples/*.postal

Unit Testing & Regression Testing

Although they differ by their names, both levels of testing have a lot in common. In fact, the regression testing simply runs the ensemble of individual unit tests.

The unit testing was done using the command line scripting. To run the entire suite of unit tests, execute the following command from the project root:

> java -jar postal.jar tests/*.postal

Each individual unit test addresses a specific feature supported by the language. For example, arithmetics.portal unit test verifies the math operations. The names for unit tests were chosen accordingly to reflect the feature the unit test was designed for.

To execute an individual unit test, use the command above except instead of using the wildcard instructing the POSTaL interpreter to execute all files with extension .postal, specify an exact name of the unit test you wish to run:

> java -jar postal.jar tests/equal.postal

The unit tests may produce an output to the standard console; the assertions are used to validate the computation results against the golden standard to ensure the proper functioning of a specific operation or function. The golden standards have been chosen based on the expected correct response for the feature. Due to the time constraints and significant effort involved in the development of the regression test suite, the regression test result verification has been done manually and visually instead of fully being a fully automated process.

Here are some unit tests used during the testing phase:

Arithmetics.postal

```
function arithmetic_expressions()
{
	var result;
	var opr1 = 2.0; // declare as double
	var opr2 = 3.0; // declare as double
```

```
	var retval_add = 5;
	var retval_minus = -1;
	var retval_multi = 6;
	var retval_divide = 0.666666;

	result = assert(nil == nil);  if (result, println("success"), println("failure"));
	result = assert(retval_add == opr1 + opr2);  if (result, println("success"),
println("failure"));
	result = assert(retval_minus == opr1 - opr2); if (result, println("success"),
println("failure"));
	result = assert(retval_multi == opr1 * opr2); if (result, println("success"),
println("failure"));
	result = assert((retval_divide - opr1 / opr2) < 0.000001); if (result, println("success"),
println("failure"));
}

function main()
{
	arithmetic_expressions();
}
```

The above program tests the math operations supported by the language and compares the results with the test values. Assertions are generated when the expected and evaluated values do not match.

Greaterorequal.postal

```
function greater_or_equal_than_expressions()
{
	var result;
	var opr1 = 1;
	var opr2 = 2;
	var space = "|     ";

	println("        integer    double   string  boolean |");
	println("=======+=========+========+========+========+");
	print("integer  "); print(assert((1 >= 2) == nil)); print(space);
                                print(assert((1 >= 1.2))); print(space);
                                print(assert((1 >= "3") == nil)); print(space);
                                print(assert((1 >= true))); println(space);
	print(" double  "); print(assert((1.2 >= 2) == nil)); print(space);
                                print(assert(1.2 >= 2.2e-10)); print(space);
                                print(assert(3.4 >= "3")); print(space);
                                print(assert(14.2 >= true)); println(space);
	print(" string  "); print(assert(("1" >= 2))); print(space);
                                print(assert(("2" >= 2.2e-10) == nil)); print(space);
                                print(assert("2.2" >= "3")); print(space);
                                print(assert(("true" >= true))); println(space);
	print("boolean  "); print(assert((true >= 2) == nil)); print(space);
                                print(assert((true >= 2.2e-10))); print(space);
                                print(assert(true >= "nil")); print(space);
                                print(assert((nil >= true) == nil)); println(space);
}

function main()
{
	 greater_or_equal_than_expressions();
```

```
}
```

The above program tests the 'greater-or-equal' comparison operation. Assertions are reported if the expected results do not match the results of the evaluation.

Lessons Learned

In general, the experience using the ANTLR to generate compilers from grammar have been good. I'd suggest getting a book published by the author of the ANTLR to make your life easier. The Internet is a good source of information, but as far as ANTLR is concerned, there isn't much available due to the fact that the online documentation for the ANTLR is a work in progress, which is not likely to be finished any time soon (as of 2008).

The decision whether to make a compiler or an interpreter shall be made early during the requirements phase. Both solutions have their pros and cons. Unless explicitly mandated by the professor, I'd suggest the interpreter as it eliminates the code generation phase. However, if the performance is a concern, then the compiler should be the target. For languages such as Java, optimization is not a major concern, especially when reflection is used.

When allocating time out of your schedule just I like I had consider taking a couple of days off to concentrate of the design and the documentation as they both time consuming.

Appendix A.

POSTaL Grammar

```
header
{
package WS4115.Projects.Postal.Grammar;
}

options
{
	language = "Java";
}

class POSTALLexer extends Lexer;

options {
	k=6; // increased the number of lookahead symbols to prevent ANTLR warnings for ML_COMMENT token
	exportVocab=POSTAL;
	testLiterals=false;
}

tokens {
	"function"; "var"; "if"; "nil"; "true"; "return";
	"while"; "dolist"; "lambda"; "numeric"; "apply";
}

SL_COMMENT: "//"
	(~'\n')* '\n'
	{ _ttype = Token.SKIP; newline(); }
	;

ML_COMMENT
	:	"/*"
		(
			/* the rules '\r\n' and '\r' cause the ambiquity therefore turning the warnings
off.
			*/
			options {
				generateAmbigWarnings=false;
			}
		: { LA(2)!='/' }? '*'
		|	'\r' '\n'		{newline();}
		|	'\r'			{newline();}
		|	'\n'			{newline();}
		|	~('*'|'\n'|'\r')
		)*
		"*/"
			{ $setType(Token.SKIP); }
	;

LPAREN
options {
	paraphrase="'('";
}
	:	'('
	;

RPAREN
options {
	paraphrase="')'";
}
	:	')'
	;

LCURLY
```

```
options {
        paraphrase="'{'";
}
        :       '{'
        ;

RCURLY
options {
        paraphrase="'}'";
}
        :       '}'
        ;

STAR
options {
        paraphrase="'*'";
}
        :       '*'
        ;

MINUS
options {
        paraphrase="'-'";
}
        :       '-'
        ;

PLUS
options {
        paraphrase="'+'";
}
        :       '+'
        ;

DIV
options {
        paraphrase="'/'";
}
        :       '/'
        ;

ASSIGN
options {
        paraphrase="'='";
}
        :       '='
        ;

SEMI
options {
        paraphrase="';'";
}
        :       ';'
        ;

COMMA
options {
        paraphrase="','";
}
        :       ','
        ;

EQ
        : "=="
        ;

NEQ
        : "!="
        ;
GT
        : '>'
        ;
```

```
LT
	: '<'
	;
GTE
	: ">="
	;
LTE
	: "<="
	;
AND
	: "&&"
	;

OR
	: "||"
	;

CHAR_LITERAL
options {
	paraphrase="a char";
}
	:	'\'' (ESC|~'\'') '\''
	;

STRING_LITERAL
options {
	paraphrase="a string";
}
	:	'"' (ESC|~'"')* '"'
	;

protected
ESC	:	'\\'
		(	'n'
		|	'r'
		|	't'
		|	'b'
		|	'f'
		|	'"'
		|	'\''
		|	'\\'
		|	'0'..'3'
			(
				options {
					warnWhenFollowAmbig = false;
				}
			:	DIGIT
				(
					options {
						warnWhenFollowAmbig = false;
					}
				:	DIGIT
				)?
			)?
		|	'4'..'7'
			(
				options {
					warnWhenFollowAmbig = false;
				}
			:	DIGIT
			)?
		)
	;

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

protected
DOT: '.'
```

```
        ;

protected
NUM_INT
        : (DIGIT)+
        ;

protected
NUM_FLOAT
        : '.' (DIGIT)+
        ;

NUM_CONST
        : NUM_INT (NUM_FLOAT) ? (EXPONENT)?
        ;

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

ID
options {
        testLiterals = true;
        paraphrase="an identifier";
}
        :       ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*
        ;

WS!     :       (' '
        |       '\t'
        |       '\n'    {newline();}
        |       '\r')
                { _ttype = Token.SKIP; }
;
```

POSTaL Parser Grammar

```
header {
package WS4115.Projects.Postal.Grammar;
}

options
{
	language = "Java";
}

/*
 * Parser starts here
 */
class POSTALParser extends Parser;

options
{
	k=2;
	importVocab=POSTAL;
	buildAST=true;
}

tokens {
	ELIST; METHOD_CALL; DECL_SCOPE; DECL_VAR; DECL_METHOD; EXPLIST; VAR_INITIALIZER;
}


{
	private boolean _errorOccured = false;

	public void reportError(RecognitionException ex) {
		_errorOccured = true;
		super.reportError(ex);
	}

	public boolean hasErrorOccured () {
		return _errorOccured;
	}

	public void reportError(String error) {
		_errorOccured = true;
		super.reportError(error);
	}
}

// The root element represents the entire .postal program
program
	: (declaration)* EOF!
	;

declaration
	: (variableDeclaration) => variableDeclaration
	| functionDeclaration
	;

variableDeclaration
	: LITERAL_var! ID variableInitializer SEMI!
		{#variableDeclaration = #(#[DECL_VAR, "DECL_VAR"], #variableDeclaration); }
	;

inlineVariableDeclaration
	: LITERAL_var! variableInitializer
		{#inlineVariableDeclaration = #(#[DECL_VAR, "DECL_VAR"], #inlineVariableDeclaration); }
	;

functionDeclaration
	: LITERAL_function! ID LPAREN! argList RPAREN! functionBody
	{ #functionDeclaration = #(#[DECL_METHOD, "DECL_METHOD"], #functionDeclaration); }
	;
```

```
private
variableInitializer
	: (ASSIGN! expression)?
		{ #variableInitializer = #(#[VAR_INITIALIZER, "VAR_INITIALIZER"], #variableInitializer);
}
	;

functionBody
	: scopeStatement
	;

lambdaExpressionBody
	: functionBody
	;

lambdaActionBody
	: scopeStatement
	;

lambdaExpression
	: LITERAL_lambda^ LPAREN! argList RPAREN! lambdaExpressionBody
	;

statement
	: variableDeclaration
	| expression SEMI!
	| scopeStatement
	| if_statement
	| while_statement
	| dolist_statement
	| returnStatement
	| SEMI!
	;

returnStatement
	: LITERAL_return^ (expression)? SEMI!
	;

if_statement
	: LITERAL_if^ LPAREN! expr COMMA! expression (COMMA! expression)? RPAREN!
	// statement (options {greedy=true;}: LITERAL_else statement)?
	;

while_statement
	: LITERAL_while^ LPAREN! expr COMMA! expression RPAREN!
	;

dolist_statement
	: LITERAL_dolist^ LPAREN! sequenceExpression COMMA!
		(lambdaExpression|identifier)
		RPAREN!
	;

scopeStatement
	: LCURLY! (statement)* RCURLY!
		{#scopeStatement = #(#[DECL_SCOPE, "DECL_SCOPE"], #scopeStatement); }
	;

argList
	:	(identifier (COMMA! identifier)*
	| /* nothing */
	)
		{#argList = #(#[ELIST,"ELIST"], #argList);}
	;

// A list of expressions.
expressionList
	:	(expression (COMMA! expression)*
	| /* nothing */
	)
		{#expressionList = #(#[EXPLIST,"EXPLIST"], #expressionList);}
	;
```

```
expression
	: assignmentExpression
	;

assignmentExpression
	: conditionalExpression (
		options { greedy=true; }:
			ASSIGN^ assignmentExpression
		)*
	;

expr
	: conditionalExpression
	;

conditionalExpression
	: additionExpression (
		(EQ^|NEQ^|LT^|GT^|GTE^|LTE^|AND^|OR^) additionExpression
		)?
	;

additionExpression
	: multiplicationExpression (
		(PLUS^|MINUS^) multiplicationExpression
		)*
	;

multiplicationExpression
	: primaryExpression (
		(STAR^|DIV^) primaryExpression
		)*
	;

primaryExpression
	: primaryIdentExpression
	| NUM_CONST
	| CHAR_LITERAL
	| STRING_LITERAL
	| MINUS^ primaryExpression
	| LITERAL_true
	| LITERAL_nil
	| numericExpression
	| (LPAREN! expression RPAREN!)
	| lambdaExpression
	| evaluationExpression
	;

evaluationExpression
	: LITERAL_apply^ LPAREN! (identifier | lambdaExpression) (COMMA! expressionList)? RPAREN!
	;

identifier
	: ID
    ;

primaryIdentExpression
	: ID^
		(lp:LPAREN^ {#lp.setType(METHOD_CALL); #lp.setText("METHOD_CALL");} expressionList
RPAREN! )?
	;

numericExpression
	: LITERAL_numeric^ LPAREN! expr RPAREN!
	;

sequenceExpression
	: primarySequenceExpession
	;

primarySequenceExpession
	: primaryIdentExpression
```

```
| STRING_LITERAL
;
```

