Middle East Technical University Department of Computer Engineering

CENG 443

Introduction to Object Oriented Programming Languages and Systems

Spring 2019-2020

Homework 1 - A Testing Library for Symbolic Computation Systems

Due date: March 19, 2020, Thursday, 23:55

1 Introduction

In this homework you will practice the object oriented programming concepts you have learned in class. You will implement a testing library for symbolic computation rules that prints its results in HTML, using the Visitor design pattern.

1.1 Symbolic Computation and Rule Matching

A symbolic computation system is any mathematical software with the ability to manipulate mathematical expressions in a way similar to the traditional manual computations of mathematicians and scientists. For example, where an expression $\frac{x}{2} + \frac{x}{2}$ would not be always equal to x in a regular programming environment (due to floating point precision errors etc.), this kind of system would be able to reduce this expression to x. Some of the other typical use cases for symbolic math tools are symbollically calculating complicated derivatives, expanding, factorizing or simplifying polynomials, and even symbollically solving equations.

Most symbolic math tools have a set of **rules** for certain kind operations (such as the symbolic simplification given as an example above). These rules generally include a premise **expression** with **rule variables**, and if the premise is **matched** with another expression, an output(also an expression) is produced related with the matched one.

Assume we have the rule for distribution of multiplication over addition:

$$(V_0 \times (V_1 + V_2)) \vdash ((V_0 \times V_1) + (V_0 \times V_2))$$

With rule variables V_0, V_1, V_2. And then we try to match our expression with it:

- Let's say it is the number 3. The number itself is not a multiplication, so the pattern match fails.
- Let's say it is $(x + 4) \times y$. (Do not confuse the x and y with the **rule variables** above. They are the part of the actual expression, they just represent numerical unknowns in the normal, MATH119 sense!). The expression *is* a multiplication, and V_0 *could* represent $(x + 4)$. However y is not a summation. The match fails. The process is visualized below, "?" is for expressions that are not matched yet, and "!" is for the failed matches.

$$(V_0 \times (V_1 + V_2))[(x + 4) \times y]?$$
$$V_0[(x + 4)]? \times (V_1 + V_2)[y]?$$
$$V_0[(x + 4)] \times (V_1 + V_2)[y]?$$
$$V_0[(x + 4)] \times (V_1 + V_2)[y]!$$

- Let's say it is $(\frac{z}{3+2}) \times (y + (\frac{v+z}{x \times y}))$. It does match:

$$(V_0 \times (V_1 + V_2))[(\frac{z}{3+2}) \times (y + (\frac{v+z}{x \times y}))]?$$
$$V_0[(\frac{z}{3+2})]? \times (V_1 + V_2)[(y + (\frac{v+z}{x \times y}))]?$$
$$V_0[(\frac{z}{3+2})] \times (V_1 + V_2)[(y + (\frac{v+z}{x \times y}))]?$$
$$V_0[(\frac{z}{3+2})] \times (V_1[y]? + V_2[\frac{v+z}{x \times y}]?)$$
$$V_0[(\frac{z}{3+2})] \times (V_1[y] + V_2[\frac{v+z}{x \times y}]?)$$
$$V_0[(\frac{z}{3+2})] \times (V_1[y] + V_2[\frac{v+z}{x \times y}])$$

  Since it matched, the output expression would be:

$$(V_0 \times V_1) + (V_0 \times V_2)$$
$$(V_0[(\frac{z}{3+2})] \times V_1[y]) + (V_0[(\frac{z}{3+2})] \times V_2[\frac{v+z}{x \times y}])$$
$$((\frac{z}{3+2}) \times y) + ((\frac{z}{3+2}) \times (\frac{v+z}{x \times y}))$$

1.2 The Visitor Design Pattern

As you may have guessed, you will have to deal with a *polymorphic tree data structure* of numbers, operations and variables, that all implement the same `MathExpression` interface that will have different behavior in each element. In this assignment, you will implement the following for this tree:

- An operation that prints the tree structure in HTML, viewable in browser.
- An operation that removes all the matches from rule variables in an expression.
- An operation that counts the number of symbols, rule variables, and numbers.

