Algebra Homework Application

Nick Hobbs

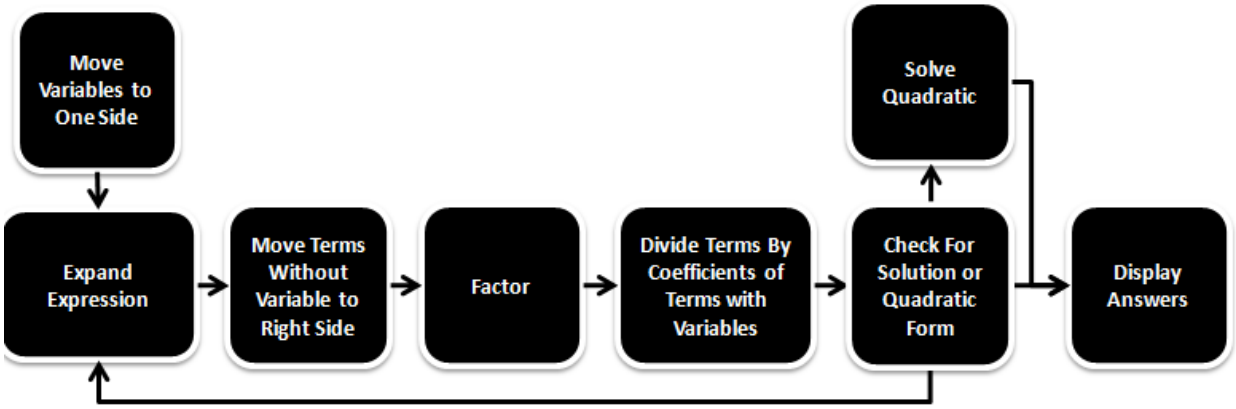
Project Overview:

The goal of this project was to create a dynamic math textbook. The application would provide instant feedback to students working on math homework, much like a python interpreter does for a computer science student. Thus, the goal was to create an application that could solve basic algebraic expressions and quadratics. Then, tell a user when they have made an error.

The solver and the checking method were both implemented. The details of how the solver worked are shown below. The method of checking required the program to first solve the expression, then, substitute in the correct value for the variables for each step of work provided. If the user ever puts in an expression that was not equal for the proper variable value, then an error acceptation is produced.

The Solver:

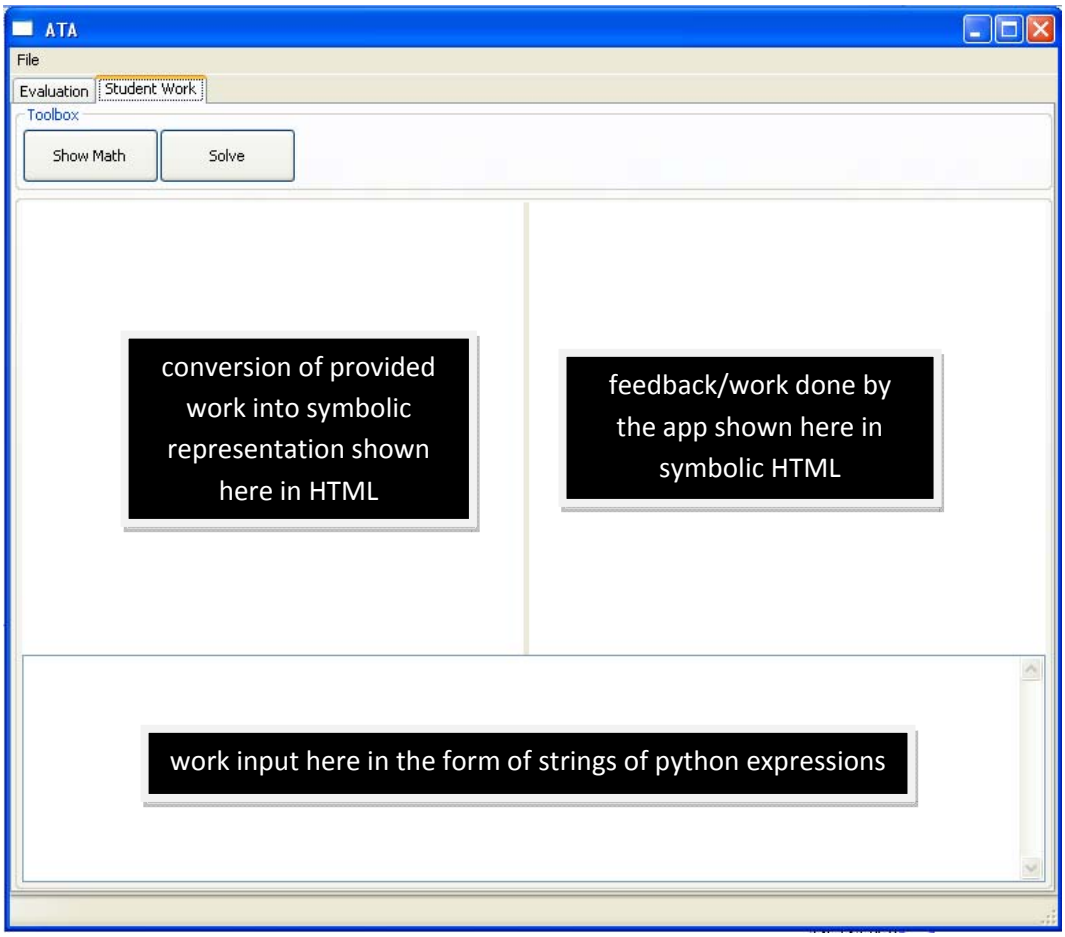
The primary function of the code is an algebra solver capable of solving algebraic expressions. It is capable of solving to the complexity of quadratic equations, and any system of equations that is comprised of such equations. The solver follows a simple algorithm shown in the figure below.