POSTaL Translator (based on the AST tree walker)

```
header {
  package WS4115.Projects.Postal.Grammar;
  import java.*;
  import java.io.*;
  import java.util.*;
  import WS4115.Projects.Postal.Types.*;
  import WS4115.Projects.Postal.AST.*;
  import java.lang.reflect.InvocationTargetException;
}

class POSTALWalker extends TreeParser;
options
{
	importVocab=POSTALParser;
}

{
	protected IPostalType evaluateFunction(String methodName, AST argList, AST targetNode) throws
RecognitionException
	{
		IPostalType result = null;
		 IPostalType func = PostalEnv.instance().findSymbol(methodName, true);
		 if (func instanceof PostalLink)
			func = ((PostalLink)func).resolve();

		 if (!func.isValid() || (func.getKind() != PostalTypeKind.eKindFunction)) {
			PostalEnv.instance().ReportError("Function \"" + methodName + "\" is
undefined.");
		 }
		 PostalList exprList = null;
		 if (argList != null)
			exprList = expressionList(argList);
		else
			exprList = (PostalList)PostalTypeFactory.createList("EXPR_LIST");

		if (func instanceof PostalFunctionInterpreter) {
			PostalFunctionInterpreter intrf = (PostalFunctionInterpreter)func;
			if (exprList.size() != intrf.arguments().size())
			{
				result = PostalTypeFactory.createError("The function is declared to
take \"" + intrf.arguments().size() + "\", but \"" + exprList.size() + "\" was specified.");
			}
			else {
				// Create a function scope and populate it with arguments
				PostalScope scope =
(PostalScope)PostalTypeFactory.createScope("METHOD_CALL_SCOPE");
				scope.putFlags(PostalScopeFlags.eReturnAllowed);

				for (int i = 0; i < intrf.arguments().size(); i++) {
					String varName = intrf.arguments().get(i).getName();
					scope.put(PostalTypeFactory.createVariable(varName,
exprList.get(i)));
				}

				PostalEnv.instance().enterScope(scope);
				// Interpret and execute the function implementation
				result = blockStatement((AST)intrf.getImplementation());
				PostalEnv.instance().resetReturn();
				PostalEnv.instance().exitScope();
			}
		}
		else if (func instanceof ReflectedFunction)
		{
			//System.out.println("Invoking the built-in function \"" + func.getName() +
"\"...");
			ReflectedFunction reflectedFunc = (ReflectedFunction)func;
			try
			{
				result = reflectedFunc.invoke(exprList);
```

```
                        }
                        catch(InvocationTargetException ex)
                        {
                                result = PostalTypeFactory.createNil();
                                long line = ((AdvancedCommonAST)targetNode).getline();
                                long column = ((AdvancedCommonAST)targetNode).getcolumn();
                                PostalEnv.instance().ReportError("Assertion at line:" + line + ", column:"
+ column + "\n" + ex);
                        }
                        catch(Exception ex)
                        {
                                result = PostalTypeFactory.createNil();
                                long line = ((AdvancedCommonAST)targetNode).getline();
                                long column = ((AdvancedCommonAST)targetNode).getcolumn();
                                PostalEnv.instance().ReportError("Exception at line:" + line + ", column:"
+ column + "\n" + ex);
                        }
                }
                return result;
        }
}

program
{
        IPostalType result = null;
        PostalEnv.instance().setInterpreter(this);
}
        : (result=functionDeclaration | result=variableDeclaration)*
        ;

functionDeclaration returns [IPostalType result]
{
        PostalList parameters = null;
        result = null;
}
        : #(DECL_METHOD fid:ID parameters=argList implementation:.)
                {
                        IPostalType found = PostalEnv.instance().findSymbol(#fid.getText(), true);
                        if (found.isValid() && found.getKind() == PostalTypeKind.eKindFunction)
                        {
                                PostalEnv.instance().ReportError("Function \"" + #fid.getText() + "\" has
already been defined.");
                        }

                         PostalFunction f =
(PostalFunction)PostalTypeFactory.createFunctionInterpreter(#fid.toString(), parameters,
#implementation);
                         PostalEnv.instance().getRootScope().put(f);
                         result = f;
                }
        ;

private
argList returns [PostalList sv]
{
        sv = null;
        PostalList tempList = (PostalList)PostalTypeFactory.createList("ARG_LIST");
}
        : #(ELIST (s:ID
                                {
                                        tempList.add(PostalTypeFactory.createString(s.getText(),
s.getText()));
                                }
                        )*
                )
                {
                        sv = tempList;
                }
        ;

private
expressionList returns [PostalList result]
```

```
{
	result = (PostalList)PostalTypeFactory.createList("EXPLIST");
}
	: #(EXPLIST {  IPostalType param = null; }   (param=expression { result.add(param); } )* )
	;

variableDeclaration returns [IPostalType result]
{
	result = null;
}
	: #(DECL_VAR vid:ID
		{
			String varName = #vid.getText();
			IPostalType found = PostalEnv.instance().findSymbol(varName, false); // do not
look up parent scopes
			if (found.isValid() && found.getKind() == PostalTypeKind.eKindLink)
			{
				PostalEnv.instance().ReportError("Variable \"" + varName + "\" has already
been defined.");
			}
			result = PostalTypeFactory.createVariable(varName);
			PostalEnv.instance().getCurrentScope().put(result);
		}
				#(VAR_INITIALIZER
					(initexpr:.
						{
							// Note: forward declarations are not supported,
i.e. any functions returning
							// values used to initialize the variable must be
declared before the
							// variable declaration.
							IPostalType initialValue = expression(#initexpr);
							result.assign(initialValue);
						}
					)?
				)
		)
	;

blockStatement returns [IPostalType result]
{
	result = null;
}
	: #(DECL_SCOPE
			{
				PostalScope scope =
(PostalScope)PostalTypeFactory.createScope("DECL_SCOPE");
				scope.putFlags(PostalScopeFlags.eTransparent);
				PostalEnv.instance().enterScope(scope);
			}
			(stat:.
				{
					result = statement(#stat);
					// If the program requested an early return, then
					// short-circuit the current control flow and exit the loop.
					if (PostalEnv.instance().isReturnRequested())
						break;
				}
			)*
			{
				scope = PostalEnv.instance().exitScope();
			}
		)
	;

statement returns [IPostalType result]
{
	result = null;
}
	: result = variableDeclaration
	| #(LITERAL_return (result = expression)?
			{
```

```
                        if (result == null)
                                result = PostalTypeFactory.createBoolean("RET_VAL", true);
                        if (!PostalEnv.instance().isReturnAllowed())
                        {
                                PostalEnv.instance().ReportError("Current program flow does not
allow early return.");
                        }
                        PostalEnv.instance().requestReturn();
                }
        )
        | #(LITERAL_if cond1:. expr1:. (elseexpr:.)?
                {
                        boolean bSuccess = false;
                        IPostalType oBool = expression(#cond1);
                        oBool = PostalOperations.instance().convert(oBool, Boolean.TYPE);
                        if (oBool.isValid() && (oBool instanceof BooleanVariable))
                                bSuccess = ((BooleanVariable)oBool).value();
                        if (bSuccess)
                                result = expression(#expr1);
                        else
                        {
                                if (elseexpr != null)
                                        result = expression(#elseexpr);
                        }
                }
        )
        | #(LITERAL_dolist dolistiter:. dolistexpr:.
                {
                        IPostalType o = expression(#dolistiter);
                        if (o.getKind() == PostalTypeKind.eKindLink)
                                o = ((PostalLink)o).resolve();
                        if (!(o instanceof PostalList)) {
                                System.out.println("Specified type is: " + o.getKind());
                                PostalEnv.instance().ReportError("dolist operation requires a list.");
                        }
                        else
                        {
                                PostalList list = (PostalList)o;
                                assert list != null;

                                String methodName=lambdaOrIdExpression(#dolistexpr);
                                IPostalType func = PostalEnv.instance().findSymbol(methodName, true);
                                if (!func.isValid() || (func.getKind() != PostalTypeKind.eKindFunction)) {
                                        PostalEnv.instance().ReportError("Function \"" + methodName + "\"
is undefined.");
                                }
                                if (((PostalFunction)func).arguments().size() != 1)
                                        PostalEnv.instance().ReportError("dolist expects a function or a
lambda expression with a single argument.");

                                PostalList exprList =
(PostalList)PostalTypeFactory.createList("EXPR_LIST");
                                while (list.size() != 0)
                                {
                                        exprList.clear();
                                        exprList.add(list.get(0));

                                        if (func instanceof PostalFunctionInterpreter) {
                                                PostalFunctionInterpreter intrf =
(PostalFunctionInterpreter)func;
                                                // Create a function scope and populate it with arguments
                                                PostalScope scope =
(PostalScope)PostalTypeFactory.createScope("METHOD_CALL_SCOPE");
                                                scope.putFlags(PostalScopeFlags.eReturnAllowed);

                                                for (int i = 0; i < intrf.arguments().size(); i++) {
                                                        String varName = intrf.arguments().get(i).getName();
                                                        scope.put(PostalTypeFactory.createVariable(varName,
exprList.get(i)));
                                                }

                                                PostalEnv.instance().enterScope(scope);
```

```
                                        // Interpret and execute the function implementation
                                        result = blockStatement((AST)intrf.getImplementation());
                                        PostalEnv.instance().resetReturn();
                                        PostalEnv.instance().exitScope();
                                }
                                else if (func instanceof ReflectedFunction)
                                {
                                        ReflectedFunction reflectedFunc = (ReflectedFunction)func;
                                        try
                                        {
                                                result = reflectedFunc.invoke(exprList);
                                        }
                                        catch(Exception ex)
                                        {
                                                result = PostalTypeFactory.createNil();
                                                PostalEnv.instance().ReportError(ex.getMessage());
                                        }
                                }
                                list = list.tail();
                        }
                        list = list.head();
                }
        }
        )
        | #(LITERAL_while cond:. while_body:.
                {
                        boolean bContinue =  false;
                        IPostalType oBool = result = expression(#cond);
                        oBool = PostalOperations.instance().convert(oBool, Boolean.TYPE);
                        if (oBool.isValid() && (oBool instanceof BooleanVariable))
                                bContinue = ((BooleanVariable)oBool).value();
                        while (bContinue)
                        {
                                result = expression(#while_body);
                                oBool = expression(#cond);
                                //System.out.println(oBool.toString());
                                if (oBool.isValid() && (oBool instanceof BooleanVariable))
                                        bContinue = ((BooleanVariable)oBool).value();
                                else
                                        bContinue = false;
                        }
                }
        )
        | result = expression
        ;

expression returns [IPostalType result]
{
        result = null;
}
        : #(ASSIGN lhs:. rhs:.)
                {
                        result = expression(#lhs);
                        IPostalType rhsVal = expression(#rhs);
                        result = result.assign(rhsVal);
                }
        | result = expressionNoAssignment
        ;

private
expressionNoAssignment returns [IPostalType result]
{
        result = null;
        IPostalType rhs = null;
        IPostalType expr = null;
        String methodName = null;
        IPostalType lhs;
}
        : #(LT lhs=expression rhs=expression
                {
                        result = PostalOperations.instance().lessThan(lhs, rhs);
                }
```

```
		)
	| #(GT lhs=expression rhs=expression
		{
			result = PostalOperations.instance().greaterThan(lhs, rhs);
		}
		)
	| #(EQ lhs=expression rhs=expression
		{
			result = PostalOperations.instance().equalTo(lhs, rhs);
		}
		)
	| #(NEQ lhs=expression rhs=expression
		{
			result = PostalOperations.instance().nonEqualTo(lhs, rhs);
		}
		)
	| #(GTE lhs=expression rhs=expression
		{
			result = PostalOperations.instance().greaterOrEqualThan(lhs, rhs);
		}
		)
	| #(LTE lhs=expression rhs=expression
		{
			result = PostalOperations.instance().lessOrEqualThan(lhs, rhs);
		}
		)
	| #(AND lhs=expression rhs=expression
		{
			result = PostalOperations.instance().and(lhs, rhs);
		}
		)
	| #(OR lhs=expression rhs=expression
		{
			result = PostalOperations.instance().or(lhs, rhs);
		}
		)
	| #(PLUS lhs=expression rhs=expression
		{
			result = PostalOperations.instance().add(lhs, rhs);
		}
		)
	| #(MINUS lhsexp:. (rhsexp:.)?
		{
			lhs = expression(#lhsexp);
			if (rhsexp != null) {
				rhs = expression(#rhsexp);
				result = PostalOperations.instance().subtract(lhs, rhs);
			}
			else {
				lhs = expression(#lhsexp);
				if (lhs instanceof PostalLink)
					lhs = ((PostalLink)lhs).resolve();
				if (lhs instanceof IntegerVariable)
					result =
PostalOperations.instance().subtract(PostalTypeFactory.createInteger("", 0), lhs);
				else
					result =
PostalOperations.instance().subtract(PostalTypeFactory.createDouble("", 0.0), lhs);
			}
		}
		)
	| #(STAR lhs=expression rhs=expression
		{
			result = PostalOperations.instance().multiply(lhs, rhs);
		}
		)
	| #(DIV lhs=expression rhs=expression
		{
			result = PostalOperations.instance().divide(lhs, rhs);
		}
		)
	| #(NUM_CONST
```

```
		{
			// NUM_CONST can be a integer or floating-point number.
			try
			{
				// Trying to convert to integer value
				int iVal = Integer.parseInt(#NUM_CONST.getText());
				result = PostalTypeFactory.createInteger("INT_CONST", iVal);
			}
			catch(NumberFormatException ex)
			{
				// conversion to integer has failed, next step is to
				// try converting to a floating-point number.
				try
				{
					Double dblVal = Double.parseDouble(#NUM_CONST.getText());
					result = PostalTypeFactory.createDouble("DBL_CONST", dblVal);
				}
				catch(NumberFormatException ex2)
				{
					// There is something wrong, bail out
					throw ex2;
				}
			}
		 })
	| #(CHAR_LITERAL
		{ result = PostalTypeFactory.createString("STR_CONST", #CHAR_LITERAL.getText()); })
	| #(STRING_LITERAL
		{
			String s = #STRING_LITERAL.getText();
			result = PostalTypeFactory.createString("STR_CONST", s.substring(1, s.length() -
1));
		})
	| #(LITERAL_true
		{ result = PostalTypeFactory.createBoolean("BOOL_CONST", true); })
	| #(LITERAL_nil
		{ result = PostalTypeFactory.createNil(); })
	| #(LITERAL_numeric rhs=expression
		{ result = PostalOperations.instance().convert(rhs, Integer.TYPE); } )
	| result = lambdaExpression
	| #(LITERAL_apply methodName=lambdaOrIdExpression (explist:.)?
			{
				result = evaluateFunction(methodName, #explist, #LITERAL_apply);
			}
		)
	| #(ID
			{
				result = PostalEnv.instance().findSymbol(#ID.getText(), true);
				if (result.getKind() != PostalTypeKind.eKindLink) {
					PostalEnv.instance().ReportError("Symbol \"" + #ID.getText() + "\"
is undefined.");
				}
			}
		)
	| #(METHOD_CALL methodName=methodNameIdentifier (method_explist:.)?
			{
				result = evaluateFunction(methodName.toString(), #method_explist,
#METHOD_CALL);
			}
		)
	;

private
methodNameIdentifier returns [String result]
{
	result = null;
}
	: #(ID
		{ result = #ID.getText(); }
		)
	;

private
```

```
lambdaExpression returns [IPostalType fnLambda]
{
	PostalList parameters = null;
	fnLambda = null;
}
	 : #(LITERAL_lambda parameters=argList implementation:.)
		{
			// Create a transient name for the lambda expression
			String name = "anonymous_lambda_expression-" +
Long.toString(System.currentTimeMillis());
			fnLambda = PostalTypeFactory.createFunctionInterpreter(name, parameters,
#implementation);
			PostalEnv.instance().getCurrentScope().put(fnLambda);
		}
	 ;


private
lambdaOrIdExpression returns [String anonymousName]
{
	anonymousName = null;
	IPostalType lambdaDecl = null;
}
	: #(ID { anonymousName = #ID.getText(); } )
	| lambdaDecl=lambdaExpression
		{ anonymousName = lambdaDecl.getName(); }
	;
```

Appendix B.

Source code listing (by module)

AdvancedCommonAST.java

```
package WS4115.Projects.Postal.AST;
import antlr.CommonAST;
import antlr.Token;
import antlr.collections.AST;

public class AdvancedCommonAST extends CommonAST {

	public static final long serialVersionUID = 2000;

	private long _line;
	private long _column;

	public void initialize(Token tok) {
		super.initialize(tok);
		_line = tok.getLine();
		_column = tok.getColumn();
	}

	public void initialize(AST t)
	{
		super.initialize(t);
		_line = t.getLine();
		_column = t.getColumn();
	}
	public long getline() { return _line; }
	public long getcolumn() { return _column; }
}
```

HMMTagger.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
```

```
 *
 */
package WS4115.Projects.Postal.brown;

import com.aliasi.hmm.HiddenMarkovModel;
import com.aliasi.hmm.HmmDecoder;
import com.aliasi.util.Streams;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.util.*;

/**
 * Performs the Part-of-Speech tagging using pre-built model file. The model is trained
 * using the Brown Corpus corpora. The project does not actually do the training; instead,
 * a pre-existing model file is used.
 *
 * @author pnalyvayko
 *
 */
public final class HMMTagger {

	private HmmDecoder _decoder = null;
	private static Hashtable<String, HMMTagger> _taggers = null;

	protected HMMTagger() {
	}

	protected void loadModel(String modelFile) throws IOException, ClassNotFoundException {
		// Try loading the HMM model from a file on the disk
		boolean bLoadFromManifest = false;
		ObjectInputStream objIn = null;
		try
		{
		 FileInputStream fileIn = new FileInputStream(modelFile);
		 objIn = new ObjectInputStream(fileIn);
		}
		catch(Exception ex)
		{
			bLoadFromManifest = true;
		}
		if (bLoadFromManifest) {
			InputStream inputStream = this.getClass().getResourceAsStream("/" + modelFile);
			if (inputStream == null)
				throw new IOException();
		 objIn = new ObjectInputStream(inputStream);
		}
	 HiddenMarkovModel hmm = (HiddenMarkovModel) objIn.readObject();
	 Streams.closeInputStream(objIn);
	 _decoder = new HmmDecoder(hmm);
	}

	public static HMMTagger getTagger(String modelFile) {
		if (_taggers == null) {
```

```
                        _taggers = new Hashtable<String, HMMTagger>();
                }
                String filePath = modelFile.toUpperCase();
                HMMTagger instance = null;
                if (!_taggers.containsKey(filePath))
                {
                        try
                        {
                                instance = new HMMTagger();
                                instance.loadModel(modelFile);
                                _taggers.put(filePath, instance);
                        }
                        catch(IOException ex) {
                                System.out.println("IO Exception:" + ex.getMessage());
                        }
                        catch(ClassNotFoundException ex) {
                                System.out.println("ClassLoader Exception:" + ex.getMessage());
                        }
                } else {
                        instance = _taggers.get(filePath);
                }
                return instance;
        }

         public String[] firstBest(String[] tokens) {
        return _decoder.firstBest(tokens);
    }

