SL Language Manual

CS W4115: Programming Languages and Translators

Professor Stephen A. Edwards

Computer Science Department

Submitted by

Majid Khan {UID = mk2759, Email = majidkhan@yahoo.com }

1. Introduction

Search engines are still emerging markets and lack tools which enable building search applications e.g. Search engines, price matching engines, Meta crawlers etc. faster and easier. This search language would provide user capability to create text search applications with lesser efforts. The scope of this project would be limited to text search in HTML and plain text files.

A typical search application should provide

1) Ability to define repositories and maintain repositories
2) Ability to understand document formats and manipulate data inside repositories
3) Ability to gather statistical data based on repository data and then store it for later use.
4) Ability to provide operations on statistical data to perform and produce results

1.1 Glossary

Repository – A repository would be a hierarchy of directories containing documents.

Document – HTML and plain texts would be the file types used as documents. Only files with extensions html, htm, txt and without extensions (would be considered as plain text) would be processed by this language.

Data manipulation – It is a multi step process which is as following

Parse data and get words.

Perform stemming on these words (http://en.wikipedia.org/wiki/Stemming)

Get frequencies of stemmed words and create an inverted vector table using this data.

Calculate term frequency–inverse document frequency (tf-idf) for every stemmed word.

Store this data in for future use.

Search – Search would need two items. Query string and a repository on which we have already performed data manipulation. Query is a series of alphanumeric words separated by white space.

This process would return a list of documents and also a number that represents the closeness of query with document.

2. Lexical conventions

There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and separators. White space would be ignored (would not be converted to any tag) but it may be serving as tag separators. A token is the longest consuctive string not separated by white space and other declared seperators e.g. Comma, semiColon, braces, new line, space, tab etc.

2.1 Comments

C style multiline comments are used i.e. Anything between /* and */ is comment. There is no hierarchy of comments i.e. /* /* */ is one comment while /* /* */ */ would be an error because outermost */ is extra.

2.2 Identifiers

An identifier is a sequence of letters and digits; the first character must be alphabetic. Identifiers are case insensitive. Although it could be controlled but identifiers could be of any length. All part of identifier is significant.

2.3 Keywords

int
null
float
if
else
for
each
string
name
value
document
documentlist
operation

2.4 Types and Literals

There are several kinds of constants, as follows:

2.4.1 Integer

An integer is a sequence of digits and could precede an optional (‘+’ or ‘-‘) character. Integer must be in the range -2147483648 to 2147483647.

2.4.2 Floats

A float consists of an integer part, a decimal point, and a fraction part. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing.

2.4.3 String

A string is a sequence of characters surrounded by double quotes `“`. A string has the type arrayofcharacters (see below) and refers to an area of storage initialized with the given characters. In a string, the character `"` (double quote) must be preceded by a `”` (double quote). Character constant is missing and this could be fulfilled by a string of size 1.

2.4.4 Name Value

A name value is an assignment of string constant to an identified. A name value can not exist at its own (cannot be declared as a variable). It can only be defined when an operation is invoked. The name part is predefined for built in function and can be defined for custom functions by user. Its value have specific patterns (If time permits I would try to introduce regular expression to match this value). The only difference between name

value and oridnary variable that is passed through a function is that a regular expression acn be checked against this inside function (this feature may not be implemented) which would check or reject the value. Each operation or data declaration would define its own required name value types.

A name value cannot be declared at its own in declaration section of program. Once name value is in scope i.e. in a function that contains name value parameters it behaves like identifier and can become lvalue if needed.

2.4.5 Repository

Repository is the base datatype that holds information about data source. It could be defined in two ways, creating from scratch or from some operation on existing repositories.

Repository data type needs three name value fields "location", "inlcude" and "type". Name value "location" contains a value that points to a directory structure (compiler would not validate value string contains a valid directory). Name value "include" contains a string that contains comma separated wildcard file patterns (default value if not provided is "*.html,*.txt,*" which means all files that has extension .html, .txt and without extensions in all subdirectories would be considered as doucments). Name value "type" contains the method which would be use to produce statistical data for search.

Example initialization:

repository arep is location=”c:\adirectory\anotherDirectory”
include=”*.html,*.txt,*.htm,*” type=”cosine”;

2.4.6 Query

Query is the base data type that holds search string, repository reference and where the search result would be stored. It has two name values "query" and "store". Name value "query" is a string that we are searching in repository and "store" is the file name where query results would be stored (if it is already present then file would be overwritten).