This process uses basic subtraction and division of terms and solves quadratics using the quadratic equation. For a more detailed look at how this code is implemented, view the documentation.

The GUI:

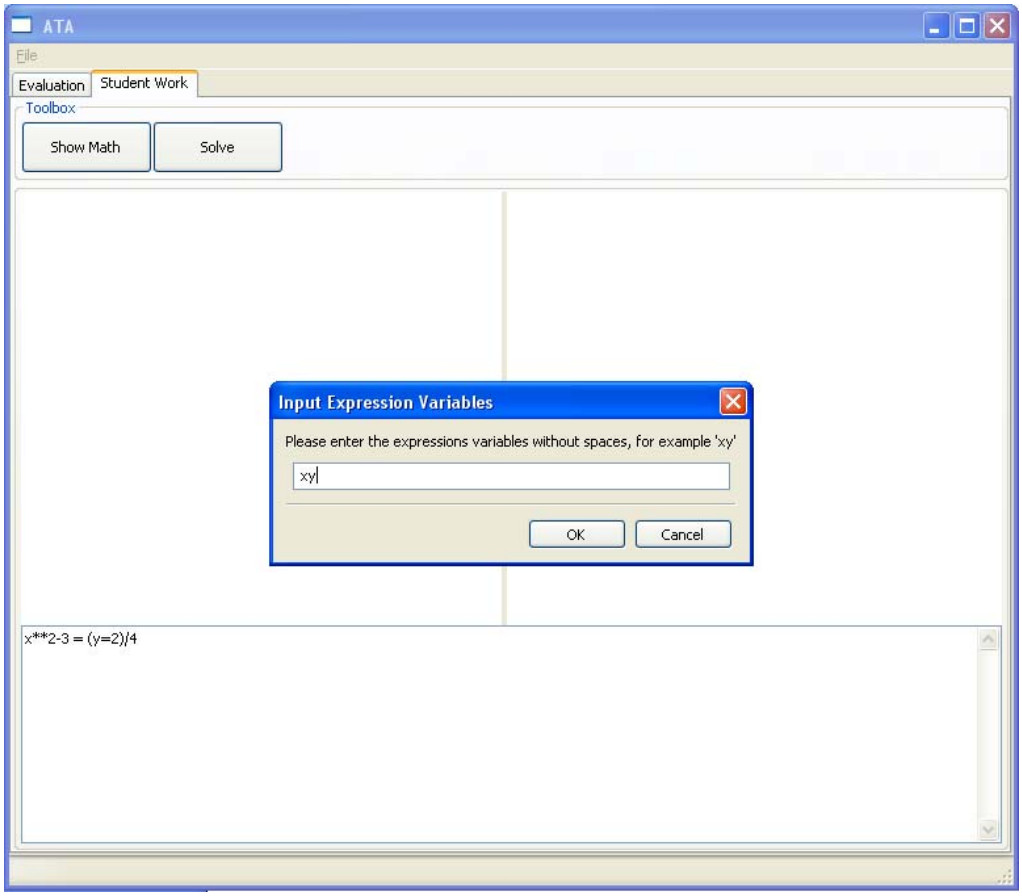
The GUI has three main fields contain in by movable splitter wx widgets. The top two fields display HTML conversions of Latex code. This allows for rich, symbolic expressions much like those found in a text book. The right HTML field displays the converted text provided by the user in the bottom field. The right field shows any computations done by the program with associated work. The idea behind this being, as student could modify their own work will still looking at the feedback provided by the program.