}
```

BuiltinFunctionProvider.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
package WS4115.Projects.Postal.builtin;

import WS4115.Projects.Postal.Types.*;
import com.aliasi.tokenizer.Tokenizer;
import com.aliasi.tokenizer.IndoEuropeanTokenizerFactory;
import WS4115.Projects.Postal.brown.*;

//import java.io.FileInputStream;
//import java.util.*;
```

```
//import java.io.FileInputStream;
//import java.io.IOException;
import java.io.*;

public final class BuiltinFunctionProvider {

	public BuiltinFunctionProvider() {

	}

	protected IPostalType removeLink(IPostalType link) {
		if (link instanceof PostalLink)
			return ((PostalLink)link).resolve();
		return link;
	}
	/**
	 * Prints the expression to the standard output.
	 * @param expression
	 * @return nil
	 */
	public IPostalType println(IPostalType expression) {
		System.out.println(expression.toString());
		return PostalTypeFactory.createNil();
	}
	/**
	 * Prints the expression to the standard output. Carriage return and new line are not added.
	 * @param expression
	 * @return
	 */
	public IPostalType print(IPostalType expression) {
		System.out.print(expression.toString());
		return PostalTypeFactory.createNil();
	}
	/**
	 * Displays an assertion.
	 * @param expression
	 * @return
	 */
	public IPostalType doassert(IPostalType expression) throws Exception, AssertionException {
		boolean bDisplayAssertion = false;
		IPostalType value = removeLink(expression);
		value = removeLink(value);
		if (value instanceof PostalNilPtr)
			bDisplayAssertion = true;
		else {
			value = PostalOperations.instance().convert(value, Boolean.TYPE);
			bDisplayAssertion = !(value != null && value.isValid() && (value instanceof
BooleanVariable) && ((BooleanVariable)value).value());
		}
		if (bDisplayAssertion) {
			if (value == null)
				value = PostalTypeFactory.createNil();
			//PostalEnv.instance().ReportError(value.toString() + " ==> assertion failed.");
		}
		//System.out.println(bDisplayAssertion);
		if (bDisplayAssertion) {
```

```
                throw new AssertionException("Assertion's occured.");
        }
        return PostalTypeFactory.createBoolean("assertion_result", !bDisplayAssertion);
}

/**
 * Returns the rest of the list minus the first element. If the list contains
 * just one element or the list is not actually a list, then the result of
 * the tail function is nil.
 * @param expression
 * @return a list object
 */
public IPostalType tail(IPostalType expression) {
        expression = removeLink(expression);
        if (expression instanceof PostalList) {
                PostalList newList = (PostalList)((PostalList)expression).tail();
                return newList;
        }
        return PostalTypeFactory.createNil();
}

/**
 * Returns the head of the list.
 * @param expression
 * @return a list object
 */
public IPostalType head(IPostalType expression) {
        expression = removeLink(expression);
        if (expression instanceof PostalList) {
                PostalList newList = (PostalList)((PostalList)expression).head();
                return newList;
        }
        return PostalTypeFactory.createNil();
}

public IPostalType dup(IPostalType expression) {
        IPostalType result = null;
        expression = removeLink(expression);
        try
        {
                result = expression.clone();
        }
        catch(CloneNotSupportedException ex) {
                result = PostalTypeFactory.createError("Cannot duplicate the given expression");
        }
        return result;
}

/**
 * Creates an empty list.
 * @param expression
 * @return
 */
public IPostalType list() {
        return PostalTypeFactory.createList("TRANSIENT_LIST");
```

```
	}
	/**
	 * Creates an empty associative collection.
	 * @return
	 */
	public IPostalType hashtable() {
		return PostalTypeFactory.createMap("TRANSIENT_HASHTABLE");
	}

	/**
	 * Returns a length of the list.
	 * @param list
	 * @return
	 */
	public IPostalType list_length(IPostalType list) {
		list = removeLink(list);
		if (list instanceof PostalList) {
			return PostalTypeFactory.createInteger("LIST_LENGTH",
((PostalList)list).size());
		}
		return PostalTypeFactory.createInteger("LIST_LENGTH", 0);
	}
	/**
	 * Returns whether the specified object is a list.
	 * @param list
	 * @return
	 */
	public IPostalType isList(IPostalType list) {
		list = removeLink(list);
		return list instanceof PostalList ? PostalTypeFactory.createBoolean("", true):
PostalTypeFactory.createBoolean("", false);
	}
	/**
	 * Returns true if the given expression is a function or lambda expression.
	 * @param expression
	 * @return
	 */
	public IPostalType isLambda(IPostalType expression) {
		expression = removeLink(expression);
		return expression instanceof PostalFunction ? PostalTypeFactory.createBoolean("", true):
PostalTypeFactory.createBoolean("", false);
	}
	/**
	 * Returns true if the given expression represents a numeric value.
	 * @param expression
	 * @return
	 */
	public IPostalType isNumeric(IPostalType expression) {
		expression = removeLink(expression);
		return (expression instanceof IntegerVariable) || (expression instanceof DoubleVariable)
? PostalTypeFactory.createBoolean("", true): PostalTypeFactory.createBoolean("", false);
	}

	/**
	 * Returns true if the given expression represents a literal (single character or a sequence of
characters).
	 */
```

```
	public IPostalType isLiteral(IPostalType expression) {
		expression = removeLink(expression);
		return (expression instanceof IntegerVariable) || (expression instanceof DoubleVariable)
? PostalTypeFactory.createBoolean("", true): PostalTypeFactory.createBoolean("", false);
	}

	/**
	 * Returns n-th element of a list, an error if the position is invalid or
	 * nil if the specified list object is not a list.
	 * @param pos
	 * @param list
	 * @return
	 */
	public IPostalType nth(IPostalType pos, IPostalType list) {
		pos = removeLink(pos);
		list =removeLink(list);
		if (list instanceof PostalList) {
			if (pos.getKind() == PostalTypeKind.eKindValue) {
				pos = ((PostalVariable)pos).convert(Integer.class);
				if (pos.isValid() && pos instanceof IntegerVariable) {
					return ((PostalList)list).get(((IntegerVariable)pos).value());
				}
			}
		}
		return PostalTypeFactory.createNil();
	}
	/**
	 * Inserts the element to the head of the list. If the specified list is not
	 * a list, then the function returns nil.
	 * @param elem
	 * @param list
	 * @return
	 */
	public IPostalType cons(IPostalType elem, IPostalType list) {
		list = removeLink(list);
		elem = removeLink(elem);

		if (list instanceof PostalNilPtr)
			list = (PostalList)PostalTypeFactory.createList("AUTO_LIST");
		if (list instanceof PostalList) {
			((PostalList)list).insert(0, elem);
			return list;
		}
		return PostalTypeFactory.createNil();
	}
	/**
	 * Removes and returns the first element of the list. The original list is affected.
	 * @param list
	 * @return the first element of the list or nil if the list is empty.
	 */
	public IPostalType pop(IPostalType list) {
		list = removeLink(list);
		if (list instanceof PostalList) {
			PostalList l = (PostalList)list;
			if (l.size() > 0) {
				IPostalType result = l.get(0);
```

```
                        l.removeAt(0);
                        return result;
                }
        }
        return PostalTypeFactory.createNil();
    }

    /**
     * Tokenizes the input string and returns a list of tokens. The function uses the LingPipe API
     * classes sunch as IndoEuropeanTokenizerFactory and Tokenizer to perform the actual
     * tokenization.
     * @param str
     * @return
     */
    public IPostalType words(IPostalType val) {
        //System.out.println("Calling 'words' with " + val.toString());
        val = removeLink(val);
        IPostalType str = PostalOperations.instance().convert(val, String.class);
        if (str.isValid() && str.getKind() == PostalTypeKind.eKindValue) {
                String s = str.toString();
                Tokenizer tokenizer =
IndoEuropeanTokenizerFactory.FACTORY.tokenizer(s.toCharArray(), 0, s.length());
                String[] tokens = tokenizer.tokenize();

                PostalList list = (PostalList)PostalTypeFactory.createList("TOKENIZED_LIST");
                for(String token : tokens) {
                        list.add(PostalTypeFactory.createString("TOKEN", token));
                }
                return list;
        }
        return PostalTypeFactory.createError("The input parameter was not a string.");
    }

    public IPostalType nbest(IPostalType val) {
        //System.out.println("Calling 'nbest' with " + val.toString());
        val = removeLink(val);
        if (val != null && val.getKind() == PostalTypeKind.eKindList) {
                PostalList list = (PostalList)val;
                HMMTagger tagger = HMMTagger.getTagger("WS4115/Projects/Postal/data/pos-en-
general-brown.HiddenMarkovModel");
                if (tagger != null) {

                        String[] tokens = new String[list.size()];
                        for(int i = 0; i < list.size(); i++) {
                                IPostalType t = list.get(i);
                                tokens[i] = t.toString();
                        }
                        String[] tags = tagger.firstBest(tokens);

                        if (tags.length != tokens.length)
                                return PostalTypeFactory.createError("The tokens and the tags
collections have different sizes.");

                        PostalList result = (PostalList)PostalTypeFactory.createList("POS_LIST");

                        for(int pos = 0; pos < tokens.length; pos++) {
```

```
                                        PostalList elem =
(PostalList)PostalTypeFactory.createList("POST_NTAG_ELEM");
                                        elem.add(PostalTypeFactory.createString("TOKEN", tokens[pos]));
                                        elem.add(PostalTypeFactory.createString("TAG", tags[pos]));
                                        result.add(elem);
                                }

                                return result;
                        }
                }
                return PostalTypeFactory.createError("Expected a list as an argument.");
        }

        public IPostalType openf(IPostalType fileName) {
                fileName =removeLink(fileName);
                if (fileName.isValid() && fileName.getKind() == PostalTypeKind.eKindValue) {
                        fileName = ((PostalVariable)fileName).convert(String.class);
                        String filePath = fileName.toString();

                        String contents = TextFileHelper.getContents(new File(filePath));
                        Tokenizer tokenizer =
IndoEuropeanTokenizerFactory.FACTORY.tokenizer(contents.toCharArray(), 0, contents.length());
                        String[] tokens = tokenizer.tokenize();

                        PostalList list = (PostalList)PostalTypeFactory.createList("TOKENIZED_LIST");
                        for(String token : tokens) {
                                list.add(PostalTypeFactory.createString("TOKEN", token));
                        }
                        return list;
                }
                return PostalTypeFactory.createError("Cannot open specified file.");
        }
        /**
         * Stores the contents of the given list into the file using the file name. The
         * contents of the existing file are either overwritten or appended to depending
         * on the value of the third argument.
         * @param list
         * @param fileName
         * @param overwrite
         * @return
         */
        public IPostalType savef(IPostalType list, IPostalType fileName) throws FileNotFoundException,
IOException {
                list = removeLink(list);
                fileName = removeLink(fileName);
                if (!(list instanceof PostalList)) {
                        return PostalTypeFactory.createError("The first argument must be a list.");
                }
                if (!(fileName instanceof StringVariable)) {
                        return PostalTypeFactory.createError("The second argument must a string.");
                }
                PostalList l = (PostalList)list;
                String sFileName = ((StringVariable)fileName).value();

                StringBuilder sb = new StringBuilder();
                for(int i = 0; i < l.size(); i++) {
```

```
                }

                File f = new File(sFileName);
                TextFileHelper.setContents(f, sb.toString());
                return PostalTypeFactory.createNil();
        }
        /**
         * Returns a value associated with the given key. If the specified object
         * is not a hashtable, the function will return nil.
         * @param set
         * @param key
         * @return
         */
        public IPostalType assoc(IPostalType set, IPostalType key) {
                set = removeLink(set);
                key = removeLink(key);
                if (set instanceof PostalSet) {
                        PostalSet aSet = (PostalSet)set;
                        String sKey = PostalOperations.instance().convert(key, String.class).toString();
                        if (aSet.containsKey(sKey)) {
                                return aSet.get(sKey);
                        }
                }
                return PostalTypeFactory.createNil();
        }
        /**
         * Sets a new value for the specified key.
         * @param set
         * @param key
         * @param value
         * @return
         */
        public IPostalType sassoc(IPostalType set, IPostalType key, IPostalType value) {
                set = removeLink(set);
                key = removeLink(key);
                if (set instanceof PostalSet) {
                        PostalSet aSet = (PostalSet)set;
                        String sKey = PostalOperations.instance().convert(key, String.class).toString();
                        aSet.put(sKey, value);
                        return aSet;
                }
                return PostalTypeFactory.createNil();
        }
}
```

TextFileHelper.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
```

```
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//

/**
 *
 */
package WS4115.Projects.Postal.builtin;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

/**
 * @author pnalyvayko
 *
 */
public final class TextFileHelper {

          /**
          * Fetch the entire contents of a text file, and return it in a String.
          * This style of implementation does not throw Exceptions to the caller.
          *
          * @param aFile is a file which already exists and can be read.
          */
          static public String getContents(File aFile) {
            //...checks on aFile are elided
            StringBuilder contents = new StringBuilder();

            try {
              //use buffering, reading one line at a time
              //FileReader always assumes default encoding is OK!
              BufferedReader input =  new BufferedReader(new FileReader(aFile));
              try {
                String line = null; //not declared within while loop
                /*
                * readLine is a bit quirky :
                * it returns the content of a line MINUS the newline.
                * it returns null only for the END of the stream.
                * it returns an empty String if two newlines appear in a row.
                */
                while (( line = input.readLine()) != null){
                  contents.append(line);
                  contents.append(System.getProperty("line.separator"));
                }
              }
```

```
              finally {
                input.close();
              }
            }
            catch (IOException ex){
              ex.printStackTrace();
            }

            return contents.toString();
          }

          /**
          * Change the contents of text file in its entirety, overwriting any
          * existing text.
          *
          * This style of implementation throws all exceptions to the caller.
          *
          * @param aFile is an existing file which can be written to.
          * @throws IllegalArgumentException if param does not comply.
          * @throws FileNotFoundException if the file does not exist.
          * @throws IOException if problem encountered during write.
          */
          static public void setContents(File aFile, String aContents)
                                         throws FileNotFoundException, IOException {
            if (aFile == null) {
              throw new IllegalArgumentException("File should not be null.");
            }
            if (!aFile.exists()) {
              throw new FileNotFoundException ("File does not exist: " + aFile);
            }
            if (!aFile.isFile()) {
              throw new IllegalArgumentException("Should not be a directory: " + aFile);
            }
            if (!aFile.canWrite()) {
              throw new IllegalArgumentException("File cannot be written: " + aFile);
            }

            //use buffering
            Writer output = new BufferedWriter(new FileWriter(aFile));
            try {
              //FileWriter always assumes default encoding is OK!
              output.write( aContents );
            }
            finally {
              output.close();
            }
          }

}
```

AdditionOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
```

```
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class AdditionOperations {


	private static AdditionOperations _instance;
	public static AdditionOperations instance() {
		if (_instance == null) {
			_instance = new AdditionOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() + i2.value());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() + i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() + (i2.value()?1:0));
		}
	}
	public class DoString2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
```

```
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createInteger("", i1.value() +
Integer.parseInt(i2.value()));
                }
        }

        public class DoDouble2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createDouble("", i1.value() + i2.value());
                }
        }
        public class DoInt2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createDouble("", i1.value() +
i2.value().doubleValue());
                }
        }
        public class DoBoolean2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createDouble("", i1.value() + (i2.value() ? 1.0 :
0.0));
                }
        }
        public class DoString2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createDouble("", i1.value() +
Double.parseDouble(i2.value()));
                }
        }
        /*
        public class DoBoolean2Boolean extends OpClass {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createInteger("", i1.value() - i2.value());
                }
        }
        public class DoInteger2Boolean extends OpClass {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createInteger("", i1.value() != (i2.value() == 0 ?
false : true));
                }
        }
        public class DoDouble2Boolean extends OpClass {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
```

```
                        return PostalTypeFactory.createInteger("", i1.value() != (i2.value().intValue()
== 0 ? false : true));
                }
        }
        public class DoString2Boolean extends OpClass {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createInteger("", i1.value() != (i2.value().intValue()
== 0 ? false : true));
                }
        }
        */
        public class DoString2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createString("", i1.value() + i2.value());
                }
        }
        public class DoInteger2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createString("", i1.value() + i2.value().toString());
                }
        }
        public class DoDouble2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createString("", i1.value() + i2.value().toString());
                }
        }
        public class DoBoolean2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createString("", i1.value() + i2.value().toString());
                }
        }
}
```

ANDOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
```

```
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class ANDOperations {

	private static ANDOperations _instance;
	public static ANDOperations instance() {
		if (_instance == null) {
			_instance = new ANDOperations();
		}
		return _instance;
	}
	public class DoBoolean2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().booleanValue() &&
i2.value().booleanValue());
		}
	}
}
```

AssertionException.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//

package WS4115.Projects.Postal.Types;

public class AssertionException extends Exception {

	public static final long serialVersionUID = 1000;

	public AssertionException(String message) {
		super(message);
```

```
	}
}
```

AssignOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
package WS4115.Projects.Postal.Types;

public class AssignOperations {

	private static AssignOperations _instance;
	public static AssignOperations instance() {
		if (_instance == null) {
			_instance = new AssignOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			i1.forceAssign(i2.value());
			return i1;
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			i1.forceAssign(i2.value().intValue());
			return i1;
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			i1.forceAssign(i2.value() ? 1 : 0);
			return i1;
		}
	}
```

