Dynamo: Language Reference Manual

Abhinav Saini Archana Balakrishnan Pradeep Dasigi
Srilekha V K Raghavan Muthuregunathan

1 Introduction

Dynamo is a programming language that allows the programmer to mathematically represent Dynamic Programming (DP) algorithms. The compiler translates it into Java. Problems that can be solved using DP are characterized by overlapping sub problems and optimal substructure which are expressed recursively. However, representation of DP problems in high level languages is cumbersome with significant book keeping involved. Dynamo abstracts the book keeping from the programmer and simplifies the program representation. The program consists of only math-like syntax, is simple and intuitive. The compiler translates the recursive structure specified by the programmer into iterative code in Java.

2 Lexical Conventions

The tokens are identifiers, keywords, and constants. Blank lines, newline characters, spaces, tabs and comments are ignored and are used only to separate tokens.

2.1 Comments

Single line and multi-Line comments are supported. Single line comments are specified by two back slashes "//" at the beginning of the line. Multiple line comments are bounded by /* at the beginning and */ at the end.

2.2 Embedded Java Code

Embedded java code is bound by '$' at either end. The embedded code will be inserted verbatim into the generated Java file without alteration. No embedded Java code may contain the character '$'.

2.3 Identifiers

Any string that is not a keyword or an operator, and contains at least one alphabet or an underscore ('_') is an identifier. It can also contain numerals and hyphens ('-').

2.4 Seperators

The following characters are used as separators:
'{' '}' '(' ')' and ';'

2.5 Keywords

The following are reserved as keywords, and should not be used for any other purpose.

data-init logic if else min max sum product argmax argmin memoize

2.6 Type Inference

The Dynamo language does not have explicitly defined data types. The default type of a variable is assumed to be String. Type is inferred depending on the operators used in the logic section of the code. If conflicting operators are used, then an exception is thrown.

2.7 Constants

All the constants behave similarly to the ones in Java, since they will appear verbatim in the Java code. The constants are of types: boolean, integer, float, char, string.

2.8 Operators and Overloading

2.8.1 Overloaded Operators

The following operators are overloaded:

+	:	*Integer and Float Addition, String Concatenation*
-	:	*Integer and Float Subtraction*
/	:	*Integer and Float Division*
***	:	*Integer and Float Multiplication*
=	:	*Integer, Float, Character, Boolean and String Comparison*

2.8.2 Non-overloaded Operators

The following are non-overloaded operators:

&	:	*Boolean "and"*
||	:	*Boolean "or"*
|..|	:	*Length of String*
(i:j)	:	*Subarray from i to j*

2.9 Structure of a typical program

```
 <name of the program>
{
  data-init <dataName-1>;
    .
    .
  data-init <dataName-n>;
  memoize <table-name>(<dimensions>);
```

```
  logic
  {
    if <conditional1>
    <statement1 to be executed>;
    else if <conditional2>
    <statement2 to be executed>;
            .
            .
    if <conditional-n>
    <statement-n to be executed>;
   }
}
```

The indentation does not have any semantic implications, it may be used for convenience.

2.9.1 Grammar Specification

DynamoProgram → `ProblemName Code`
ProblemName → `\w+`
Code → `‘{’(‘data-init’ Idenifier ‘;’)+ ‘logic’ ‘{’ LogicStatementJavaCode ‘}’ ‘}’`
Identifier → `([‘0’-’9’ \’-’]*[‘A’-’Z’ ‘a’-’z’ \’_’]+[‘0’-’9’ \’-’]*)+`
LogicStatement → `IfStatement (ElseStatement)+`
IfStatement → `‘if’ (expr) expr ‘;’`
ElseStatement → `(‘else’ IfStatement)+ ‘else’ expr ‘;’`
JavaCode → `‘\$’ JavaStatements ‘\$’`

A Sample Code Translation

A.1 Dynamo Code for calculating Levenshtein Distance

```
1    LevDist
2    {
3        data-init str1;
4        data-init str2;
5        logic
6        {
7            if (str1 = "" & str2 = "")
8                0;
9            else if (str2 = "")
10                |str1|;
11            else if (str1 = "")
12                |str2|;
13            else if (str1[i] = str2[j])
14                LevDist(str1(:i-1), str2(:j-1));
15            else
16                min(LevDist(str1(:i-1),str2(:j-1))+1, LevDist(str1(:i),str2(:j-1))+1,
18        }
19    }
```

A.2 Corresponding Java Code

```
1    public class LevDist {
2        public static int LevDist(String str1, String str2) {
3            int temp1R = str1.length()+1;
4            int temp1C = str2.length()+1;
5            int temp1[][] = new int[temp1R][temp1C];
6            for(int i=0;i<temp1R;i++) {
7                for(int j=0;j<temp1C;j++) {
8                    if(i==0&&j==0) {
9                        temp1[i][j]=0;
10                   }
11                   else if(j==0) {
12                       temp1[i][j]=i;
13                   }
14                   else if(i==0) {
15                       temp1[i][j]=j;
16                   }
17                   else if(str1.charAt(i-1)==str2.charAt(j-1)) {
18                       temp1[i][j]=temp1[i-1][j-1];
19                   }
20                   else {
21                       int min = temp1[i-1][j-1]+1;
22                       min = min>(temp1[i][j-1]+1)?temp1[i][j-1]+1:min;
23                       min = min>(temp1[i-1][j]+1)?temp1[i-1][j]+1:min;
24                       temp1[i][j]=min;
25                   }
```

```
26                }
27            }
28            return temp1[temp1R-1][temp1C-1];
29        }
30
31        public static void main(String args[]) {
32            int LevDist_res = LevDist(args[0], args[1]);
33        }
34    }
```

A.3 Notes

Essentially, the mapping is done by converting the logic for different possibilities on strings to filling up a table.

The return types and argument types of the function may be inferred while parsing and used to set the corresponding environment variables which are later used while generating the Java code. If any conflict occurs after the environment variables are set, a type-mismatch exception will be reported.

Lines 5, 6 and 7 in this Java code will be present in almost all the programs (with different data types and dimensions) since any recursive program is mapped to a procedure of filling up a table.

Lines 8 to 25 in this Java code are the ones mapped from logic in the Dynamo code. The length of the sub-strings being dealt with in the Dynamo code are mapped to the indexes of the table being filled. So, condition (str1="" && str2="") in the former is mapped to (i==0 &&j==0) in the latter. The other conditions follow the same logic. Finally the value that has to be returned is LevDist(str1,str2) in Dynamo. So, temp1[temp1R-1][temp1C-1] is returned in Java.

Lines 21 to 24 in Java code will be lines of code that replace the in built function "min" in Dynamo. Corresponding replacements can be defined for max and argmax