Example initialization:

performquery q on arep with query=”search this data please” store=”search.query”
reference=docList;

2.4.7 Document and DocumentList

Document type is not in the scope of current project (I mentioned earlier that if time would permit I will add this). Query result can be returned as a collection of Document which is a builtin type called Document list. A for each loop can extract documents from documentList type.

Result of the query would be sorted in descending order with respect to matching criteria returned by statisitcal measure used. (This is not in the scope).

2.4.8 Block

'{' starts a new code block and matching '}' ends this code block. A block can contain zero or more expressions.

2.4.9 Separator

';' is expression separator. A semicolon without any expression is null statement. Multiple null statements can exist in a sequence. '(' and ')' could be used for expression grouping.

2.4.10 Expression

Following sections are in the order of the precedence of operations.

2.4.10.1 Primary expressions

2.4.10.1.1 Identifier

Identified is a primary expression.

2.4.10.1.2 Constants

Cosntants are primary expressions.

2.4.10.1.3 Function call

A function call is followed by required parameters and list of name values. It also includes some literal strings e.g. "with", "on" or "is" for readability. These literals could be defined when defining a function. For e.g. performquery is defined as

```
operation performquery query q "on", reposiotry arep "with",
                        name value query,
                        name value store,
                        documentList reference
{
                /* body */
};
```

Here "on" and "with" are just added for readability.

2.4.10.1.4 Binary Operators

+, - , * , / are the binary operators supported. * and / have same precedency and its higher than the precedence of + and -.

+ opeartor is defined on strings and results in concatenation of two strings

+ operator is defined on repository and results in addition of two repositories to one.

2.4.10.1.5 Relational operators

All relational operators have same precedence

> (expression > expression) greater than
< (expression < expression) less than
>= (expression >= expression) greater than or equal to
<= (expression <= expression) less than or equal to
<> (expression <> expression) not equal to
not (not (expression = expression)) not equal to (negation of equal to)

2.4.10.1.6 Declaration

Declarations are done at the start of the program. Each declaration is terminated by semicolon.

DECLARATIONLIST : DECLARATION *;

DECLARATION : TYPE IDENTIFIER SEMICOLON ;

2.4.10.1.7 Assignment

An assignment has an identifier that is lvalue and it has a `=` then a statement. This

statement is evaluated and the result is then stored in lvalue.

BINARYSTATEMENT : IDENTIFIER
| BINARYSTATEMENT BINARYOPERATOR
(IDENTIFIER | FUNCTIONCALL);

ASSIGNMENTSTATEMENT : IDENTIFIER '=' (IDENTIFIER | FUNCTIONCALL) (BINARYOPERATOR BINARYSTATEMENT)*;

2.4.10.1.8 If – Else

If statement is a very simple statement which contains a boolean expression that either returns zero or not zero (zero means false and non zero means true). At runtime the boolean statement would be evaluated and if result equals to zero then else would be executed otherwise the first set of statements would be executed.

IFSTATEMENT : 'IF' '(' BOOLEANSTATEMENT ')'
(STATEMENT | '{' STATEMENTS '}')
('ELSE' (STATEMENT | '{' STATEMENTS '}'))?

2.4.10.1.9 For and For each

For has two flavors one that is to execute loop for a range of numbers. For loop variables values can be modified inside the loop (start, end and index variables). Another variation is to loop through the collection of documents (for each).

FORSTATEMENT : 'FOR' (IDENTIFIER '=' NUMBER 'TO' NUMBER
| 'EACH' IDENTIFIER 'IN' IDENTIFIER)
(STATEMENT | '{' STATEMENTS '}');

3. Grammar

DIGIT : ('0'..'9') ;

NUMBER : DIGIT(DIGIT)* ;

DECIMAL : '.' ;

SIGN : ('+'|'-');

ALPHABET : ('A'..'Z') | ('a' .. 'z');

SEMICOLON : ';' ;

BINARYOPERATOR : '+' | '-' | '*' | '/' ;

IDENTIFIER : ALPHABET (DIGIT | ALPHABET)*;

INT : SIGN? NUMBER;

FLOAT : SIGN? (NUMBER (DECIMAL NUMBER?)
| DECIMAL NUMBER) ;

TYPE : 'integer' |
'float' |
'string' |
repository' |
'documentList' |
document'
'query';

DECLARATION : TYPE IDENTIFIER SEMICOLON ;

DECLARATIONLIST : DECLARATION *;