```
	public class DoString2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			i1.forceAssign(Integer.parseInt(i2.value()));
			return i1;
		}
	}

	public class DoDouble2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			i1.forceAssign(i2.value());
			return i1;
		}
	}
	public class DoInt2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			i1.forceAssign(i2.value().doubleValue());
			return i1;
		}
	}
	public class DoBoolean2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			i1.forceAssign((i2.value() ? 1.0 : 0.0));
			return i1;
		}
	}
	public class DoString2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			i1.forceAssign(Double.parseDouble(i2.value()));
			return i1;
		}
	}

	public class DoBoolean2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			i1.forceAssign(i2.value());
			return i1;
		}
	}
	public class DoInteger2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			i1.forceAssign(i2.value() == 0 ? false : true);
```

```
                        return i1;
                }
        }
        public class DoDouble2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        i1.forceAssign(i2.value().intValue() == 0 ? false : true);
                        return i1;
                }
        }
        public class DoString2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        i1.forceAssign(i2.value().intValue() == 0 ? false : true);
                        return i1;
                }
        }

        public class DoString2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        i1.forceAssign(i2.value());
                        return i1;
                }
        }
        public class DoInteger2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        i1.forceAssign(i2.value().toString());
                        return i1;
                }
        }
        public class DoDouble2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        i1.forceAssign(i2.value().toString());
                        return i1;
                }
        }
        public class DoBoolean2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        i1.forceAssign(i2.value().toString());
                        return i1;
                }
        }


}
```

BooleanVariable.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public class BooleanVariable extends PostalVariable<Boolean> {

	public BooleanVariable(String name, Boolean val) {
		super(Boolean.TYPE, name, val);
	}

}
```

DivideOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
```

```
 *
 */
public final class DivideOperations {

	private static DivideOperations _instance;
	public static DivideOperations instance() {
		if (_instance == null) {
			_instance = new DivideOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() / i2.value());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() / i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() / (i2.value()?1:-1));
		}
	}
	public class DoString2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() /
Integer.parseInt(i2.value()));
		}
	}

	public class DoDouble2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createDouble("", i1.value() / i2.value());
		}
	}
	public class DoInt2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createDouble("", i1.value() /
i2.value().doubleValue());
		}
```

```
	}
	public class DoBoolean2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createDouble("", i1.value() / (i2.value() ? 1.0 :
-1.0));
		}
	}
	public class DoString2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createDouble("", i1.value() /
Double.parseDouble(i2.value()));
		}
	}
}
```

DoubleVariable.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public class DoubleVariable extends PostalVariable<Double> {

	public DoubleVariable(String name, Double val) {
		super(Double.TYPE, name, val);
	}
}
```

EQOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
```

```
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class EQOperations {

	private static EQOperations _instance;
	public static EQOperations instance() {
		if (_instance == null) {
			_instance = new EQOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() ==
i2.value().intValue());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() ==
i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value() != 0 ? true : false) ==
i2.value().booleanValue());
		}
	}
	public class DoString2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
```

```
                        IntegerVariable i1 = (IntegerVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().intValue() ==
Integer.parseInt(i2.value()));
                }
        }

        public class DoDouble2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() ==
i2.value().doubleValue());
                }
        }
        public class DoInt2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() ==
i2.value().doubleValue());
                }
        }
        public class DoBoolean2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() ==
(i2.value() ? 1.0 : 0.0));
                }
        }
        public class DoString2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() ==
Double.parseDouble(i2.value()));
                }
        }

        public class DoBoolean2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().booleanValue() ==
i2.value().booleanValue());
                }
        }
        public class DoInteger2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().booleanValue() ==
(i2.value() == 0 ? false : true));
                }
        }
        public class DoDouble2Boolean extends OpClass implements IBinaryOp {
```

```
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().booleanValue() ==
(i2.value().intValue() == 0 ? false : true));
		}
	}
	public class DoString2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().booleanValue() ==
i2.value().equalsIgnoreCase("false")|i2.value().equalsIgnoreCase("nil") ? false : true);
		}
	}

	public class DoString2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("",
i1.value().equalsIgnoreCase(i2.value()));
		}
	}
	public class DoInteger2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("",
i1.value().equalsIgnoreCase(i2.value().toString()));
		}
	}
	public class DoDouble2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("",
i1.value().equalsIgnoreCase(i2.value().toString()));
		}
	}
	public class DoBoolean2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("",
i1.value().equalsIgnoreCase(i2.value().toString()));
		}
	}


}
```

GTEOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
```

```
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class GTEOperations {


	private static GTEOperations _instance;
	public static GTEOperations instance() {
		if (_instance == null) {
			_instance = new GTEOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() >=
i2.value().intValue());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() >=
i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() >=
(i2.value() ? 1 : 0));
		}
	}
	public class DoString2Int extends OpClass implements IBinaryOp {
```

```
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() >=
Integer.parseInt(i2.value()));
		}
	}

	public class DoDouble2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() >=
i2.value().doubleValue());
		}
	}
	public class DoInt2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() >=
i2.value().doubleValue());
		}
	}
	public class DoBoolean2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() >=
(i2.value() ? 1.0 : 0.0));
		}
	}
	public class DoString2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() >=
Double.parseDouble(i2.value()));
		}
	}

	public class DoBoolean2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value()?1:0) >= (i2.value()?
1:0));
		}
	}
	public class DoInteger2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value() ? 1 : 0) >=
i2.value().intValue());
		}
	}
```

```
	public class DoDouble2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value()?1:0) >=
i2.value().intValue());
		}
	}
	public class DoString2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value()?1:0) >=
(Boolean.parseBoolean(i2.value())?1:0));
		}
	}

	public class DoString2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().length() >=
i2.value().length());
		}
	}
	public class DoInteger2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().length()  >=
i2.value().toString().length());
		}
	}
	public class DoDouble2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().length() >=
i2.value().toString().length());
		}
	}
	public class DoBoolean2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().length() >=
i2.value().toString().length());
		}
	}


}
```

GTOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
```

```
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public class GTOperations {


	private static GTOperations _instance;
	public static GTOperations instance() {
		if (_instance == null) {
			_instance = new GTOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() >
i2.value().intValue());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() >
i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() > (i2.value() ?
1 : 0));
		}
	}
```

```
	public class DoString2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() >
Integer.parseInt(i2.value()));
		}
	}

	public class DoDouble2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() >
i2.value().doubleValue());
		}
	}
	public class DoInt2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() >
i2.value().doubleValue());
		}
	}
	public class DoBoolean2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() >
(i2.value() ? 1.0 : 0.0));
		}
	}
	public class DoString2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() >
Double.parseDouble(i2.value()));
		}
	}

	public class DoBoolean2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value()?1:0) > (i2.value()?1:0));
		}
	}
	public class DoInteger2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value() ? 1 : 0) >
i2.value().intValue());
		}
	}
```

```
	public class DoDouble2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value()?1:0) >
i2.value().intValue());
		}
	}
	public class DoString2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value()?1:0) >
(Boolean.parseBoolean(i2.value())?1:0));
		}
	}

	public class DoString2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().length() >
i2.value().length());
		}
	}
	public class DoInteger2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().length()  >
i2.value().toString().length());
		}
	}
	public class DoDouble2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().length() >
i2.value().toString().length());
		}
	}
	public class DoBoolean2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().length() >
i2.value().toString().length());
		}
	}


}
```

IbinaryOp.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
```

```
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * Represents a binary operation.
 * @author pnalyvayko
 *
 */
public interface IBinaryOp {

	/**
	 * Performs a binary operation using the two input arguments and returns
	 * a result of the operation.
	 * @param lhs
	 * @param rhs
	 * @return
	 */
	IPostalType invoke(PostalVariable lhs, PostalVariable rhs);
}
```

IbyRef.java

```
package WS4115.Projects.Postal.Types;

/**
 * A signature interface which, when implemented by a class, indicates
 * that by-reference copy must be used instead of by-value.
 * @author pnalyvayko
 *
 */
public interface IByRef {

}
```

IntegerVariable.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
```

```
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class IntegerVariable extends PostalVariable< Integer > {

	public IntegerVariable(String name, int val) {
		super(int.class, name, val);
	}
}
```

IntrinsicLinkValue.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public class IntrinsicLinkValue implements IPostalLinkResolver {

	private IPostalType _intrinsicValue;

	public IntrinsicLinkValue(IPostalType intrinsicValue) {
```

```
                _intrinsicValue = intrinsicValue;
        }
        /* (non-Javadoc)
         * @see
WS4115.Projects.Postal.Types.IPostalLinkResolver#resolve(WS4115.Projects.Postal.Types.PostalLink)
         */
        public IPostalType resolve(PostalLink lhs) {
                // TODO Auto-generated method stub
                return _intrinsicValue;
        }

}
```

IpostalListStrategy.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public interface IPostalListStrategy extends Cloneable {

        int size(PostalList lhs);
        IPostalType get(PostalList lhs, int pos);
        int add(PostalList lhs, IPostalType item);
        void insert(PostalList lhs, int pos, IPostalType item);
        void removeAt(PostalList lhs, int pos);
        void clear(PostalList lhs);
        IPostalType assign(PostalList lhs, IPostalType rhs);
        boolean canAssign(PostalList lhs, IPostalType rhs);
        IPostalListStrategy clone(PostalList lhs) throws CloneNotSupportedException;
        Object[] toArray(PostalList lhs);
        IPostalListStrategy tail(PostalList lhs);
        String toString(PostalList lhs);
        IPostalListStrategy head(PostalList lhs);
}
```

IpostalSetStrategy.java

```
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public interface IPostalSetStrategy {

	int size(PostalSet lhs);
	IPostalType assign(PostalSet lhs, PostalSet rhs);
	void put(PostalSet lhs, String key, IPostalType elem);
	IPostalType get(PostalSet lhs, String key);
	PostalList keys(PostalSet lhs);
	PostalList values(PostalSet lhs);
	void remove(PostalSet lhs, String key);
	IPostalSetStrategy clone(PostalSet lhs) throws CloneNotSupportedException;
	boolean containsKey(PostalSet lhs, String key);
	String toString(PostalSet lhs);
}
```

IpostalType.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
package WS4115.Projects.Postal.Types;

/**
 * IPostalType interface is a base interface for any class used by the POSTAL compiler.
 * The derived classes represent various aspects of the system, such as scopes,  functions,
 * lists and variables.
 * @author pnalyvayko
 *
 */
public interface IPostalType extends Cloneable {

	/**
	 * Returns a name of the instance.
	 * @return
	 */
```

```
        public String getName();
        /**
         * Assigns the rhs object to the current instance. The object returned by the method is used to
verify
         * if the assignment has succeeded.
         * @param rhs
         * @return if the return value's isValid method returns false, then the
         * assignment has failed.
         */
        public IPostalType assign(IPostalType rhs);
        /**
         * Returns true if the rhs object can be assigned to the current instance.
         * @param rhs
         * @return
         */
        public boolean canAssign(IPostalType rhs);
        /**
         * Returns the instance's kind.
         * @return
         */
        public PostalTypeKind getKind();
        /**
         * Returns whether the instance is valid.
         * @return
         */
        public boolean isValid();
        /**
         * Creates a deep copy of the object.
         * @return
         */
        public IPostalType clone() throws CloneNotSupportedException;
        /**
         * Returns true if the instance is Nil.
         * @return
         */
        public boolean isNil();
}
```

IpostalVariable.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
```

```
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public interface IPostalVariable {

	Class getVarClass();
}
```

IunaryOp.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public interface IUnaryOp {

	/**
	 * Performs an unary operation using the single input argument and returns
	 * the result of the operation. Unary operations can be type conversions,
	 * unary arithmetic operations, etc.
	 * @param lhs
	 * @return
	 */
	IPostalType invoke(PostalVariable lhs);
}
```

LTEOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
```

```
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class LTEOperations {


	private static LTEOperations _instance;
	public static LTEOperations instance() {
		if (_instance == null) {
			_instance = new LTEOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() <=
i2.value().intValue());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() <=
i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() <=
(i2.value() ? 1 : 0));
		}
	}
	public class DoString2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
```

```
                        IntegerVariable i1 = (IntegerVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().intValue() <=
Integer.parseInt(i2.value()));
                }
        }

        public class DoDouble2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() <=
i2.value().doubleValue());
                }
        }
        public class DoInt2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() <=
i2.value().doubleValue());
                }
        }
        public class DoBoolean2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() <=
(i2.value() ? 1.0 : 0.0));
                }
        }
        public class DoString2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() <=
Double.parseDouble(i2.value()));
                }
        }

        public class DoBoolean2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createBoolean("", (i1.value()?1:0) <= (i2.value()?
1:0));
                }
        }
        public class DoInteger2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createBoolean("", (i1.value() ? 1 : 0) <=
i2.value().intValue());
                }
        }
        public class DoDouble2Boolean extends OpClass implements IBinaryOp {
```

```
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createBoolean("", (i1.value()?1:0) <=
i2.value().intValue());
                }
        }
        public class DoString2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", (i1.value()?1:0) <=
(Boolean.parseBoolean(i2.value())?1:0));
                }
        }

        public class DoString2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().length() <=
i2.value().length());
                }
        }
        public class DoInteger2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().length()  <=
i2.value().toString().length());
                }
        }
        public class DoDouble2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().length() <=
i2.value().toString().length());
                }
        }
        public class DoBoolean2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().length() <=
i2.value().toString().length());
                }
        }



}
```

LTOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
```

```
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class LTOperations {


	private static LTOperations _instance;
	public static LTOperations instance() {
		if (_instance == null) {
			_instance = new LTOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() <
i2.value().intValue());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() <
i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() < (i2.value() ?
1 : 0));
		}
	}
	public class DoString2Int extends OpClass implements IBinaryOp {
```

```
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() <
Integer.parseInt(i2.value()));
		}
	}

	public class DoDouble2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() <
i2.value().doubleValue());
		}
	}
	public class DoInt2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() <
i2.value().doubleValue());
		}
	}
	public class DoBoolean2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() <
(i2.value() ? 1.0 : 0.0));
		}
	}
	public class DoString2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() <
Double.parseDouble(i2.value()));
		}
	}

	public class DoBoolean2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value()?1:0) < (i2.value()?1:0));
		}
	}
	public class DoInteger2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value() ? 1 : 0) <
i2.value().intValue());
		}
	}
	public class DoDouble2Boolean extends OpClass implements IBinaryOp {
```

```
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createBoolean("", (i1.value()?1:0) <
i2.value().intValue());
                }
        }
        public class DoString2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", (i1.value()?1:0) <
(Boolean.parseBoolean(i2.value())?1:0));
                }
        }

        public class DoString2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().length() <
i2.value().length());
                }
        }
        public class DoInteger2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().length()  <
i2.value().toString().length());
                }
        }
        public class DoDouble2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().length() <
i2.value().toString().length());
                }
        }
        public class DoBoolean2String extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        StringVariable i1 = (StringVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().length() <
i2.value().toString().length());
                }
        }
}
```

MulOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
```

```
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class MulOperations {

	private static MulOperations _instance;
	public static MulOperations instance() {
		if (_instance == null) {
			_instance = new MulOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() * i2.value());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() * i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() * (i2.value()?1:0));
		}
	}
	public class DoString2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() *
Integer.parseInt(i2.value()));
		}
```

