Trigonometry Manipulation Language (TML)

COMS W 4115 GROUP PROJECT
2012 FALL

FENG XUECHEN (XF2120)
CHEN QISHU (QC2166)
WAN YU (YW2506)
QU LIANHAO (LQ2140)
ZHANG WANQING (WZ2241)

Motivation

Trigonometry induction problem can be as simple as naïve submission and subtraction, yet sometimes it could drive people crazy with its subtle logic and hidden clues. We believe most science students share the painful process of solving trigonometry induction problems in high math classes. The language is designed to express trigonometry concept and apply it to real world scenarios in a simple and effective manner. The language could be use for educational purpose for both geometric class and entry level programming class. It could be used as a stand-alone language for educational purpose. The language could also be wrapped and called from OCaml if necessary.

Example Scenario

A typical example of the language is to define trigonometry rules, such as the concept of equilateral triangle or regular polygons with three vertexes. There are two definitions of equilateral triangle:

- If all three internal angles of a triangle have the same degree, it is an equilateral triangle.
- If all three sides of a triangle have the same length, it is an equilateral triangle.

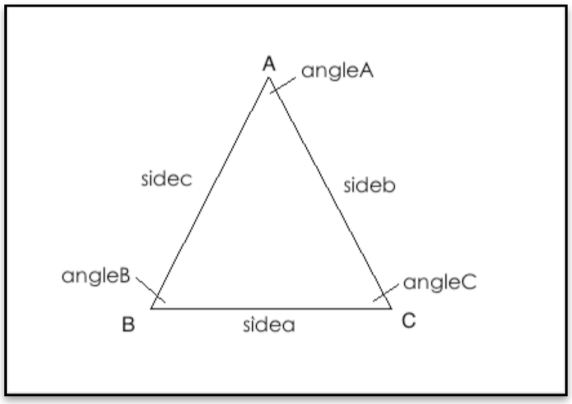
Now we could apply the concept of equilateral triangle to some given triangles to see whether they are equilateral triangles.

Those given triangles are defined in the initialize section. The concept of equilateral triangle is expressed in the rule section. And the concept can be applied to the triangles in the operation section.
Note that the same concept “equilateral triangle” has different explanation, namely same inner angle or same side length. The language is able to accept different explanation on the same concept and choose the viable explanation for the given triangles. Also, User could define the concept equal triangles in the rules section and

apply it later in operation section to prove two triangles are the same in shape. And induct if a triangle is equal shape of a regular triangle, it is also a regular triangle.

Concept and Syntax

The basic naming convention for triangle is as below



A compliable program file consists of three sections each section starts by
- initialize:
- rules:
- operation:

Keywords

The following identifiers are reserved for the use as keywords, and may not be used otherwise:

Triangle	true	initialize	false	rule	print	operation
Else	for	result	angle	area	if	

Initialize

Initialize section create triangle objects. I.E.

```
triangle ABC v((1.0, 1.0) , (2.0, 2.0) , (5.0, 1.0)) ;
```

initialize triangle with vertex location

```
triangle DEF l(3.0 , 2.0 , 3.0) ;
```

initialize triangle with line length

```
triangle GHI v((0.0,0.0) , _ , (0.0,0.0))  l(5.0,  _ , 5.0) ;
```

initialize triangle with vertex location and line length.

Note the initialization parameters for single definition is not necessarily complete. However incomplete definition may potentially leads to an incomplete logic and leads to a compilation error. In this case initialization of triangle GHI would yield a compilation error since no triangle could be made to satisfy the parameter in 2-D plane.

Rule

Rule section is the second section that can be defined by users. The rules are similar to the definition of C functions, which will enable users to set rules' names, arguments and statements. Rule section starts with "Rule:"

The expressions of rules are as follows:

rule -> RuleName (arg){exp}{return_type}
arg -> arg, arg | nil
exp -> exp logicOpt exp | (exp)| variable opt variable |number
logicOpt-> AND|OR|NOT
opt -> + | - | * | / | ==
return_type -> nil | float

Example of rules associate with trigonometry concepts:

R1: specifies a triangle is a regular triangle if all three sides are equal.

```
regular_triangle (triangle_1)
{(triangle_1.sidea == triangle_1.sideb)       AND
  (triangle_1. sideb == triangle_1. sidec)};
```

R2: specifies a triangles is a regular triangle if all three angle are 60 degree

```
regular_triangle (triangle_1)
{(triangle_1.angleA == 60)   AND
  (triangle_1.angleB == 60)   AND
  (triangle_1.angleC == 60)};
```

Note different expressions of regular triangle's concept are express separately yet the language is capable to comprehend it. ALL calculation and logic induction involve elements within a triangle such as inner angle can be defined in Rule section in order to reuse or can be directly be implemented in operation section.

Operation

Operation section applies rules objects created in initialize section and provide system output. I.E.

```
if (right_triangle(ABC)) {
        foreach (Triangles temp){
                similar_triangle (temp, ABC);
                print (temp + "is regular triangle");
        }
}else{
        print (ABC + "is not a regular triangle");
}
```

traverse all initialized triangles on plane and check if a triangle is similar to ABC. If positive then print the result to console

Comment
@ starts comment section till the end of the line. There is no nested comment. "@" inside string count as normal character.
Note that during the scenario: @@, it takes the first @ as the start of the comment, and the second as the comment's content.

Sample Code:

```
@ keyword "initialize:" starts triangle initialization phase
initialize:

@ initialize triangle with 2-D vertex location
triangle ABC = v((1.1, 2.2) , (3.3, 4.4) , (5.5, 6.6));
@initialize triangle with line segment length
triangle DEF = l(4.2 , 3.5 ,3.6);

@ keyword "rules:" starts rules construction phase
rule:
@ explain regular triangle's meaning in terms of line length
identical_triangle (Tri_1, Tri_2) {
        ((Tri_1.A == Tri_2.A ) AND (Tri_1.B == Tri_2.B) AND (Tri_1.C == Tri_2.C)) OR
        ((Tri_1.A == Tri_2.B) AND (Tri_1.B == Tri_2.C) AND (Tri_1.C == Tri_2.A)) OR
        ((Tri_1.A == Tri_2.C) AND (Tri_1.B == Tri_2.A) AND (Tri_1.C == Tri_2.B))};
@ explain right triangle's meaning in terms of angle

regular_triangle (A) {(A.sidea == A.sideb AND A.sideb == A.sidec};
regular_triangle (A) {(A.angleA == 60 AND A.angleB == 60};

@ explain what means of two triangles be equal
equal_triangle (A B) {A.sidea == B.sidea AND A.sideb == B.sibeb AND A.sidec ==
B.sidec};
@ explain angleC in terms of sides
angle_C (A) () {arcos(sqr(A.sidea) + sqr(A.sideb) – sqr(A.sidec) / 2 * A.sidea *
A.sideb)};

@ keyword "operation:" starts operation and calculation phase
operation:
if (identical_triangle (ABC, DEF))
        print (ABC + " and " + DEF + "are identical");
if (regular_triangle(ABC))
        print (ABC + "is regular triangle");
foreach (Triangle TEMP)
        if (TEMP.area > 3.0 AND X.sidea< 0.1) print ("weird shaped triangle");
```