STATEMENT : IFSTATEMENT |
FORSTATEMENT |
ASSIGNMENTSTATEMENT |
FUNCTIONCALL ;

BOOLEANSTATEMENT : 'NOT'? IDENTIFIER BOOLEANOPERATOR
(BOOLEANSTATEMENT | IDENTIFIER) ;

```
IFSTATEMENT        :        'IF' ‘(‘ BOOLEANSTATEMENT ‘)’
                            ( STATEMENT | '{' STATEMENTS '}')
                            ('ELSE' ( STATEMENT | '{' STATEMENTS '}') )?
                            ;

FORSTATEMENT       :        'FOR'  (IDENTIFIER '=' NUMBER 'TO' NUMBER
                            | 'EACH' IDENTIFIER 'IN' IDENTIFIER)
                            ( STATEMENT | '{' STATEMENTS '}');

BINARYSTATEMENT :           IDENTIFIER
                            | BINARYSTATEMENT BINARYOPERATOR
                            (IDENTIFIER | FUNCTIONCALL);

ASSIGNMENTSTATEMENT        :        IDENTIFIER '=' (IDENTIFIER |
FUNCTIONCALL) (BINARYOPERATOR BINARYSTATEMENT)*;

STATEMENTS         :        STATEMENT *;

FUNCTIONDECLARATIONS       : FUNCTIONDECLARATION*       ;

STRINGLITERAL      :        '"' * '"';

TYPEWITHNAMEVALUE  :        TYPE | 'NAME VALUE';

TYPEIDENTIFIERTRAIL        :        (',' TYPEWITHNAMEVALUE IDENTIFIER
STRINGLITERAL?
TYPEIDENTIFIERTRAIL)* ;

FUNCTIONCALL       :        IDENTIFIER (IDENTIFIER IDENTIFIER)*
                             (
                            (IDENTIFIER '=' STRINGLITERAL) |
                            (IDENTIFIER '=' IDENTIFIER)* SEMICOLON ;

FUNCTIONDECLARATION        :        'OPERATION' IDENTIFIER (
TYPEWITHNAMEVALUE IDENTIFIER STRINGLITERAL?
TYPEIDENTIFIERTRAIL)?
                                    '{' STATEMENTS '}'

        ;
PROGRAM
        :        DECLARATIONLIST STATEMENTS* FUNCTIONDECLARATIONS*;
```

4. A Sample

```
/* Program starts with declarations */

Repository arep;
Query aQyery;
int counter;
string q;
string myQuery;
string documentURI;
string documentData;
DocumentList docList;
Document currentDocument;

/* an assignment expression */
q = "assignment";

repository arep is location=”c:\mywebpages” include=”*.html,*.txt,*.htm,*”
type=”cosine”;

/* this initiates the repository and initialize directory location,
file patterns and method of computation */

process arep with location=”c:\adirectory\anotherDirectory” stemmer =”paice”
overrwite=”true” filename=”arep.rep”;

/* this command processes repositories and perform statistical measures that are
required for searching */

/* for loop flavor that runs for integer ranges */
for counter = 1 to 4
{
        myQuery= q + counter
                /* string + int = string (concatenation) */

/* this loop will generate values asssignent1, assignment2, assignment3 and
assignment4 */

        performquery aQyery on arep with queryString=myQuery store=”search.query”
reference=docList;

/* this will search assignment? in repository and store pages in two places
1) search.query 2) docList data structure */
```

```
        /* a sample of if */

        if (docList == null) {
                print " No document was returned ";
        }
        else
        {
                /* for loop flavor that runs on document collections */
                for each currentDocument in docList
{


                        documentURI = getDocumentURI currentDocument;
                        documentData = getDocumentData currentDocument;
                        print documentURI;
                        print documentData;

                }
        }
}
```

5. Builtin functions

getDocumentURI document;

This method returns the URI for the document passed as parameter.

getDocumentData document;

This method returns the data for the document passed as parameter.

Print string;

This method prints the string to console.

repository repository “is”, name value location, name value include, name value type;

This method instantiates the repository.

performquery query “on” , repository “with”, name value queryString, name value store, name value reference;

This method executes the query on repository and stores result in file passed in store (name value) and return documentList in reference (name value).

process repository “with”, name value location, name value stemmer , name value overwrite, name value filename;

This method computes statistical data on all the document that are contained in repository and store the result for later use. This data is then used by performquery operation at the time of search. Most likely only implementation of tf-idf using cosine similarity would be implemented in term project.