```
	}

	public class DoDouble2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createDouble("", i1.value() * i2.value());
		}
	}
	public class DoInt2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createDouble("", i1.value() *
i2.value().doubleValue());
		}
	}
	public class DoBoolean2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createDouble("", i1.value() * (i2.value() ? 1.0 :
0.0));
		}
	}
	public class DoString2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createDouble("", i1.value() *
Double.parseDouble(i2.value()));
		}
	}
}
```

NEQOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;
```

```
/**
 * @author pnalyvayko
 *
 */
public final class NEQOperations {

	private static NEQOperations _instance;
	public static NEQOperations instance() {
		if (_instance == null) {
			_instance = new NEQOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() !=
i2.value().intValue());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() !=
i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", (i1.value() != 0 ? true : false) !=
i2.value().booleanValue());
		}
	}
	public class DoString2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().intValue() !=
Integer.parseInt(i2.value()));
		}
	}

	public class DoDouble2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			DoubleVariable i1 = (DoubleVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().doubleValue() !=
i2.value().doubleValue());
		}
	}
	public class DoInt2Double extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
```

```
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() !=
i2.value().doubleValue());
                }
        }
        public class DoBoolean2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() !=
(i2.value() ? 1.0 : 0.0));
                }
        }
        public class DoString2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().doubleValue() !=
Double.parseDouble(i2.value()));
                }
        }

        public class DoBoolean2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().booleanValue() !=
i2.value().booleanValue());
                }
        }
        public class DoInteger2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().booleanValue() !=
(i2.value() == 0 ? false : true));
                }
        }
        public class DoDouble2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().booleanValue() !=
(i2.value().intValue() == 0 ? false : true));
                }
        }
        public class DoString2Boolean extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        BooleanVariable i1 = (BooleanVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createBoolean("", i1.value().booleanValue() !=
(i2.value().equalsIgnoreCase("false")|i2.value().equalsIgnoreCase("nil")?false:true));
                }
        }

        public class DoString2String extends OpClass implements IBinaryOp {
```

```
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			StringVariable i2 = (StringVariable)rhs;
			return PostalTypeFactory.createBoolean("", !
i1.value().equalsIgnoreCase(i2.value()));
		}
	}
	public class DoInteger2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createBoolean("", !
i1.value().equalsIgnoreCase(i2.value().toString()));
		}
	}
	public class DoDouble2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createBoolean("", !
i1.value().equalsIgnoreCase(i2.value().toString()));
		}
	}
	public class DoBoolean2String extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			StringVariable i1 = (StringVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", !
i1.value().equalsIgnoreCase(i2.value().toString()));
		}
	}
}
```

OpClass.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
package WS4115.Projects.Postal.Types;

/**
 * Represents a base type for all operations arithmetic, logical and conversion operations
 * that can be performed using the language.
 * @author pnalyvayko
 *
```

```
 */
public abstract class OpClass {

}
```

OROperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class OROperations {

	private static OROperations _instance;
	public static OROperations instance() {
		if (_instance == null) {
			_instance = new OROperations();
		}
		return _instance;
	}
	public class DoBoolean2Boolean extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			BooleanVariable i1 = (BooleanVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createBoolean("", i1.value().booleanValue() ||
i2.value().booleanValue());
		}
	}
}
```

PostalEnv.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
```

```
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;
import WS4115.Projects.Postal.builtin.*;
import java.util.*;
import antlr.*;
import java.lang.reflect.Method;
//import java.lang.reflect.Type;
/**
 * @author pnalyvayko
 *
 */
public final class PostalEnv {

	private static PostalEnv _instance;
	private Stack<ScopeLinkedItem> _scopes;
	private Object _interpreter;
	private boolean _returnRequested;

	public final class ScopeLinkedItem {
		private PostalScope _scope;
		private ScopeLinkedItem _parent;

		public ScopeLinkedItem(PostalScope scope) {
			_scope = scope;
			_parent = null;		}
		public ScopeLinkedItem(PostalScope scope, ScopeLinkedItem parent) {
			_scope = scope;
			_parent = parent;
		}
		public PostalScope getScope() { return _scope; }
		public ScopeLinkedItem getParent() { return _parent; }
	}

	protected PostalEnv() {
		_scopes = new Stack<ScopeLinkedItem>();
		PostalScope scope;
		_scopes.push(new ScopeLinkedItem(scope =
(PostalScope)PostalTypeFactory.createScope("GLOBALSCOPE")));
		scope.putFlags(PostalScopeFlags.eNone);

		_returnRequested = false;

		// populate the global scope with buil-in functions
		BuiltinFunctionProvider builtinProvider = new BuiltinFunctionProvider();
		try
```

```
                {
                        Method mi;
                        ReflectedFunction f;

                        // println function
                        mi = builtinProvider.getClass().getMethod("println", IPostalType.class);
                        f = new ReflectedFunction("println", builtinProvider, mi);
                        scope.put(f);

                        // print function
                        mi = builtinProvider.getClass().getMethod("print", IPostalType.class);
                        f = new ReflectedFunction("print", builtinProvider, mi);
                        scope.put(f);

                        // tail function
                        mi = builtinProvider.getClass().getMethod("tail", IPostalType.class);
                        f = new ReflectedFunction("tail", builtinProvider, mi);
                        scope.put(f);

                        // head function
                        mi = builtinProvider.getClass().getMethod("head", IPostalType.class);
                        f = new ReflectedFunction("head", builtinProvider, mi);
                        scope.put(f);

                        // list function
                        mi = builtinProvider.getClass().getMethod("list");
                        f = new ReflectedFunction("list", builtinProvider, mi);
                        scope.put(f);

                        // hashtable function
                        mi = builtinProvider.getClass().getMethod("hashtable");
                        f = new ReflectedFunction("hashtable", builtinProvider, mi);
                        scope.put(f);

                        // words function
                        mi = builtinProvider.getClass().getMethod("words", IPostalType.class);
                        f = new ReflectedFunction("words", builtinProvider, mi);
                        scope.put(f);

                        // nbest function
                        mi = builtinProvider.getClass().getMethod("nbest", IPostalType.class);
                        f = new ReflectedFunction("nbest", builtinProvider, mi);
                        scope.put(f);

                        // openf function
                        mi = builtinProvider.getClass().getMethod("openf", IPostalType.class);
                        f = new ReflectedFunction("openf", builtinProvider, mi);
                        scope.put(f);

                        // savef function
                        mi = builtinProvider.getClass().getMethod("savef", IPostalType.class,
IPostalType.class);
                        f = new ReflectedFunction("savef", builtinProvider, mi);
                        scope.put(f);

                        // cons function
```

```
                    mi = builtinProvider.getClass().getMethod("cons", IPostalType.class,
IPostalType.class);
                    f = new ReflectedFunction("cons", builtinProvider, mi);
                    scope.put(f);

                    // pop function
                    mi = builtinProvider.getClass().getMethod("pop", IPostalType.class);
                    f = new ReflectedFunction("pop", builtinProvider, mi);
                    scope.put(f);

                    // length function
                    mi = builtinProvider.getClass().getMethod("list_length", IPostalType.class);
                    f = new ReflectedFunction("length", builtinProvider, mi);
                    scope.put(f);

                    // isList function
                    mi = builtinProvider.getClass().getMethod("isList", IPostalType.class);
                    f = new ReflectedFunction("islist", builtinProvider, mi);
                    scope.put(f);

                    // nth function
                    mi = builtinProvider.getClass().getMethod("nth", IPostalType.class,
IPostalType.class);
                    f = new ReflectedFunction("nth", builtinProvider, mi);
                    scope.put(f);

                    // DEBUG: assert function
                    mi = builtinProvider.getClass().getMethod("doassert", IPostalType.class);
                    f = new ReflectedFunction("assert", builtinProvider, mi);
                    scope.put(f);

                    // Symbol tables
                    mi = this.getClass().getMethod("printSymbolTables");
                    f = new ReflectedFunction("symboltable", this, mi);
                    scope.put(f);

                    // Dup
                    mi =  builtinProvider.getClass().getMethod("dup", IPostalType.class);
                    f = new ReflectedFunction("dup", builtinProvider, mi);
                    scope.put(f);

                    // isLambda
                    mi =  builtinProvider.getClass().getMethod("isLambda", IPostalType.class);
                    f = new ReflectedFunction("islambda", builtinProvider, mi);
                    scope.put(f);

                    // isNumeric
                    mi =  builtinProvider.getClass().getMethod("isNumeric", IPostalType.class);
                    f = new ReflectedFunction("isnumeric", builtinProvider, mi);
                    scope.put(f);

                    // isLiteral
                    mi =  builtinProvider.getClass().getMethod("isLiteral", IPostalType.class);
                    f = new ReflectedFunction("isliteral", builtinProvider, mi);
                    scope.put(f);
```

```
                        // assoc
                        mi =  builtinProvider.getClass().getMethod("assoc", IPostalType.class,
IPostalType.class);
                        f = new ReflectedFunction("assoc", builtinProvider, mi);
                        scope.put(f);

                        // sassoc
                        mi =  builtinProvider.getClass().getMethod("sassoc", IPostalType.class,
IPostalType.class, IPostalType.class);
                        f = new ReflectedFunction("sassoc", builtinProvider, mi);
                        scope.put(f);
                }
                catch(NoSuchMethodException ex) {
                        // eat the exception
                        System.out.println("EXCEPTION:" + ex.getLocalizedMessage());
                }
        }

        public static PostalEnv instance() {
                if (_instance == null) {
                        _instance = new PostalEnv();
                }
                return _instance;
        }

        public static void clear() {
                _instance = null;
        }

        public void ReportError(String errMsg) throws RecognitionException {
                throw new RecognitionException(errMsg);
        }

        public IPostalType findSymbol(String name, boolean parentLookup) {

                ScopeLinkedItem last = _scopes.lastElement();
                while (last != null) {
                        PostalScope scope = last.getScope();
                        if (scope.contains(name)) {
                                return scope.get(name);
                        }
                        if (parentLookup)
                                last = last.getParent();
                        else
                                last = null;
                }
                String msg = "Symbol \"" + name + "\" not found.";
                return PostalTypeFactory.createError(msg);
        }

        public void setInterpreter(Object interpreter) {
                _interpreter = interpreter;
        }

        public Object getInterpreter() {
                return _interpreter;
```

```
	}
	/**
	 * The method is used to create a new scope parented to the last scope. Use this function
	 * when calling lambda expressions (anonymous functions), block statements, etc.
	 * @param scope
	 */
	public void enterScope(PostalScope scope) {
		_scopes.push(new ScopeLinkedItem(scope, _scopes.lastElement()));
	}
	/**
	 * Function scope is always linked to the global scope as its parent scope.
	 * @param scope
	 */
	public void enterFunctionScope(PostalScope scope) {
		_scopes.push(new ScopeLinkedItem(scope, _scopes.firstElement()));
	}

	public PostalScope exitScope() throws IllegalStateException {

		// There always must be at least one scope
		if (!_scopes.empty() && _scopes.size() > 1) {
			ScopeLinkedItem last = _scopes.pop();
			return last.getScope();
		}
		else {
			throw new IllegalStateException();
		}
	}
	/**
	 * Returns the scope that was last added to the context
	 * @return
	 */
	public PostalScope getCurrentScope() {
		ScopeLinkedItem last = _scopes.peek();
		return last.getScope();
	}
	/**
	 * Returns the global scope object
	 * @return
	 */
	public PostalScope getRootScope() {
		return _scopes.firstElement().getScope();
	}

	public boolean isReturnAllowed() {
		ScopeLinkedItem last = _scopes.lastElement();
		while (last != null) {
			PostalScope scope = last.getScope();
			last = last.getParent();
			PostalScopeFlags flags = scope.getFlags();
			if (flags == PostalScopeFlags.eTransparent)
				continue;
			if (flags == PostalScopeFlags.eReturnAllowed)
				return true;
			break;
```

```
                }
                return false;
        }

        public void requestReturn() {
                _returnRequested = true;
        }

        public void resetReturn() {
                _returnRequested = false;
        }

        public boolean isReturnRequested() {
                return _returnRequested;
        }

        public IPostalType printSymbolTables() {
                PostalList list = (PostalList)PostalTypeFactory.createList("SYMBOL_LIST");

                ScopeLinkedItem last = _scopes.lastElement();
                while (last != null) {
                        PostalScope scope = last.getScope();
                        last = last.getParent();

                        Enumeration<String> symbols = scope.keys();
                        while(symbols.hasMoreElements()) {
                                IPostalType elem = scope.get(symbols.nextElement());
                                PostalList elelem =
(PostalList)PostalTypeFactory.createList("SYMBOL_ELEM");
                                elelem.add(PostalTypeFactory.createString("SCOPE", scope.getName()));
                                elelem.add(PostalTypeFactory.createString("SYMBOL", "'" + elem.getName()
+ "'" ));
                                elelem.add(PostalTypeFactory.createString("VALUE", "'" + elem.toString()
+ "'"));
                                list.add(elelem);
                        }
                }
                return list;
        }

}
```

PostalError.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
```

```
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public class PostalError implements IPostalType {

	private String _errMsg;
	private String _errSource;

	public PostalError(String errMsg) {
		_errMsg = errMsg;
		_errSource = "";
	}

	public String getName() {
		return "Error Object";
	}

	public PostalError(String errMsg, String errSource) {
		_errMsg = errMsg;
		_errSource = errSource;
	}
	public String getMessage() {
		return _errMsg;
	}
	public String getSource() {
		return _errSource;
	}
	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalType#assign(WS4115.Projects.Postal.Types.IPostalType)
	 */
	public IPostalType assign(IPostalType rhs) {
		if (rhs != null && rhs.getKind() == PostalTypeKind.eKindError) {
			PostalError rhsError = (PostalError)rhs;
			_errMsg = rhsError._errMsg;
			_errSource = rhsError._errSource;
		}
		return this;
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalType#canAssign(WS4115.Projects.Postal.Types.IPostalType)
	 */
	public boolean canAssign(IPostalType rhs) {
		if (rhs != null && rhs.getKind() == PostalTypeKind.eKindError) {
			return true;
		}
```

```
                return false;
        }

        /* (non-Javadoc)
         * @see WS4115.Projects.Postal.Types.IPostalType#getKind()
         */
        public PostalTypeKind getKind() {
                return PostalTypeKind.eKindError;
        }

        /* (non-Javadoc)
         * @see WS4115.Projects.Postal.Types.IPostalType#isValid()
         */
        public boolean isValid() {
                return false;
        }

        public IPostalType clone() throws CloneNotSupportedException {

                Object cloned = super.clone();

                PostalError e = (PostalError)cloned;
                e._errMsg = _errMsg;
                e._errSource = _errSource;

                return (IPostalType)cloned;
        }

        public boolean isNil() {
                return false;
        }

        public String toString() {
                return _errMsg;
        }
}
```

PostalFunction.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
package WS4115.Projects.Postal.Types;
```

```
public abstract class PostalFunction implements IPostalType {

	private String _name;

	public PostalFunction(String name) {
		_name = name;
	}

	public String getName() {
		return _name;
	}
	/**
	 * Returns a list of arguments.
	 * @return
	 */
	public abstract PostalList arguments();

	public PostalTypeKind getKind() {
		// TODO Auto-generated method stub
		return PostalTypeKind.eKindFunction;
	}

	public IPostalType assign(IPostalType rhs) {
		// Assign simply returns the current instance
		return this;
	}

	public boolean canAssign(IPostalType rhs) {
		return false;
	}

	public boolean isValid() {
		return true;
	}

	public IPostalType clone() throws CloneNotSupportedException {

		Object cloned = super.clone();

		//PostalFunction f = (PostalFunction)cloned;

		return (IPostalType)cloned;
	}

	public boolean isNil() {
		return false;
	}
}
```

PostalFunctionInterpreter.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
```

```
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class PostalFunctionInterpreter extends PostalFunction {

	private PostalList _argList;
	private Object _implementation;

	public PostalFunctionInterpreter(String name, PostalList argList, Object impl) {
		super(name);
		_argList = argList;
		_implementation = impl;
	}

	public PostalList arguments() {
		return _argList;
	}

	public Object getImplementation() {
		return _implementation;
	}

	public String toString() {
		return "User-defined function: \"" + this.getName() + "\"";
	}
}
```

