# Triangle Manipulation Language
# TrML
## Language Reference Manual

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

Oct 28 2012

**Table of Content**

# 1 Introduction

Trigonometry induction problem can be as simple as naïve substitution 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. TrML is a programming language that allows user express trigonometry concept, and construct/solve complex trigonometry puzzles.
The language could be use for educational purpose for both geometric class and entry level programming class. This manual provides a comprehensive description of the TrML language.

# 2 Lexical Conventions

TrML has six kinds of tokens: keywords, identifiers, constants, string literals, operators, and separators. Whitespace such as blanks, tabs, and newlines are ignored except when they serve to separate tokens. Comments are also ignored.

## 2.1 Comments

The character @ introduces a comment, which terminate with the new-line character \n. Comments do not nest, and they do not occur within string.

## 2.2 Identifiers

An identifier is a series of alphabetical letters and digits; the first character must be alphabetic.

## 2.3 Keywords

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

| | | | |
|---|---|---|---|
| triangle | true | initialization | false |
| rule | print | operations | while |
| result | string | area | if |
| AND | OR | NOT | sin |
| cos | tan | arcsin | arccos |
| arctan | sqrt | V | L |
| print | _ | angleA | angleB |
| angleC | sideA | sideB | sideC |

## 2.4 Constants

There are five constants in TrML:

### 2.4.1 Value

Value constant consists of an integer part, a decimal point and a fraction part. No exponential part is supported.

### 2.4.2 Triangle

Triangle constant can be either Vertex-constant starts with letter V followed by three tuples of values in square brackets. i.e. V[ (1.1, 2.2), (3.3, 4.4), (1.2, 4.2)] or Line-segment-constant starts with letter L followed by three values in square brackets. i.e. L[1.3, 4.2, 5.4]

### 2.4.3 Boolean

The reserved Boolean constants are *true* and *false*.

### 2.4.4 Sh-cat

The reserved Sh-cat constant is character _ . As Schrödinger's cat means there is a field to be filled but the value in corresponding field is unknown.

### 2.4.5 String

String constant is a sequence of characters surrounded by double quotes, as in "…". A string's value is initialized with the given characters. Only alphabet and number is allowed in string.

# 3 Types

There are two data types in TrML

| | |
|---|---|
| Value | A floating point number |
| Triangle | A Triangle in 2D plane |

# 4 Operators

TrML support most of the standard C Programming Language's arithmetic operations and inherit it's standard operator.

**Operators' Precedence and order of evaluation**

| **Operators** | **Associativity** |
|---|---|
| ( ) [ ] | left to right |
| cos sin tan arcsin arccos arctan sqrt | left to right |
| * / % | left to right |
| + - | left to right |
| >= <= > < | left to right |
| == != | left to right |
| NOT AND OR | left to right |

# 5 Expression and Statement

## 5.1 Expression

Expression consists of arithmetic expression, logic expression and general expression

### 5.1.1 Arithmetic Expression

Operators in arithmetic expressions group left to right

*arithmetic-expression :*

*arithmetic-expression operator arithmetic-expression*

### 5.1.2 Logical Expression

Operators in Logical expressions group left to right

*logical-expression:*

*logical-expression operator logical-expression*

### 5.1.3 General Expression

Operators in Logical expressions group left to right

*general-expression:*

*general-expression operator general-expression*

## 5.2 Statements

A statement consists of while statement, conditional statement, assignment and built-in function calls.

### 5.2.1 Statements and Statement

*statements:*

*{ statement }*

*statement:*

*while-statement*

*conditional-statement*

*assignment-statement*

*built-in-function*

### 5.2.2 Assignment Statement

*assignment-statement:*
*identifier* **=** *expression*

### 5.2.3 Conditional Statement

*conditional-statement:*
***if*** *general-expression* ***then*** *statements*

### 5.2.3 While Statement

*while-statement:*
***while*** *general-expression statements*

### 5.2.4 Built-in-Function

TrML has built-in print function.
*built-in-function:*
*prints ( string-constant )*
*printv ( arithmetic-expression)*

# 6 Building Blocks

The program consists of three building blocks initialization, rules and operations

## 6.1 Initialization

Initialization declares all variables used in the program. A variable may be a value or a triangle. All variable declarations end with semicolon.

value-declaration:
    value-identifier value-constant
*triangle-declaration:*
    *triangle-identifier vertex-constant*
    *triangle-identifier line-segment-constant*
    *triangle-identifier vertex-constant line-segment-constant*

note a triangle with empty information could be created by using sh-cat.

## 6.2 Rules

Trigonometry rules are defined in the rules section. Rule declaration starts with rule name followed by general expression as rules qualification and statements as rule definition.

*rule:*
    *identifier general-expression statements*

Note that the there are two different rules in trigonometry: judgment rule and calculation rule. Both rules can be expressed using the same rule expression:

Judgment Rule:
    *identifier general-expression* ***true***
Calculation Rule:
    *identifier* ***true*** *statements*

## 6.3 Operations

Operations invoke rules from previous block and variables from initialization block. Operations consist of a series of statements. A statement always ends with a semi-colon. No statement can return a value. A statement either calls a built-in function or making an assignment.