The two buttons in the top left corner provide two different functions. Pressing the “Show Math” button displays the math in the input box into the left HTML field in rich form. Before doing this, it prompts a dialog box (as show below) asking the user to provide the characters of an variables in the expression. This allows the back end of the program to instantiate Expression objects of the represented math. The other option is to solve the equation. Pressing this button prompts two dialog boxes: one to declare the variables, and on to choose which variable to solve for. The result is rich, symbolic HTML displayed in the right html field. This has the step by step representations of how the solver came to its answer.

Finally, the tabs at the top of the page allow different toolbars to be selected. Much like a Microsoft Office ribbon, this allows access to different tools. I was unable to implement functionality to the second

toolbar, however, this would contain resources to be used by teachers, such as writing problems for the student, or viewing old assignments completed by the student.



Documentation for Solver Class:

Expression:

__init__(self,exp,vars)

Instantiates an instance of the expression class. Exp is a string representation of the expression in python syntax(eg: ‘x=(-2*y)/(1+3)’), the variables are declared by the string vars (eg: ‘xy’).

__str__(self):

Returns the string representation of the current state of the expression.

add_terms(self,terms):

Adds the terms provided from each side of the expression. Terms should be a list of expressions. Adds modifications to the work and explanation strings.

check_work(self,work,var):

Returns the line number of the last valid line in the work provided. Work should be a string with each step of work shown on a separate line.

divide_terms(self,terms):

Divides the terms provided from each side of the expression. Terms should be a list of expressions. Adds modifications to the work and explanation strings.

expands(self,var,el=True,er=False):

Expands the sides of the expression (el=expression left, er=expression right) set to true. Adds modifications to the work and explanation strings.

explain_work(self):

Prints a string showing the work done to an expression and the accompanying explanation string.

factors(self,var,el=True,er=False):

Factors the sides of the expression (el=expression left, er=expression right) set to true. Adds modifications to the work and explanation strings.

quadratic_solver(self,var):

Solves a quadratic quation in the form a*x**2+b*x+c = 0 for the variable var. The expression self must already be in this form. If the form is correct and a solution exists, it will add an entry for the given variable in the self.answers dictionary. Otherwise an entry for the variable will be added with the value = None.

simplifies(self,var,el=True,er=False):

Simplifies the sides of the expression (el=expression left, er=expression right) set to true. Adds modifications to the work and explanation strings.

solve_for_var(self,var):

Solves the expression for the specified variable. Solutions are stored in the self.answers attribute. The expression is also converted to the form var=_____. Adds modifications to the work and explanation strings.

subtract_terms(self,term):

Subtracts the terms provided from each side of the expression. Terms should be a list of expressions. Adds modifications to the work and explanation strings.

terms_with_var(self,exp,var):

Returns an expression containing the sum of all of the terms in the expression provided that contain the variable var.

terms_without_var(self,exp,var):

Returns an expression containing the sum of all of the terms in the expression provided that do not contain the variable var.

var_on_one_side(self,var):

Using subtraction, moves all terms containing of the variable var to the left side of the expression. All other terms are moved to the right side. Adds modifications to the work and explanation strings.