PostalHashtableStrategy.java

```
/**
 *
 */
package WS4115.Projects.Postal.Types;

import java.util.Map;
import java.util.Hashtable;

/**
 * @author pnalyvayko
```

```
 *
 */
public class PostalHashtableStrategy implements IPostalSetStrategy {

	private Map<String, IPostalType> _map;

	public PostalHashtableStrategy() {
		_map = new Hashtable<String, IPostalType>();
	}

	public IPostalType assign(PostalSet lhs, PostalSet rhs) {
		return lhs.reconstruct(rhs);
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalSetStrategy#clone(WS4115.Projects.Postal.Types.PostalSet)
	 */
	public IPostalSetStrategy clone(PostalSet lhs)
			throws CloneNotSupportedException {
		// TODO Auto-generated method stub
		PostalHashtableStrategy cloned = (PostalHashtableStrategy)super.clone();
		cloned._map = new Hashtable<String, IPostalType>();

		for(String key : _map.keySet()) {
			cloned._map.put(key, _map.get(key));
		}

		return cloned;
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalSetStrategy#get(WS4115.Projects.Postal.Types.PostalSet,
java.lang.String)
	 */
	public IPostalType get(PostalSet lhs, String key) {
		// TODO Auto-generated method stub
		return _map.get(key);
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalSetStrategy#keys(WS4115.Projects.Postal.Types.PostalSet)
	 */
	public PostalList keys(PostalSet lhs) {
		// TODO Auto-generated method stub
		PostalList l = (PostalList)PostalTypeFactory.createList("KEYS_LIST");
		for(String key:_map.keySet())
			l.add(PostalTypeFactory.createString("KEY", key));
		return l;
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalSetStrategy#put(WS4115.Projects.Postal.Types.PostalSet,
java.lang.String, WS4115.Projects.Postal.Types.IPostalType)
```

```
	 */
	public void put(PostalSet lhs, String key, IPostalType elem) {
		// TODO Auto-generated method stub

		_map.put(key, elem);
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalSetStrategy#remove(WS4115.Projects.Postal.Types.PostalSet,
java.lang.String)
	 */
	public void remove(PostalSet lhs, String key) {
		// TODO Auto-generated method stub
		_map.remove(key);
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalSetStrategy#size(WS4115.Projects.Postal.Types.PostalSet)
	 */
	public int size(PostalSet lhs) {
		// TODO Auto-generated method stub
		return _map.size();
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalSetStrategy#values(WS4115.Projects.Postal.Types.PostalSet)
	 */
	public PostalList values(PostalSet lhs) {
		PostalList l = (PostalList)PostalTypeFactory.createList("KEYS_LIST");
		for(IPostalType value:_map.values())
			l.add(value);
		return l;
	}

	public boolean containsKey(PostalSet lhs, String key) {
		return _map.containsKey(key);
	}

	public String toString(PostalSet lhs) {
		StringBuilder sb = new StringBuilder();
		for(Map.Entry<String, IPostalType> entry : _map.entrySet())
		{
			if (sb.length() != 0)
				sb.append(' ');
			sb.append('(');
			sb.append(entry.getKey());
			sb.append(' ');
			sb.append(entry.getValue());
			sb.append(')');
		}
		sb.insert(0, '(');
		sb.append(")");
		return sb.toString();
```

```
	}

}
```

PostalLink.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class PostalLink implements IPostalType {

	private String _name;
	private IPostalLinkResolver _resolver;

	public PostalLink(String name, IPostalLinkResolver resolver) {
		_name = name;
		_resolver = resolver;
	}

	public IPostalType resolve() {
		if (_resolver != null) {
			return _resolver.resolve(this);
		}
		return null;
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalType#assign(WS4115.Projects.Postal.Types.IPostalType)
	 */
	public IPostalType assign(IPostalType rhs) {
		// TODO Auto-generated method stub
		//System.out.println("assignining  " + rhs.toString());
		if (!canAssign(rhs))
		{
			//System.out.println("Cannot assign...");
```

```
                        return this;
                }
                if (_resolver != null) {
                        IPostalType resolved = _resolver.resolve(this);
                        if (resolved != null) {
                                if (rhs instanceof PostalLink) {
                                        rhs = ((PostalLink)rhs).resolve();
                                        if (rhs == null) {
                                                //System.out.println("Invalid link...");
                                                return PostalTypeFactory.createError("Invalid link.");
                                        }
                                }
                                if (resolved instanceof PostalNilPtr) {
                                        try
                                        {
                                                IPostalType copy;
                                                if (!(rhs instanceof IByRef)) {
                                                        copy = rhs.clone();
                                                }
                                                else {
                                                        //System.out.println("Detected IByRef interface.");
                                                        copy = rhs;
                                                }
                                                _resolver = new IntrinsicLinkValue(copy);
                                        }
                                        catch(CloneNotSupportedException ex)
                                        {
                                                _resolver = new
IntrinsicLinkValue(PostalTypeFactory.createError("Cannot clone the right side."));
                                        }
                                        return rhs;
                                } else {
                                        return resolved.assign(rhs);
                                }
                        }
                }
                return this;
        }

        /* (non-Javadoc)
         * @see
WS4115.Projects.Postal.Types.IPostalType#canAssign(WS4115.Projects.Postal.Types.IPostalType)
         */
        public boolean canAssign(IPostalType rhs) {
                IPostalType resolved = resolve();
                if (rhs instanceof PostalLink) {
                        rhs = ((PostalLink)rhs).resolve();
                        if (rhs == null)
                                return false;
                }
                return resolved != null ? (resolved instanceof PostalNilPtr) ||
resolved.canAssign(rhs) : false;
        }

        /* (non-Javadoc)
         * @see WS4115.Projects.Postal.Types.IPostalType#getKind()
```

```
	 */
	public PostalTypeKind getKind() {
		// TODO Auto-generated method stub
		return PostalTypeKind.eKindLink;
	}

	/* (non-Javadoc)
	 * @see WS4115.Projects.Postal.Types.IPostalType#getName()
	 */
	public String getName() {
		// TODO Auto-generated method stub
		return _name;
	}

	/* (non-Javadoc)
	 * @see WS4115.Projects.Postal.Types.IPostalType#isNil()
	 */
	public boolean isNil() {
		// TODO Auto-generated method stub
		if (_resolver != null) {
			IPostalType resolved = _resolver.resolve(this);
			return resolved != null ? resolved.isNil() : false;
		}
		return false;
	}

	/* (non-Javadoc)
	 * @see WS4115.Projects.Postal.Types.IPostalType#isValid()
	 */
	public boolean isValid() {
		// TODO Auto-generated method stub
		if (_resolver != null) {
			IPostalType resolved = _resolver.resolve(this);
			return resolved != null ? resolved.isValid() : false;
		}
		return false;
	}

	public IPostalType clone() throws CloneNotSupportedException {
		PostalLink cloned = (PostalLink)this.clone();

		if (_resolver != null) {
			IPostalType innerValue = _resolver.resolve(this);
			_resolver = new IntrinsicLinkValue(innerValue.clone());
		}
		return cloned;
	}

	public String toString() {
		if (_resolver != null) {
			IPostalType resolved = _resolver.resolve(this);
			return resolved.toString();
		}
		return super.toString();
	}
```

```
}
```

PostalLinkedListStrategy.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
package WS4115.Projects.Postal.Types;

import java.util.LinkedList;
import java.lang.StringBuilder;

public class PostalLinkedListStrategy implements IPostalListStrategy {

	private LinkedList<IPostalType> _list;
	private int _offset;

	public PostalLinkedListStrategy() {
		_offset = 0;
		_list = new LinkedList<IPostalType>();
	}

	public PostalLinkedListStrategy(LinkedList<IPostalType> masterList, int offset) {
		_list = masterList;
		_offset = offset;
	}

	public int add(PostalList lhs, IPostalType item) {
		_list.add(item);
		return _list.size() - 1;
	}

	public IPostalType assign(PostalList lhs, IPostalType rhs) {
		IPostalType result = lhs;
		if (canAssign(lhs, rhs)) {

			try
			{
				PostalList rhsList = (PostalList)rhs;
				((PostalList)lhs).reconstruct(rhsList);
			}
			catch(CloneNotSupportedException ex) {
				result = PostalTypeFactory.createError("Cannot reconstruct the list.");
			}
```

```
                return result;

        }
        return PostalTypeFactory.createError("Cannot assign incompatible types.");
    }

    public boolean canAssign(PostalList lhs, IPostalType rhs) {
        if (rhs != null && rhs instanceof PostalList) {
                return true;
        }
        return false;
    }

    public void clear(PostalList lhs) {
        while (_offset < _list.size())
                _list.removeLast();

    }

    public IPostalListStrategy clone(PostalList lhs) throws CloneNotSupportedException {
        PostalLinkedListStrategy cloned = (PostalLinkedListStrategy)super.clone();

        cloned._list =new LinkedList<IPostalType>();

        for(IPostalType t : _list) {
                cloned._list.add(t.clone());
        }

        return cloned;
    }

    public IPostalType get(PostalList lhs, int pos) {
        if (pos + _offset >= _list.size())
                return PostalTypeFactory.createNil();
        return _list.get(pos + _offset);
    }

    public void insert(PostalList lhs, int pos, IPostalType item) {
        _list.add(pos + _offset, item);
    }

    public void removeAt(PostalList lhs, int pos) {
        _list.remove(pos + _offset);
    }

    public int size(PostalList lhs) {
        return _list.size() - _offset;
    }

    public Object[] toArray(PostalList lhs) {
        Object[] copy = new Object[_list.size() - _offset];
        for (int i = _offset; i < _list.size(); i++) {
                copy[i - _offset] = _list.get(i);
        }
        return copy;
```

```
	}

	public IPostalListStrategy tail(PostalList lhs) {
		return new PostalLinkedListStrategy(_list, _offset + 1);
	}

	public IPostalListStrategy head(PostalList lhs) {
		return new PostalLinkedListStrategy(_list, 0);
	}

	public String toString(PostalList lhs) {
		StringBuilder sb = new StringBuilder();
		for(int i = _offset; i < _list.size(); i++) {
			IPostalType t = _list.get(i);
			if (sb.length() != 0)
				sb.append(' ');
			sb.append(t.toString());
		}
		sb.insert(0, '(');
		sb.append(")");
		return sb.toString();
	}

}
```

PostalList.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
package WS4115.Projects.Postal.Types;

public class PostalList implements IPostalType, IByRef {

	private String _name;
	private IPostalListStrategy _strat;

	public PostalList(String name, IPostalListStrategy strat) {
		_name = name;
		_strat = strat;
	}

	public int size() {
		return _strat.size(this);
```

```
	}

	public IPostalType get(int pos) {
		return _strat.get(this, pos);
	}

	public int add(IPostalType item) {
		return _strat.add(this, item);
	}

	public PostalList reconstruct(PostalList lhs) throws CloneNotSupportedException {
		_strat = lhs._strat;
		return this;
	}

	public void insert(int pos, IPostalType item) {
		_strat.insert(this, pos, item);
	}

	public void removeAt(int pos) {
		_strat.removeAt(this, pos);
	}

	public void clear() {
		_strat.clear(this);
	}

	public String getName() {
		return _name;
	}

	public IPostalType assign(IPostalType rhs) {
		return _strat.assign(this, rhs);
	}

	public boolean canAssign(IPostalType rhs) {
		return _strat.canAssign(this, rhs);
	}

	public PostalTypeKind getKind() {
		// TODO Auto-generated method stub
		return PostalTypeKind.eKindList;
	}

	public boolean isValid() {
		return _strat != null ? true : false;
	}

	public IPostalType clone() throws CloneNotSupportedException {
		Object cloned = super.clone();

		PostalList l = (PostalList)cloned;
		l._strat = _strat.clone(this);

		return l;
```

```
	}

	public boolean isNil() {
		return false;
	}

	public Object[] toArray() {
		return _strat.toArray(this);
	}

	public PostalList tail() {
		return new PostalList("Tail of " + this.getName(), _strat.tail(this));
	}

	public PostalList head() {
		return new PostalList("Head of " + this.getName(), _strat.head(this));
	}

	public String toString() {
		return _strat.toString(this);
	}
}
```

PostalNilPtr.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class PostalNilPtr extends BooleanVariable {

	public PostalNilPtr() {
		super("NIL", false);
	}
	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalType#assign(WS4115.Projects.Postal.Types.IPostalType)
```

```
         */
        public IPostalType assign(IPostalType rhs) {
                return this;
        }

        /* (non-Javadoc)
         * @see
WS4115.Projects.Postal.Types.IPostalType#canAssign(WS4115.Projects.Postal.Types.IPostalType)
         */
        public boolean canAssign(IPostalType rhs) {
                return true;
        }

        /* (non-Javadoc)
         * @see WS4115.Projects.Postal.Types.IPostalType#isValid()
         */
        public boolean isValid() {
                // TODO Auto-generated method stub
                return true;
        }

        public IPostalType clone() throws CloneNotSupportedException {
                return (IPostalType)super.clone();
        }

        public boolean isNil() {
                return true;
        }

        public String toString() {
                return "nil";
        }
}
```