*operations:*

*statements*

# 7 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];
value agl 0.0;
@ Keyword “rules:” starts rules construction phase
rule:
@ Explain regular triangle’s meaning in terms of line length.
@ This is a judgment rule
identical_triangle (Tri_1, Tri_2) (
        ((Tri_1.sideA == Tri_2.sideA ) AND (Tri_1. sideB == Tri_2. sideB) AND
(Tri_1. sideC == Tri_2. sideC)) OR
        ((Tri_1. sideA == Tri_2. sideB) AND (Tri_1. sideB == Tri_2. sideC) AND
(Tri_1. sideC == Tri_2. sideA)) OR
        ((Tri_1. sideA == Tri_2. sideC) AND (Tri_1. sideB == Tri_2. sideA) AND
(Tri_1. sideC == Tri_2. sideB))) {true};
@ Explain right triangle’s meaning in terms of angle
regular_triangle (A) (A.sidea == A.sideb AND A.sideb == A.sidec) {true};
regular_triangle (A) (A.angleA == 60 AND A.angleB == 60) {true};

@ 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
@ This is a calculation rule
```

```
angle_C (A) (true) {arcos((A.sideA * A.sideA) + (A.sideB * A.sideB) – (A.sideC
* A.sideC) / 2 * A.sideA * A.sideB)};

@ keyword “operation:” starts operation and calculation phase
operation:
if (identical_triangle (ABC, DEF)) {
        prints (“ABC and  DEF are identical”);
}
if (regular_triangle(ABC))
        print (ABC + “is regular triangle”);
while (agl < 180){
        agl = agl + 5;
        prints (“value agl’s valus is: ”);
        printv ( agl );
}
```

# Appendix:

YACC parser generator grammar pseudo-code for TrML

**@ INITIALIZE**

1. initialization -> **INITIALIZE COLON** initial-declarator

2. initial-declarator-list -> initial-declarator-list initial-declarator

3. initial-declarator-> value-declarator
   | triangle-declarator

4. value-declarator -> **VALUE VID FLOAT SEMICOLON**

5. triangle-declarator -> **TRIANGLE TID** triangle-initializer **SEMICOLON**

6. triangle-initializer-> vertex-status
   | line-status
   | vertex-status line-state
   |

7. line-status -> **L LB** line-initializer **RB**

8. line-initializer-> initial-value **COMMA** initial-value **COMMA** initial-value

9. vertex-status ->**V LB** vertex-initializer **RB**

10. vertex-initializer -> **LP** initial-value **COMMA** initial-value **RP LP** initial-value **COMMA**
    initial-value **RP LP** initial-value **COMMA** initial-value **RP**

11. initial-value->**FLOAT**
    | **sh-toekn**

**@ RULES**

12. rules-def -> **RULE COLON** rule-declarator-list

13. rule-declarator-list -> rule-declarator-list **SEMICOLON** rule-declarator

| rule-declarator

14. rule-declarator -> **ID LP** para-lst **RP LP** expr **RP LP** statement **RP**

15. para-lst -> para-lst **COMMA** parameter
| parameter

16. parameter -> **TID**
|

17. statement->numerical-expr

18. numerical-expr->numerical-expr **MINUS** numerical-expr
| numerical-expr **ADD** numerical-expr
| numerical-expr **MULT** numerical-expr
| numerical-expr **DIVIDE** numerical-expr
| **SIN** numerical-expr
| **COS** numerical-expr
| **TAN** numerical-expr
| **ARCSIN** numerical-expr
| **ARCCOS** numerical-expr
| **ARCTAN** numerical-expr
I **SQRT** numerical-expr
| **FLOAT**
| **VID**
| **TRUE**
| **LP** numerical-expr **RP**
| tri-element

19. tri-element -> para-lst **DOT** element
**| TID DOT** element

20. element -> **A**
**| B**
| **C**
| **SIDEA**
| **SIDEB**
| **SIDEC**
| **ANGLEA**
| **ANGLEB**

| **ANGLEC**
| **AREA**

21. comparison-expr-> comparison **NEQ** comparison
| comparison **EQL** comparison
| comparison **GEQ** comparison
| comparison **SEQ** comparison
| comparison **GRTR** comparison
| comparison **SMLR** comparison
| **LP** comparison **RP**

22. comparison -> numerical-expr

23. expr -> expr **AND** expr
| expr **OR** expr
| **NOT** expr
| **LP** expr **RP**
| comparison-expr

24. operations -> **OPERATION COLON** operation-list

25. operation-list -> operation **SEMICOLON** operation

26. operation -> print
| condition-statement
| while-statement
| assignment

27. print -> **PRINT LP** print-statement **RP**

28. print-statement ->expr
| **STRING**

29. condition-statement -> **IF** expr **THEN** operation-list

30. while-statement -> **WHILE** expr operation-list