

# CSCE625: Introduction to Artificial Intelligence

## Programming Assignment 1: Simplifying Mathematical Expressions via Search

Dylan Shell

Sept 3rd, 2014

### 1 Problem Domain

We are interested in automating the process of simplifying symbolic mathematics. If you've used MATHEMATICA, MATLAB, or MAPLE, chances are that you've seen how symbolic mathematics can be manipulated automatically in order to reduce, solve, or simplify algebraic expressions. This assignment asks you to write a program to manipulate symbolic mathematics.

### 2 Assignment

Write a program in python that takes as input an equation and a symbolic variable. Your program should then use an informed search procedure to simplify and attempt to solve the equation. It should focus on symbolic manipulation of the algebraic expressions, and not focus on the numerical computing. For example, floating point, exact integer (and rational numbers), and symbolic ( $\pi$ ,  $e$ ) values should be kept as separate.

Your assignment requires that you think about the mathematical rules and identities which can be employed as actions in a search procedure (sometimes call operators) in order to attempt to solve for a given variable. Rules to include (these are examples, and not exhaustive; you are expected to start with these and add your own embellishments), the following:

#### Arithmetic evaluation

Arithmetic evaluation which will allow a transformations like:

Example input:

*Eq.* >  $x = (2 + 10) * (2^2)$

*Var.* >  $x$

Produces output:  $x = 48$

Another example input:

*Eq.* >  $x = 6 * 2 / (-1 + 4 * 0 + 1)$

*Var.* >  $x$

Produces output:  $x = \text{undefined}$

#### Applying inverses

Simplifications involving the definition of inverse operations for standard operators should allow for solutions such as:

Example input:

*Eq.* >  $(2 * \text{sqrt}(x) * 3) - y = \pi$

*Var.* >  $x$

Produces output:  $x = ((\pi + y) / 6)^2$

(Or potentially other variations of this final output, depending on how far the arithmetic evaluation proceeds.)

### Associativity and Commutativity

Several of the basic operations (*e.g.*, addition, multiplication, etc.) are associative and commutative. Incorporating these aspects should allow simplifications as follows:

```
Eq.> (2 * x * 3 * y * 4 * z * 5 * 6) = 800
Var.> x
Produces output: x = 80 / (y * z)
```

### Identities

Several logarithmic identities and rules may be incorporated in your treatment of simplification by computing “inverses.” You can further supplement this with a large number of trigonometric identities that, for example, enable the following simplifications:

```
Eq.> e^x = z * (sin(y)^2 + cos(y)^2)
Var.> x
Produces output: x = log(z)
```

A similar, but rather more challenging instance is:

```
Eq.> e^x = sin(8 + 3/2 * z + y - 1/2 * z)^2 + cos(y + 8 + z)^2
Var.> x
Produces output: x = 0
```

### Calculus

Operators for differentials and integrals allow for further simplifications:

```
Eq.> Diff(x^2 + 10x + 2, x) = 4 * z
Var.> x
Produces output: x = 2 * z - 5
```

An extremely challenging instance (via integration by parts) is:

```
Eq.> x = Integrate(z * sin(z), z)
Var.> x
Produces output: x = -z * cos(z) + sin(z) + C
```

## 3 Code and Resources

In order to take care of the tedious input parsing component of this project, I have provided Python code for parsing an input string (in infix form), producing a parse tree, and for outputting this tree (in both infix and prefix forms).

The code is available for download at:

<http://robotics.cs.tamu.edu/dshell/cs625/asgn1/equationparser-0.1.tar.gz>

The code makes use of the open-source PLY (Python Lex-Yacc) Library. It has been tested on GNU/Linux using Python 2.7.3, PLY 3.4. It is intended as example code, providing functionality to parse most of the input forms, with the exception of the (`Diff` and `Integrate`) examples above. You will need to extend the code in order to implement the calculus operators.

Additionally, you may find it useful to make use of the example Python code associated with the book (*e.g.*, for A\* search) available at <https://code.google.com/p/aima-python/>

## 4 Submission

You need to write and submit code to perform symbolic mathematical manipulation and equation solving as completely as you can.

Due date: 17 September at 11:59pm.

Submission method: Via e-mail to the TA.

Submit (in electronic form) the following:

1. Turn in a zip file which includes all the source files and a documentation PDF file, with the items below. (Do not describe your results in the e-mail)
2. The zip file should be named `student-last-name_hw-number.zip`  
*e.g., jones.hw1.zip*
3. Subject of e-mail should be named as `[csce625]last-name_hw-number`  
*e.g., [csce625] jones.hw1*

The zip should include the following:

1. The documentation PDF must include a detailed description of your approach. This should include:
  - (a) A description of the search algorithm.
  - (b) A complete list of the actions you have developed, preferably grouped by type.
  - (c) A specification of the heuristic you employ, and the rationale behind it.
2. Output from your program, for a variety of examples including the instances above, and other examples you cook up to illustrate the effectiveness of your program.
3. The code you wrote for this assignment.
4. A description of how to run the submission.
5. A list of the resources used (*e.g.*, online forums, links to example code on the web, *etc.*).
6. A statement of the Aggie Code of Honor.

You may discuss this openly with your friends and classmates, but you must write your own code and compile your submission independently. If in doubt about whether a resource you used should be included in item 5 above, err on the side of caution and include it.

## References

PLY: PLY (Python Lex-Yacc) an implementation of lex and yacc parsing tools for Python. <http://www.dabeaz.com/ply/>

AIAMA-Python: Python implementation of algorithms from Russell and Norvig's "Artificial Intelligence: A Modern Approach" <https://code.google.com/p/aima-python/>

Norvig: "*Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*," by Peter Norvig, Morgan Kaufmann, 1992.