PostalOpCodes.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
```

```
 * @author pnalyvayko
 *
 */
public enum PostalOpCodes {
	eAssign,
	eGT,
	eLT,
	eGTE,
	eLTE,
	eEQ,
	eNEQ,
	eMinus,
	ePlus,
	eDivide,
	eMultiply,
	ePostfixIncrement,
	ePostfixDecrement,
	eUnaryMinus,
	eAND,
	eOR,
}
```

PostalOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

import java.util.*;
/**
 * @author pnalyvayko
 *
 */
public final class PostalOperations {

	private static PostalOperations _instance;
	private Dictionary<UnaryOpKey, OpClass> _conversions;

	public class UnaryOpKey {
		protected Class _c1;
```

```
		protected PostalOpCodes _opCode;
		public UnaryOpKey(PostalOpCodes opCode, Class c1) {
			_c1 = c1;
			_opCode = opCode;
		}
		public PostalOpCodes OpCode() { return _opCode; }
		public Class c1() { return _c1; }
	}

	public final class BinaryOpKey extends UnaryOpKey {

		private Class _c2;
		public BinaryOpKey(PostalOpCodes opCode, Class c1, Class c2) {

			super(opCode, c1);
			_c2 = c2;
		}
		public Class c2() { return _c2; }
		public int hashCode() {
			String s = _c1.getCanonicalName() + ":" + _c2.getCanonicalName() + ":" +
_opCode;
			return s.hashCode();
		}
		public boolean equals(Object rhs) {
			if (rhs instanceof BinaryOpKey) {
				BinaryOpKey rhsKey = (BinaryOpKey)rhs;
				boolean b = _c1 == rhsKey._c1 && _c2 == rhsKey._c2 && _opCode ==
rhsKey._opCode;
				return b;
			}
			return false;
		}
	}

	protected PostalOperations() {
		_conversions = new Hashtable<UnaryOpKey, OpClass>();
		// Assignment operations
		// Integer
		_conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Integer.TYPE, Integer.TYPE),
AssignOperations.instance().new DoInt2Int());
		_conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Integer.TYPE, Double.TYPE),
AssignOperations.instance().new DoDouble2Int());
		_conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Integer.TYPE, Boolean.TYPE),
AssignOperations.instance().new DoBoolean2Int());
		_conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Integer.TYPE, String.class),
AssignOperations.instance().new DoString2Int());
		// Double
		_conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Double.TYPE, Double.TYPE),
AssignOperations.instance().new DoDouble2Double());
		_conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Double.TYPE, Integer.TYPE),
AssignOperations.instance().new DoInt2Double());
		_conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Double.TYPE, Boolean.TYPE),
AssignOperations.instance().new DoBoolean2Double());
		_conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Double.TYPE, String.class),
AssignOperations.instance().new DoString2Double());
		// Boolean
		_conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Boolean.TYPE, Boolean.TYPE),
AssignOperations.instance().new DoBoolean2Boolean());
```

```
                _conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Boolean.TYPE, Integer.TYPE),
AssignOperations.instance().new DoInteger2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Boolean.TYPE, Double.TYPE),
AssignOperations.instance().new DoDouble2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, Boolean.TYPE, String.class),
AssignOperations.instance().new DoString2Boolean());
                // String
                _conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, String.class, String.class),
AssignOperations.instance().new DoString2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, String.class, Integer.TYPE),
AssignOperations.instance().new DoInteger2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, String.class, Double.TYPE),
AssignOperations.instance().new DoDouble2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eAssign, String.class, Boolean.TYPE),
AssignOperations.instance().new DoBoolean2String());
                //
                // Equality operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Integer.TYPE, Integer.TYPE),
EQOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Integer.TYPE, Double.TYPE),
EQOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Integer.TYPE, Boolean.TYPE),
EQOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Integer.TYPE, String.class),
EQOperations.instance().new DoString2Int());
                // Double
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Double.TYPE, Double.TYPE),
EQOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Double.TYPE, Integer.TYPE),
EQOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Double.TYPE, Boolean.TYPE),
EQOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Double.TYPE, String.class),
EQOperations.instance().new DoString2Double());
                // Boolean
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Boolean.TYPE, Boolean.TYPE),
EQOperations.instance().new DoBoolean2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Boolean.TYPE, Integer.TYPE),
EQOperations.instance().new DoInteger2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Boolean.TYPE, Double.TYPE),
EQOperations.instance().new DoDouble2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, Boolean.TYPE, String.class),
EQOperations.instance().new DoString2Boolean());
                // String
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, String.class, String.class),
EQOperations.instance().new DoString2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, String.class, Integer.TYPE),
EQOperations.instance().new DoInteger2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, String.class, Double.TYPE),
EQOperations.instance().new DoDouble2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eEQ, String.class, Boolean.TYPE),
EQOperations.instance().new DoBoolean2String());
                //
                // Inequality operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Integer.TYPE, Integer.TYPE),
NEQOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Integer.TYPE, Double.TYPE),
```

```
NEQOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Integer.TYPE, Boolean.TYPE),
NEQOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Integer.TYPE, String.class),
NEQOperations.instance().new DoString2Int());
                // Double
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Double.TYPE, Double.TYPE),
NEQOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Double.TYPE, Integer.TYPE),
NEQOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Double.TYPE, Boolean.TYPE),
NEQOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Double.TYPE, String.class),
NEQOperations.instance().new DoString2Double());
                // Boolean
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Boolean.TYPE, Boolean.TYPE),
NEQOperations.instance().new DoBoolean2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Boolean.TYPE, Integer.TYPE),
NEQOperations.instance().new DoInteger2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Boolean.TYPE, Double.TYPE),
NEQOperations.instance().new DoDouble2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, Boolean.TYPE, String.class),
NEQOperations.instance().new DoString2Boolean());
                // String
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, String.class, String.class),
NEQOperations.instance().new DoString2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, String.class, Integer.TYPE),
NEQOperations.instance().new DoInteger2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, String.class, Double.TYPE),
NEQOperations.instance().new DoDouble2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eNEQ, String.class, Boolean.TYPE),
NEQOperations.instance().new DoBoolean2String());
                //
                // Greater-than operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Integer.TYPE, Integer.TYPE),
GTOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Integer.TYPE, Double.TYPE),
GTOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Integer.TYPE, Boolean.TYPE),
GTOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Integer.TYPE, String.class),
GTOperations.instance().new DoString2Int());
                // Double
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Double.TYPE, Double.TYPE),
GTOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Double.TYPE, Integer.TYPE),
GTOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Double.TYPE, Boolean.TYPE),
GTOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Double.TYPE, String.class),
GTOperations.instance().new DoString2Double());
                // Boolean
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Boolean.TYPE, Boolean.TYPE),
GTOperations.instance().new DoBoolean2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Boolean.TYPE, Integer.TYPE),
GTOperations.instance().new DoInteger2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Boolean.TYPE, Double.TYPE),
GTOperations.instance().new DoDouble2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, Boolean.TYPE, String.class),
GTOperations.instance().new DoString2Boolean());
```

```
                // String
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, String.class, String.class),
GTOperations.instance().new DoString2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, String.class, Integer.TYPE),
GTOperations.instance().new DoInteger2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, String.class, Double.TYPE),
GTOperations.instance().new DoDouble2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGT, String.class, Boolean.TYPE),
GTOperations.instance().new DoBoolean2String());
                //
                // Greater-or-equal-than operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Integer.TYPE, Integer.TYPE),
GTEOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Integer.TYPE, Double.TYPE),
GTEOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Integer.TYPE, Boolean.TYPE),
GTEOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Integer.TYPE, String.class),
GTEOperations.instance().new DoString2Int());
                // Double
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Double.TYPE, Double.TYPE),
GTEOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Double.TYPE, Integer.TYPE),
GTEOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Double.TYPE, Boolean.TYPE),
GTEOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Double.TYPE, String.class),
GTEOperations.instance().new DoString2Double());
                // Boolean
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Boolean.TYPE, Boolean.TYPE),
GTEOperations.instance().new DoBoolean2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Boolean.TYPE, Integer.TYPE),
GTEOperations.instance().new DoInteger2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Boolean.TYPE, Double.TYPE),
GTEOperations.instance().new DoDouble2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, Boolean.TYPE, String.class),
GTEOperations.instance().new DoString2Boolean());
                // String
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, String.class, String.class),
GTEOperations.instance().new DoString2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, String.class, Integer.TYPE),
GTEOperations.instance().new DoInteger2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, String.class, Double.TYPE),
GTEOperations.instance().new DoDouble2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eGTE, String.class, Boolean.TYPE),
GTEOperations.instance().new DoBoolean2String());
                //
                // Less-than operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Integer.TYPE, Integer.TYPE),
LTOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Integer.TYPE, Double.TYPE),
LTOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Integer.TYPE, Boolean.TYPE),
LTOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Integer.TYPE, String.class),
LTOperations.instance().new DoString2Int());
                // Double
```

```
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Double.TYPE, Double.TYPE),
LTOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Double.TYPE, Integer.TYPE),
LTOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Double.TYPE, Boolean.TYPE),
LTOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Double.TYPE, String.class),
LTOperations.instance().new DoString2Double());
                // Boolean
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Boolean.TYPE, Boolean.TYPE),
LTOperations.instance().new DoBoolean2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Boolean.TYPE, Integer.TYPE),
LTOperations.instance().new DoInteger2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Boolean.TYPE, Double.TYPE),
LTOperations.instance().new DoDouble2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, Boolean.TYPE, String.class),
LTOperations.instance().new DoString2Boolean());
                // String
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, String.class, String.class),
LTOperations.instance().new DoString2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, String.class, Integer.TYPE),
LTOperations.instance().new DoInteger2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, String.class, Double.TYPE),
LTOperations.instance().new DoDouble2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLT, String.class, Boolean.TYPE),
LTOperations.instance().new DoBoolean2String());
                //
                // Less-or-equal-than operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Integer.TYPE, Integer.TYPE),
LTEOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Integer.TYPE, Double.TYPE),
LTEOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Integer.TYPE, Boolean.TYPE),
LTEOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Integer.TYPE, String.class),
LTEOperations.instance().new DoString2Int());
                // Double
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Double.TYPE, Double.TYPE),
LTEOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Double.TYPE, Integer.TYPE),
LTEOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Double.TYPE, Boolean.TYPE),
LTEOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Double.TYPE, String.class),
LTEOperations.instance().new DoString2Double());
                // Boolean
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Boolean.TYPE, Boolean.TYPE),
LTEOperations.instance().new DoBoolean2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Boolean.TYPE, Integer.TYPE),
LTEOperations.instance().new DoInteger2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Boolean.TYPE, Double.TYPE),
LTEOperations.instance().new DoDouble2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, Boolean.TYPE, String.class),
LTEOperations.instance().new DoString2Boolean());
                // String
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, String.class, String.class),
LTEOperations.instance().new DoString2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, String.class, Integer.TYPE),
LTEOperations.instance().new DoInteger2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, String.class, Double.TYPE),
```

```
LTEOperations.instance().new DoDouble2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eLTE, String.class, Boolean.TYPE),
LTEOperations.instance().new DoBoolean2String());
                //
                // Subtraction operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMinus, Integer.TYPE, Integer.TYPE),
SubtractionOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMinus, Integer.TYPE, Double.TYPE),
SubtractionOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMinus, Integer.TYPE, Boolean.TYPE),
SubtractionOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMinus, Integer.TYPE, String.class),
SubtractionOperations.instance().new DoString2Int());
                // Double
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMinus, Double.TYPE, Double.TYPE),
SubtractionOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMinus, Double.TYPE, Integer.TYPE),
SubtractionOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMinus, Double.TYPE, Boolean.TYPE),
SubtractionOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMinus, Double.TYPE, String.class),
SubtractionOperations.instance().new DoString2Double());
                //
                // Addition operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, Integer.TYPE, Integer.TYPE),
AdditionOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, Integer.TYPE, Double.TYPE),
AdditionOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, Integer.TYPE, Boolean.TYPE),
AdditionOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, Integer.TYPE, String.class),
AdditionOperations.instance().new DoString2Int());
                // Double
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, Double.TYPE, Double.TYPE),
AdditionOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, Double.TYPE, Integer.TYPE),
AdditionOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, Double.TYPE, Boolean.TYPE),
AdditionOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, Double.TYPE, String.class),
AdditionOperations.instance().new DoString2Double());
                // String
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, String.class, String.class),
AdditionOperations.instance().new DoString2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, String.class, Integer.TYPE),
AdditionOperations.instance().new DoInteger2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, String.class, Double.TYPE),
AdditionOperations.instance().new DoDouble2String());
                _conversions.put(new BinaryOpKey(PostalOpCodes.ePlus, String.class, Boolean.TYPE),
AdditionOperations.instance().new DoBoolean2String());
                //
                // Multiplication operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMultiply, Integer.TYPE, Integer.TYPE),
MulOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMultiply, Integer.TYPE, Double.TYPE),
```

```
MulOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMultiply, Integer.TYPE, Boolean.TYPE),
MulOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMultiply, Integer.TYPE, String.class),
MulOperations.instance().new DoString2Int());
                // Double
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMultiply, Double.TYPE, Double.TYPE),
MulOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMultiply, Double.TYPE, Integer.TYPE),
MulOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMultiply, Double.TYPE, Boolean.TYPE),
MulOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eMultiply, Double.TYPE, String.class),
MulOperations.instance().new DoString2Double());
                //
                // Division operations
                //
                // Integer
                _conversions.put(new BinaryOpKey(PostalOpCodes.eDivide, Integer.TYPE, Integer.TYPE),
DivideOperations.instance().new DoInt2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eDivide, Integer.TYPE, Double.TYPE),
DivideOperations.instance().new DoDouble2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eDivide, Integer.TYPE, Boolean.TYPE),
DivideOperations.instance().new DoBoolean2Int());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eDivide, Integer.TYPE, String.class),
DivideOperations.instance().new DoString2Int());
                // Double
                _conversions.put(new BinaryOpKey(PostalOpCodes.eDivide, Double.TYPE, Double.TYPE),
DivideOperations.instance().new DoDouble2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eDivide, Double.TYPE, Integer.TYPE),
DivideOperations.instance().new DoInt2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eDivide, Double.TYPE, Boolean.TYPE),
DivideOperations.instance().new DoBoolean2Double());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eDivide, Double.TYPE, String.class),
DivideOperations.instance().new DoString2Double());

                //
                // Postfix increment, decrement and unary minus need to be implemented here.
                //

                // Conjunction and disjunction operations
                _conversions.put(new BinaryOpKey(PostalOpCodes.eAND, Boolean.TYPE, Boolean.TYPE),
ANDOperations.instance().new DoBoolean2Boolean());
                _conversions.put(new BinaryOpKey(PostalOpCodes.eOR, Boolean.TYPE, Boolean.TYPE),
OROperations.instance().new DoBoolean2Boolean());
        }

        public static PostalOperations instance() {
                if (_instance == null) {
                        _instance = new PostalOperations();
                }
                return _instance;
        }

        public IPostalType removeLink(IPostalType link) {
                if (link instanceof PostalLink)
                        return ((PostalLink)link).resolve();
                return link;
        }
```

```
	private boolean binaryOpExists(PostalOpCodes opCode, IPostalType lhs, IPostalType rhs) {
		lhs = removeLink(lhs);
		rhs = removeLink(rhs);
		if (lhs != null && rhs != null && lhs.isValid() && rhs.isValid() &&
				lhs.getKind() == PostalTypeKind.eKindValue && rhs.getKind() ==
PostalTypeKind.eKindValue)
		{
			PostalVariable lhsVar = (PostalVariable)lhs;
			PostalVariable rhsVar = (PostalVariable)rhs;
			OpClass op = _conversions.get(new BinaryOpKey(opCode, lhsVar.getVarClass(),
rhsVar.getVarClass()));
			return (op != null);
		}
		else
		{
			return false;
		}
	}

	private IPostalType binary_op(PostalOpCodes opCode, IPostalType lhs, IPostalType rhs) {
		// Resolve links
		lhs = removeLink(lhs);
		rhs = removeLink(rhs);
		if (lhs == null || rhs == null)
			return PostalTypeFactory.createError("Either left or right operand is null.");

		PostalVariable lhsVar = null;
		PostalVariable rhsVar = null;
		if (lhs != null && rhs != null && lhs.isValid() && rhs.isValid() &&
				lhs.getKind() == PostalTypeKind.eKindValue && rhs.getKind() ==
PostalTypeKind.eKindValue)
		{
			lhsVar = (PostalVariable)lhs;
			rhsVar = (PostalVariable)rhs;
		}
		else if (rhs != null && lhs != null && lhs.isValid() && lhs.getKind() ==
PostalTypeKind.eKindValue)
		{
			lhsVar = (PostalVariable)lhs;
			rhsVar = (PostalVariable)PostalTypeFactory.createString("", rhs.toString());
		}
		if (lhsVar != null && rhsVar != null)
		{
			OpClass op = _conversions.get(new BinaryOpKey(opCode, lhsVar.getVarClass(),
rhsVar.getVarClass()));
			if (op != null && (op instanceof IBinaryOp)) {
				IBinaryOp binaryOp = (IBinaryOp)op;
				return binaryOp.invoke(lhsVar, rhsVar);
			}
			else if (op == null)
			{
				lhs = PostalTypeFactory.createError("Operation \"" + opCode + "\" is not
supported.");
				System.out.println(((PostalError)lhs).getMessage());
			}
			else
```

```
                        {
                                lhs = PostalTypeFactory.createError("Specified operation \"" + opCode +
"\"is not a binary op.");
                                System.out.println(((PostalError)lhs).getMessage());
                        }
                        return lhs;
                }
                else
                {
                        return PostalTypeFactory.createError(opCode.toString() + " operation is not
allowed.");
                }
        }

        private IPostalType unary_op(PostalOpCodes opCode, IPostalType lhs) {
                lhs = removeLink(lhs);
                // Resolve links
                if (lhs == null)
                        return PostalTypeFactory.createError("The input operand is null.");
                PostalVariable lhsVar = null;
                if (lhs != null && lhs.isValid() &&  lhs.getKind() == PostalTypeKind.eKindValue)
                {
                        lhsVar = (PostalVariable)lhs;
                }
                else if (lhs != null)
                {
                        lhsVar = (PostalVariable)PostalTypeFactory.createString("", lhs.toString());
                }
                if (lhsVar != null)
                {
                        OpClass op = _conversions.get(new UnaryOpKey(opCode, lhsVar.getVarClass()));
                        if (op instanceof IUnaryOp) {
                                IUnaryOp unaryOp = (IUnaryOp)op;
                                return unaryOp.invoke(lhsVar);
                        }
                        return lhs;
                }
                else
                {
                        return PostalTypeFactory.createError(opCode.toString() + " operation is not
allowed.");
                }
        }

        public IPostalType convert(IPostalType lhs, Class c) {
                // Resolve links
                if (lhs == null)
                        return PostalTypeFactory.createError("The input operand is null.");
                if (lhs.getKind() == PostalTypeKind.eKindLink) {
                        lhs = ((PostalLink)lhs).resolve();
                }
                PostalVariable lhsVar = null;
                if (lhs != null && lhs.isValid() &&  lhs.getKind() == PostalTypeKind.eKindValue)
                {
                        lhsVar = (PostalVariable)lhs;
                }
```

```
		else  if (lhs != null)
		{
			lhsVar = (PostalVariable)PostalTypeFactory.createString("", lhs.toString());
		}
		if (lhsVar != null)
		{
			OpClass op = _conversions.get(new BinaryOpKey(PostalOpCodes.eAssign, c,
lhsVar.getVarClass()));
			if (op instanceof IBinaryOp)
			{
				IBinaryOp binaryOp = (IBinaryOp)op;

				PostalVariable result = null;
				if (c == Integer.TYPE)
					result = (PostalVariable)PostalTypeFactory.createInteger("", 0);
				else if (c == Double.TYPE)
					result = (PostalVariable)PostalTypeFactory.createDouble("", 0.0);
				else if (c == String.class)
					result = (PostalVariable)PostalTypeFactory.createString("", "");
				else if (c == Boolean.TYPE)
					result = (PostalVariable)PostalTypeFactory.createBoolean("",
true);
				else {
					return PostalTypeFactory.createError("Unsupported type.");
				}
				if (result != null) {
					return binaryOp.invoke(result, lhsVar);
				}
			}
		}
		else
		{
			return PostalTypeFactory.createError("No conversion exists.");
		}
		return lhs;
	}

	public IPostalType assign(IPostalType lhs, IPostalType rhs) {
		return binary_op(PostalOpCodes.eAssign, lhs, rhs);
	}
	public boolean canAssign(IPostalType lhs, IPostalType rhs) {
		return binaryOpExists(PostalOpCodes.eAssign, lhs, rhs);
	}
	public IPostalType greaterThan(IPostalType lhs, IPostalType rhs) {
		return binary_op(PostalOpCodes.eGT, lhs, rhs);
	}
	public IPostalType greaterOrEqualThan(IPostalType lhs, IPostalType rhs) {
		return binary_op(PostalOpCodes.eGTE, lhs, rhs);
	}
	public IPostalType lessThan(IPostalType lhs, IPostalType rhs) {
		return binary_op(PostalOpCodes.eLT, lhs, rhs);
	}
	public IPostalType lessOrEqualThan(IPostalType lhs, IPostalType rhs) {
		return binary_op(PostalOpCodes.eLTE, lhs, rhs);
	}
	public IPostalType equalTo(IPostalType lhs, IPostalType rhs) {
```

```
                return binary_op(PostalOpCodes.eEQ, lhs, rhs);
        }
        public IPostalType nonEqualTo(IPostalType lhs, IPostalType rhs) {
                return binary_op(PostalOpCodes.eNEQ, lhs, rhs);
        }
        public IPostalType subtract(IPostalType lhs, IPostalType rhs) {
                return binary_op(PostalOpCodes.eMinus, lhs, rhs);
        }
        public IPostalType add(IPostalType lhs, IPostalType rhs) {
                return binary_op(PostalOpCodes.ePlus, lhs, rhs);
        }
        public IPostalType multiply(IPostalType lhs, IPostalType rhs) {
                return binary_op(PostalOpCodes.eMultiply, lhs, rhs);
        }
        public IPostalType divide(IPostalType lhs, IPostalType rhs) {
                return binary_op(PostalOpCodes.eDivide, lhs, rhs);
        }

        public IPostalType increment(IPostalType lhs) {
                return unary_op(PostalOpCodes.ePostfixIncrement, lhs);
        }

        public IPostalType decrement(IPostalType lhs) {
                return unary_op(PostalOpCodes.ePostfixDecrement, lhs);
        }

        public IPostalType minus(IPostalType lhs) {
                return unary_op(PostalOpCodes.eUnaryMinus, lhs);
        }

        public IPostalType and(IPostalType lhs, IPostalType rhs) {
                return binary_op(PostalOpCodes.eAND, lhs, rhs);
        }

        public IPostalType or(IPostalType lhs, IPostalType rhs) {
                return binary_op(PostalOpCodes.eOR, lhs, rhs);
        }
}
```

