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 – Part-of-speech tagging language. The language was designed around the idea to combine the imperative and functional programming constructs to do the natural language processing tasks, including but not limited to the POS tagging.

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:

assert	break	cadr	concat
continue	dolist	dup	equal
fold	function	hash	if
islist	ishash	lambda	list
map	nbest	nil	
numeric	openf	print	println
return	savef	search	sort
true	var	while	words

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.

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.

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**
Subscript operator	[]
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.

Function reference operator '#'

Function reference operator allows to pass a function declared elsewhere in the program as an argument to another function. Examples are such intrinsic functions as **sort** function which take either a lambda expression or a reference to a function as a second argument.

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	-

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.

```
if (expression,
        statement1
         statement2
);
```

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.

```
while (
        expression,
        statement
);
```

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.

```
dolist ( expression,
        lambda(x) { statement } | '#'identifier
)
```

Intrinsics

Intrinsics are expressions defined internally by the language.

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 hash 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:

hash()

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.

The sort function

Evaluates the expression, and, if the evaluation result is a list, sorts its elements using the specified predicate expression. If no predicate was specified, the list is sorted using internal criterion (alphabetically in ascending order). The original list is not affected. The function returns a sorted list copy.

Syntax

sort (expression, predicate)

The filter function

Evaluates the expression, and, if expression evaluates into a list, removes the elements from the list using the specified predicate. The original list is not affected. The function returns a copy of the original list with the same or fewer number of elements.

Syntax

filter(expression, predicate)

The cadr function

Evaluates the expression, and, if the expression evaluates into a list, returns a reference to the element following the first element of the list or nil if the list is invalid, empty or contains a single element.

Syntax

cadr (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 algorithm. 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); // prints (('The' DT 1.0) ('movie' NN 0.999) ('starts' VB 0.89) ('at' IN 1.0) ('seven' CD 0.89) ('.' . 1.0))
```

The dup function

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

Syntax

dup(expression)

The fold function

The function takes the expression, evaluates it to a list and then "folds" the predicate in between the elements of the list.

Syntax

```
fold ( expression, // must evaluate to an instance of list
       exp, // expr is evaluated and used as the first element in the resulting list
       predicate);
```

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

The openf function

I/O operation. Opens a text file and loads the file contents into the given list.
Syntax

```
openf (identifier,
        string | identifier);
```

The map function

Takes a list and a function and applies the function to each element of the list. The original list is affected.
Syntax

```
map (expression, action)
```

Example:

```
list => ('1' '2' '3' '4' '5')
map(list, lambda(x) { return “” + numeric(x) + 1; });
println(list); // prints ('2' '3' '4' '5' '6')
```