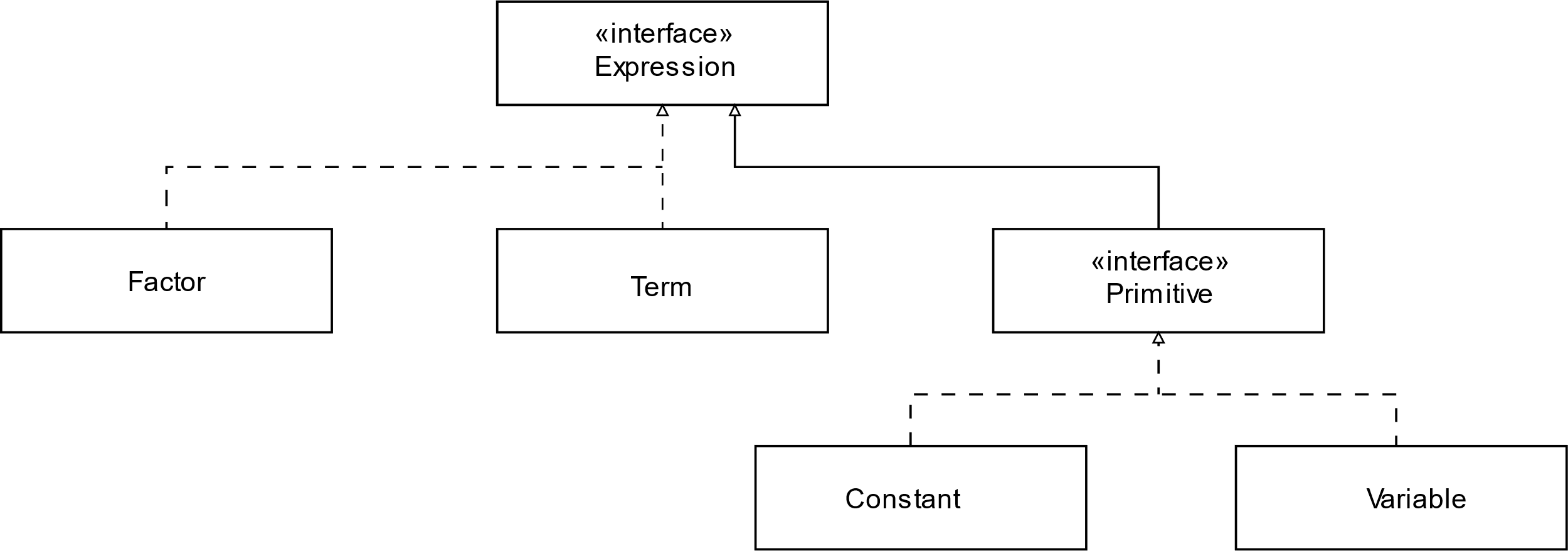
# Algebra of Bool

This project was inspired by a series of You-Tube videos by N J Wildberger. Specifically, MathFoundations 254 through 280. Wildberer based his lectures on the book *The Mathematical Analysis of Logic, being an essay towards a calculus of deductive reasoning* by George Bool which was originally published in 1847 and is currently available from Project Gutenberg. As Wildberger points out the Algebra of Bool as originally described by Bool is not Boolean Algebra. The difference is in Bool’s original the addition and multiplication operations are over a field of size 2 namely {0, 1}. Thus the addition operation corresponds to an exclusive or while Boolean Algebra uses an inclusive or.

## Class Hierarchy



Expression

The Expression interface is at the top of the hierarchy. It defines the following methods:

|  |  |
| --- | --- |
| Method | Description |
| Expression times(Expression e) | Create a new Expression that is the product of this Expression and Expression e. |
| Expression plus(Expression e) | Create a new Expression that is the sum of this Expression and Expression e. |
| default Expression and(Expression e) | Create a new Expression that is the Propositional Logic of this Expression **and** Expression e. |
| default Expression or(Expression e) | Create a new Expression that is the Propositional Logic of this Expression **or** Expression e. |
| default Expression not() | Create a new Expression that is the Propositional Logic negation of **not** this Expression. |
| default Expression impl(Expression e) | Create a new Expression that is the Propositional Logic of this Expression **implies** Expression e. |
| default Expression equiv(Expression e) | Create a new Expression that is the Propositional Logic of this Expression is **equivalent to** Expression e. |

### Primative

This is a tagging interface.

### Constant

This is an enumeration that implements Primative. It defines the constants ZERO and ONE and implements the times and plus methods.

### Variable

A Variable is a String. This class also implements Primative and the times and plus methods.