PostalScope.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
```

```
/**
 *
 */
package WS4115.Projects.Postal.Types;

import java.util.*;
import java.lang.StringBuilder;

/**
 * Declares a scope which is a container of other objects. The scope can be a global scope, function
scope
 * or a nested scope.
 *
 * @author pnalyvayko
 *
 */
public class PostalScope implements IPostalType {

	private Hashtable<String, IPostalType> _scope;
	private String _name;
	private PostalScopeFlags _flags;

	public PostalScope(String name) {
		_name = name;
		_scope = new Hashtable<String, IPostalType>();
		_flags = PostalScopeFlags.eNone;
	}

	public PostalScopeFlags getFlags() {
		return _flags;
	}
	public void putFlags(PostalScopeFlags flags) {
		_flags = flags;
	}

	public String getName()	{
		return _name;
	}

	public void put(IPostalType var) {
		_scope.put(var.getName(), var);
	}

	public boolean contains(String key) {
		return _scope.containsKey(key);
	}

	public void remove(String key) {
		_scope.remove(key);
	}

	public IPostalType get(String key) {
		return _scope.get(key);
	}

	public int size() {
```

```
		return _scope.size();
	}

	public Enumeration<String> keys() {
		return _scope.keys();
	}

	public PostalTypeKind getKind() {
		return PostalTypeKind.eKindScope;
	}

	public boolean isValid() {
		return true;
	}

	public IPostalType assign(IPostalType rhs) {
		if (rhs instanceof PostalScope) {
			PostalScope rhsScope = (PostalScope)rhs;
			Iterator<String> it = rhsScope._scope.keySet().iterator();
			while (it.hasNext()) {
				String k = it.next();
				if (_scope.containsKey(k)) {
					_scope.get(k).assign(rhsScope.get(k));
				}
			}
			return this;
		}
		return PostalTypeFactory.createError("Expected a scope on the right side.");
	}

	public boolean canAssign(IPostalType rhs) {
		if (rhs instanceof PostalScope)
			return true;
		return false;
	}

	public IPostalType clone() throws CloneNotSupportedException {

		PostalScope s = (PostalScope)super.clone();
		s._scope = new Hashtable<String, IPostalType>();

		Iterator<Map.Entry<String, IPostalType>> it = _scope.entrySet().iterator();

		while(it.hasNext()) {
			Map.Entry<String, IPostalType> entry = it.next();
			s._scope.put(entry.getKey(), entry.getValue().clone());
		}

		return s;
	}

	public boolean isNil() {
		return false;
	}
```

```
	public String toString() {
		Iterator<Map.Entry<String, IPostalType>> it = _scope.entrySet().iterator();

		StringBuilder sb = new StringBuilder();
		while(it.hasNext()) {
			Map.Entry<String, IPostalType> entry = it.next();
			sb.append(" (" + entry.getKey() + " . " + entry.getValue().toString() + ") ");
		}
		return sb.toString();
	}
}
```

PostalScopeFlags.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public enum PostalScopeFlags {
	eNone,
	eTransparent,
	eReturnAllowed,
}
```

PostalSet.java

```
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public class PostalSet implements IPostalType, IByRef {
```

```
	private IPostalSetStrategy _strat;
	private String _name;

	public PostalSet(String name, IPostalSetStrategy strat) {
		_name = name;
		_strat = strat;
	}

	public IPostalType clone() throws CloneNotSupportedException {
		PostalSet cloned = (PostalSet)super.clone();
		cloned._strat = _strat.clone(this);
		return cloned;
	}

	public IPostalType reconstruct(PostalSet rhs) {
		_strat = rhs._strat;
		return this;
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalType#assign(WS4115.Projects.Postal.Types.IPostalType)
	 */
	public IPostalType assign(IPostalType rhs) {
		if (canAssign(rhs)) {
			if (rhs instanceof PostalSet) {
				return _strat.assign(this, (PostalSet)rhs);
			}
		}
		return PostalTypeFactory.createError("The right-hand side is not a set.");
	}

	/* (non-Javadoc)
	 * @see
WS4115.Projects.Postal.Types.IPostalType#canAssign(WS4115.Projects.Postal.Types.IPostalType)
	 */
	public boolean canAssign(IPostalType rhs) {
		if (rhs instanceof PostalLink)
			rhs = ((PostalLink)rhs).resolve();
		if (rhs instanceof PostalSet)
			return true;
		return false;
	}

	/* (non-Javadoc)
	 * @see WS4115.Projects.Postal.Types.IPostalType#getKind()
	 */
	public PostalTypeKind getKind() {
		// TODO Auto-generated method stub
		return PostalTypeKind.eKindSet;
	}

	/* (non-Javadoc)
	 * @see WS4115.Projects.Postal.Types.IPostalType#getName()
	 */
```

```
	public String getName() {
		// TODO Auto-generated method stub
		return _name;
	}

	/* (non-Javadoc)
	 * @see WS4115.Projects.Postal.Types.IPostalType#isNil()
	 */
	public boolean isNil() {
		// TODO Auto-generated method stub
		return false;
	}

	/* (non-Javadoc)
	 * @see WS4115.Projects.Postal.Types.IPostalType#isValid()
	 */
	public boolean isValid() {
		// TODO Auto-generated method stub
		return _strat != null ? true : false;
	}

	public int size() {
		return _strat.size(this);
	}
	public void put(String key, IPostalType elem) {
		_strat.put(this, key, elem);
	}
	public IPostalType get(String key) {
		return _strat.get(this, key);
	}
	public PostalList keys() {
		return _strat.keys(this);
	}
	public PostalList values() {
		return _strat.values(this);
	}
	public void remove(String key) {
		_strat.remove(this, key);
	}

	public boolean containsKey(String key) {
		return _strat.containsKey(this, key);
	}

	public String toString() {
		return _strat.toString(this);
	}
}
```

PostalTypeFactory.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
```

```
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;
import java.lang.reflect.Method;
//import java.lang.reflect.Type;

/**
 * @author pnalyvayko
 *
 */
public final class PostalTypeFactory {

	public static IPostalType createInteger(String name, Integer val) {
		return new IntegerVariable(name, val);
	}
	public static IPostalType createDouble(String name, Double val) {
		return new DoubleVariable(name, val);
	}
	public static IPostalType createBoolean(String name, Boolean val) {
		return new BooleanVariable(name, val);
	}
	public static IPostalType createString(String name, String val) {
		return new StringVariable(name, val);
	}
	public static IPostalType createError(String errMsg) {
		return new PostalError(errMsg);
	}
	public static IPostalType createError(String errMsg, String errSource) {
		return new PostalError(errMsg, errSource);
	}
	public static IPostalType createObject(String name, Object val) {
		if (val instanceof Integer) {
			return createInteger(name, (Integer)val);
		}
		else if (val instanceof Double) {
			return createDouble(name, (Double)val);
		}
		else if (val instanceof Boolean) {
			return createBoolean(name, (Boolean)val);
		}
		else if (val instanceof String) {
			return createString(name, (String)val);
		}
		return new PostalVariable<Object>(Object.class, name, val);
```

```
	}
	public static IPostalType createScope(String scopeName) {
		return new PostalScope(scopeName);
	}

	public static IPostalType createNil() {
		return new PostalNilPtr();
	}

	public static IPostalType createList(String listName) {
		return new PostalList(listName, new PostalLinkedListStrategy());
	}

	public static IPostalType createFunctionInterpreter(String name, PostalList argList, Object
implementation) {
		return new PostalFunctionInterpreter(name, argList, implementation);
	}

	public static IPostalType createVariable(String name) {
		return new PostalLink(name, new IntrinsicLinkValue(PostalTypeFactory.createNil()));
	}

	public static IPostalType createVariable(String name, IPostalType intrinsicValue) {
		if (intrinsicValue instanceof PostalLink)
			intrinsicValue = ((PostalLink)intrinsicValue).resolve();
		return new PostalLink(name, new IntrinsicLinkValue(intrinsicValue));
	}

	public static IPostalType createReflectedFunction(String name, Object instance, Method mi) {
		return new ReflectedFunction(name, instance, mi);
	}

	public static IPostalType createMap(String name) {
		return new PostalSet(name, new PostalHashtableStrategy());
	}
}
```

PostalTypeKind.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
```

```
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public enum PostalTypeKind {
	eKindInvalid,
	eKindValue,
	eKindFunction,
	eKindList,
	eKindSet,
	eKindError,
	eKindScope,
	eKindNil,
	eKindLink
}
```

PostalVariable.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public class PostalVariable<T> implements IPostalType, IPostalVariable {

	private final String _name;
	private T _value;
	private Class<T> _class;

	public PostalVariable(Class<T> c, String n, T value) {
		_name = n;
		_value = value;
		_class = c;
	}
```

```
	public Class getVarClass() {
		return _class;
	}

	public boolean isValid() {
		return true;
	}

	public String getName() {
		return _name;
	}
	public T value() {
		return _value;
	}
	public void forceAssign(T value) {
		_value = value;
	}

	public PostalTypeKind getKind() {
		return PostalTypeKind.eKindValue;
	}

	public IPostalType assign(IPostalType rhs) {
		if (canAssign(rhs))
		{
			return PostalOperations.instance().assign(this, rhs);
		}
		return this;
	}

	public IPostalType convert(Class c) {
		return PostalOperations.instance().convert(this,  c);
	}

	public boolean canAssign(IPostalType rhs) {
		return PostalOperations.instance().canAssign(this, rhs);
	}

	public IPostalType clone() throws CloneNotSupportedException {
		PostalVariable cloned = (PostalVariable)super.clone();

		return cloned;
	}

	public String toString() {
		return _value.toString();
	}

	public boolean isNil() {
		return false;
	}
}
```

ReflectedFunction.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;
import java.lang.reflect.Method;
//import java.lang.reflect.Type;
import java.lang.reflect.InvocationTargetException;;

/**
 * @author pnalyvayko
 *
 */
public class ReflectedFunction extends PostalFunction {

	private Object _instance;
	private Method _mi;

	public ReflectedFunction(String name, Object instance, Method mi) {
		super(name);
		_instance = instance;
		_mi = mi;
	}

	public PostalList arguments() {
		Class[] parameterTypes = _mi.getParameterTypes();
		PostalList paramList = (PostalList)PostalTypeFactory.createList("ARG_LIST");
		//UNDONE: fetch the parameter information and populate the argument list
		for (int i = 0; i < parameterTypes.length; i++)
			paramList.add(PostalTypeFactory.createString("ARGITEM", ""));
		return paramList;
	}

	public IPostalType invoke(PostalList argList) throws InvocationTargetException,
IllegalAccessException {

		if (_mi.getReturnType() != IPostalType.class)
			return PostalTypeFactory.createError("Invalid return type.");

		Class[] parameterTypes = _mi.getParameterTypes();
		if (parameterTypes.length != argList.size())
```

```
                        return PostalTypeFactory.createError("Specified number of parameters \"" +
argList.size() + "\" does not match expected \"" + parameterTypes.length + "\"");

                // Validate the parameters and their types
                for(int i = 0; i < parameterTypes.length; i++) {
                        Class t = parameterTypes[i];
                        IPostalType arg = argList.get(i);
                        if (!t.isInstance(arg)) {
                                return PostalTypeFactory.createError("Expected \"" +
t.getClass().getCanonicalName() + "\", but was given \"" + arg.getClass().getCanonicalName() + "\"");
                        }
                }
                return (IPostalType)_mi.invoke(_instance, argList.toArray());
        }

        public String toString() {
                return "Reflected function \"" + this.getName() + "\"";
        }
}
```

StringVariable.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public class StringVariable extends PostalVariable<String> {

        public StringVariable(String name, String val) {
                super(String.class, name, val);
        }
}
```

SubtractionOperations.java

```
// This is a part of the POSTAL language package.
// Copyright (C) Peter Nalyvayko (c) 2008
```

```
// All rights reserved.
//
// Programming Languages and Translators
// COMS WS4115
// Prof. Stephen Edwards
//
// This source code is only intended as a supplement to the
// POSTAL Language Reference and related
// electronic documentation provided with the language.
// See these sources for detailed information regarding the
// POSTAL Language product.
//
/**
 *
 */
package WS4115.Projects.Postal.Types;

/**
 * @author pnalyvayko
 *
 */
public final class SubtractionOperations {


	private static SubtractionOperations _instance;
	public static SubtractionOperations instance() {
		if (_instance == null) {
			_instance = new SubtractionOperations();
		}
		return _instance;
	}

	public class DoInt2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			IntegerVariable i2 = (IntegerVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() - i2.value());
		}
	}
	public class DoDouble2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			DoubleVariable i2 = (DoubleVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() - i2.value().intValue());
		}
	}
	public class DoBoolean2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
			IntegerVariable i1 = (IntegerVariable)lhs;
			BooleanVariable i2 = (BooleanVariable)rhs;
			return PostalTypeFactory.createInteger("", i1.value() - (i2.value()?1:0));
		}
	}
	public class DoString2Int extends OpClass implements IBinaryOp {
		public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
```

```
                        IntegerVariable i1 = (IntegerVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createInteger("", i1.value() -
Integer.parseInt(i2.value()));
                }
        }

        public class DoDouble2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        DoubleVariable i2 = (DoubleVariable)rhs;
                        return PostalTypeFactory.createDouble("", i1.value() - i2.value());
                }
        }
        public class DoInt2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        IntegerVariable i2 = (IntegerVariable)rhs;
                        return PostalTypeFactory.createDouble("", i1.value() -
i2.value().doubleValue());
                }
        }
        public class DoBoolean2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        BooleanVariable i2 = (BooleanVariable)rhs;
                        return PostalTypeFactory.createDouble("", i1.value() - (i2.value() ? 1.0 :
0.0));
                }
        }
        public class DoString2Double extends OpClass implements IBinaryOp {
                public IPostalType invoke(PostalVariable lhs, PostalVariable rhs) {
                        DoubleVariable i1 = (DoubleVariable)lhs;
                        StringVariable i2 = (StringVariable)rhs;
                        return PostalTypeFactory.createDouble("", i1.value() -
Double.parseDouble(i2.value()));
                }
        }
}
```