Uniform General Algorithmic (UNIGA) Financial Trading Language

Proposal

Leon Wu (llw2107@columbia.edu)
Jiahua Ni (jn2173@columbia.edu)
Jian Pan (jp2472@columbia.edu)
Yang Sha (ys2280@columbia.edu)
Yu Song (ys2310@columbia.edu)

Columbia University in the City of New York

1. Introduction

Today many financial firms have their own trading software tools to facilitate investors' investment process. These tools often differ from each other and usually take a long time to learn. They are also not easy to be customized and very expensive. Many institutional and individual investors have their own custom-designed investment strategies. They want to implement their investment strategies using some easy-to-use software with low cost. However, designing financial trading software from scratch requires professional knowledge and can be costly and time-consuming. The UNIGA Language provides an easy and efficient way for people to design their own investment plan. The language allows users to write a program that can automatically trade financial instruments including Stock, Options, Bonds, and Mutual Funds etc. using pre-defined trading strategy that defines trading rules such as set-price, sell-price, comparisons, quantity and other elements.

2. Overview of this Language

UNIGA is a high-level scripting language. Script programming languages enable programmers to specify trading operations intuitively. Although not as comprehensive as the more well known scripting languages such as Perl or Python, the built-in keywords make the language more intuitive and easy to use. The user is able to design trading software in the form of a program. The translator will then output a Java source file that can be edited and compiled into Java byte-code.

3. Goals

UNIGA Language is a language that enables the users to perform various kinds of trading operations, mainly consisted of selling and buying but with more easy-to-use powerful features, using an interactive coding process. Thus, again, UNIGA Language is meant to be easy to use, portable, powerful, versatile and flexible.

3.1. Ease-to-use

UNIGA Language is a clean, intuitive, and easy-to-learn language which allows users to create their own buying and selling strategies by writing a few lines of code. UNIGA uses a well-defined set of basic syntaxes similar to Perl or PHP and this makes UNIGA programmer-friendly.

3.2. Portability

Since Java is a platform-independent portable language thanks to its own interpreter, UNIGA Language is also portable language because it converts user-created-program into Java code. So, you can execute the program on any platform where Java Virtual Machine is installed.

3.3. Powerful

UNIGA language enables the users to perform various trading transactions, for example buying, selling, and comparing stock prices, with just few lines of code.
This powerfulness allows work to be more and efficient and productive as well as providing a clear outline of what was used to achieve the final result.

3.4. Versatile

Users can target almost every financial market all over the world from New York, Tokyo, London, and Paris to Sydney. Also, users can deal with several financial commodities such as stock, futures, securities, options and so on.

3.5 Flexible

UNIGA Language allows users to add their own functions or even import and other libraries in order to use new functions that the users need.

4. Basic Language Features

4.1 Statement

An UNIGA Language statement represents a complete instruction. Statements can contain reserved words, operators, and punctuation marks, and always end in a semicolon. Examples are shown in Sample Code section.

4.2 Data Types

We have defined our data type as follows:
Numeric: double (to represent date, time, price, trade volume), and its array double[];
True/False: Boolean values.

4.3 Reserved Words

The basic vocabulary of UNIGA Language consists of a set of pre-defined words, which we call reserved words. Reserved words each have a specific meaning or purpose. Mainly, this can be classified into two subsets.

4.3.1 Basic Reserved Words

Close	Last traded price of a bar
Date	Date of the close of a bar
Open	First traded price of a bar
High	Highest traded price of a bar
Low	Lowest traded price of a bar
Market	Current Market Price of present bar
Volume	Number of shares or contracts traded in a bar

4.3.2 Control Reserved Words

While	Used for continuous execution of an action, stops when the preliminary condition disqualifies
For	For loop
If-else	Conditional execution clause.

4.4 Expression and Operators

In UNIGA Language, an expression is any combination of reserved words and operators that represent a value.

4.4.1 Mathematical Operators

Math Operator	Meaning
+	Addition
-	Minus
*	Multiplication
/	Division
()	Parentheses

4.4.2 Relational Operators

Relational Operator	Meaning
<	Less than
>	Greater than
=	Assign
==	Equal to
~	Not Equal

4.4.3 Logical Operators

Relational Operator	Meaning
&	Logical AND
|	Logical OR

4.5 Punctuation Marks

There are a number of punctuation marks to establish statements, define parameters, delimit words, and establish order of precedence.

Symbol	Name	Description
;	Semicolon	Ends a statement.
()	Parentheses	Group values and forces them to be calculated first.
,	Comma	Separates each parameter or input.
:	Colon	Used in declaration statements to begin the list of inputs or variables
" "	Quotation Marks	Defines a text string
[]	Square Brackets	Used to specify elements in an array variable.
{}	Curly Brackets	Used as modifier, to reference a value from a previous bar.

4.6 Built in functions:

In order to separate our language's program from data, we've defined the following built in functions for our language:

Read ( ): read in the data from a data file in predefined format

Print ( ): output the result of the program to standard output

4.7 User functions:

In order to support user defined function, we defined the following means to define and declare a function:

Function foo (para 1, para 2);

5. Sample Code

Sample code to perform operation in UNIGA Language:

5.1 Buy

```
Buy Value1 Shares;
Buy ("MSFT") This Bar on Close 100 Shares;
```

5.2 Sell

```
Sell... next bar on price stop/limit;
Sell ("MSFT") This Bar on market 5 Contracts;
```

5.3 If-else

```
If Close > High 1 bar ago Then Buy on Market;
Else Buy on Close;
```

5.4 While

```
While 25 < Market Begin
PriceDiff[i] = Market- CurrentPrice[i];
i=i+1;
End;
```

6. Optional Features

Because traders and investors tender to make mistakes when typing digits, it would be nice if there are functionalities that automatically detect such misbehaviors and return error messages according to the mistakes users made. This idea is hard to implement as the compiler or interpreter is not easy to determine the range of user input.