### Factor

A Factor is the product of one or more Primatives. It implements the times and plus methods.

### Term

A Term is the sum of one or more Factors. It implements the times and plus methods.

## Arithmetic Operations.

All mathematical operations are over the field {0, 1}, essentially arithmetic modulo 2.

### Addition

0 + *x* = *x*

1 + x = 1 if *x*=0  
0 if *x* = 1

*x* + *x* = 0

### Multiplication

0 × *x* = 0

1 × *x* = *x*

*x* × *x* = *x*

## Relation to Propositional Logic

|  |  |
| --- | --- |
| Propositional Logic | Algebra of Bool |
| ¬*x* | 1 + *x* |
| *x* ∧ *y* | *x* × *y* |
| *x* ∨ *y* | *x* + *y* + *x*×*y* |
| *x* → *y* | 1 + *x* + *x*×*y* |
| *x* ≡ *y* | 1 + *x* + *y* |

## Main Programs

Three main programs are included:

* com.pwolfgang.algebraofbool.Main

Program to verify a list of premises against a list of conclusions.

* com.pwolfgang.prolog.Prolog

Program similar to the programming language Prolog, except that normal propositional logic grammar is used instead of the disjunctive normal form.

* Com.pwolfgang.syllogisms.Main

Program to verify the 24 Aristotelian Syllogisms.

### com.pwolfgang.algebraofbool.main

The project includes an ANTLR4 grammar for Propositional Logic. ANTLR generates a parser and tree visitor. The class EvalVisitor translates each Propositional Logic statement into the corresponding Algebra of Bool Expression. The grammar defines a program (PROG) that consists of a sequence of statements (premises) followed by one or more conclusions. The premises are separated from the conclusions by a line (underscore characters). All of the premises are multiplied together. The resulting product is then combined via implication to each of the conclusions. If the result is 1 then the conclusion is validated.

#### Example

Wildberger in this MathFoundations 279 video gives this example.

A logic problem for Sir Galahad:

Knight of the golden road: This road leads to the grail. Also if the stones take you there, so does the marble road.

Knight of the marble road: Neither the gold or stone roads lead to the grail.

Knight of the stone road: Follow the gold and you’ll reach the grail, follow the marble and you’ll be lost.

All knights are liars. Which road to take?

The above statements are expressed in Propositional Logic as follows:

¬(G ∧ (S → M))

¬(¬G ∧ ¬S)

¬(G ∧ ¬M)

Where G represents the gold road, S the stone road, and M the marble road.

Output from the program is as follows:

¬(G∧(S→M))

(1+SMG+G+GS)

¬(¬G∧¬S)

(S+G+GS)

¬(G∧¬M)

(1+GM+G)

(S+SG)

\_\_\_\_\_\_\_\_\_\_

G

G

(1+S+SG)

\_\_\_\_\_\_\_\_\_\_

M

M

(1+S+SG+MS+SGM)

\_\_\_\_\_\_\_\_\_\_

S

S

1

QED

The conclusion is that the stone road leads to the grail.

### com.pwolfgang.prolog.Prolog

A separate ANTLR grammar is included to recognize Propositional Logic statements that include propositions that reference either variables or constants. The program then performs a limited unification process and the evaluates the argument using the Algebra of Bool.

#### Example

The syllogism:

All men are mortal  
Socrates is a man  
Therefore Socrates is mortal.

Is expressed as:

man(socrates).

man(X) → mortal(X).

\_\_\_\_\_\_\_\_\_\_

mortal(socrates).

The unification process converts this to:

man(socrates).

man(X/socrates) → mortal(X/socrates).

\_\_\_\_\_\_\_\_\_\_

mortal(socrates).

This is then verified using the Algebra of Bool.

A similar syllogism:

All men are mortal  
All Greeks are men  
Therefore some Greeks are mortal.

This fails to verify!

An additional premise is needed:

There is at least one Greek.

The current unification algorithm cannot deal with recursive premises.

### com.pwolfgang.syllogisms

This program verifies syllogisms in the Aristotelian form. Each statement is translated using the existential formulation into propositional logic statements and then into Algebra of Bool. The translation of the forms is as follows:

|  |  |  |
| --- | --- | --- |
| All P are Q | P a Q | ¬(P ∧ ¬Q) |
| No P are Q | P e Q | ¬(P ∧ Q) |
| Some P are Q | P i Q | P ∧ Q |
| Some P are not Q | P o Q | P ∧¬Q |

All 24 syllogisms as shown in the Wikipedia article <https://en.wikipedia.org/wiki/Syllogism> are coded in the file CompleteSyllogisms.txt. Existential assumptions are included where needed.