Now, you could declare all these operations in the `MathExpression` interface as `printHTML`,`clearRuleVars`, `countSymbols`, and implement them in the concrete classes. However, if you work like this, adding a new operation would be a hassle (assume homework specification changed and now you have to implement a printLatex operation) even for the library designer, yourself. You will have to add a new method to the interface, and add new implementations to all the concrete classes each time. Soon, the classes will be full of methods, some of which are unrelated to the concrete class (why is there a method called `clearRuleVars` in Number class?!). It will be way worse for the client that uses your library, because he will have to extend or encapsulate all your concrete classes to implement a new operation, since he cannot change definitions!

The visitor design pattern will come to the rescue here. Do not worry, this homework text will hold your hand implementing it, and hopefully show its strong and weak points at the same time. **You are strongly advised to read the link about visitor design pattern, and try to understand it from other sources too if necessary before blindly start coding.** There are some head scratcher questions throughout the text (not graded). Some are:

- *What is double dispatch?*
- *Why do we have to implement the same* `accept` *method in concrete classes? Couldn't we just implement it as a default method in the interface, or make* `MathExpression` *an abstract class that we implement the* `accept` *method?*
- *Is Visitor Pattern beneficial when we need to add new concrete classes to* `MathExpression` *often? Suppose that we need to implement a new kind of rule variable* `NumberVar` *that only matches expressions that consist of* `Num`*s. How are* `MathVisitor`*s affected?*

2 Implementation

2.1 `MathExpression` classes

All the concrete classes here should implement `MathExpression` and call `visit` methods of their visitors in their accept methods with `this` as the argument.

2.1.1 `MathExpression` interface

It should have the following methods:

- `<T> T accept(MathVisitor<T> visitor)`
- `boolean match(MathExpression me)`: returns true if `me` matches `this`. You can use `instanceof` operator here **(AND NOWHERE ELSE IN THE HOMEWORK!)**, unfortunately the pure OO implementation very cumbersome. *Why? Can't we implement it using visitors too? Here is an optional C++ template way that you cannot do in Java because of type erasure.*

2.1.2 `Op` class

A mathematical operator, holds two expressions inside as first and the second term. It should have the following:

- `public Op(String operand,MathExpression first,MathExpression second)`: The operand can be "`+`","`*`","`/`","`|-`", representing addition, multiplication, division, and entails respectively. You can safely assume `first` and `second` as non-`null`.
- `String getOperand(),MathExpression getFirst(),MathExpression getSecond()` as public getters.

2.1.3 `Num` class

Represents a number. It should have the following:

- `public Num(int value)`
- `public int getValue()` as public getter.

2.1.4 `Sym` class

Represents a mathematical variable (not a rule variable!). It should have the following:

- `public Sym(String value)`
- `public String getValue()` as public getter.

Num and Sym are very similar. Although the homework forces you to implement them this way, can you think of a more "generic" solution? How would that affect the `MathVisitor` *interface*?

2.1.5 `Var` class

Represents a rule variable. It contains the expression it previously matched, to check if that matches the next candidate expression. It should have the following:

- `public Var(int id)`: the `id` is just for easy visual identification of the same `Var` instance.
- `public int getId()`: public getter for the `id`.
- `public MathExpression getPreviousMatch()`: get the recent successfully matched expression. If there has been no match yet, returns `null`.
- `public void setPreviousMatch(MathExpression me)`: set the recent successfully matched expression.

- `@Override public boolean match(MathExpression me)`: it there has been no match for `this`, the match succeeds and it should be recorded. If there has been a previous match, returns the match result of `this.getPreviousMatch()` and `me`.

2.2 `DocElement` classes

These are essentially just here to print pretty HTML documents. **All the concrete classes here should implement `DocElement` and call `visit` methods of their visitors in their accept methods with `this` as the argument.**

2.2.1 `DocElement` interface

Only has the accept method for its sibling visitor `DocVisitor`:

- `<T> T accept(TextVisitor<T> visitor)`

2.2.2 `Paragraph` class

Holds text. It should have the following:

- `public Paragraph(String text)`
- `public String getText()` as public getter.

2.2.3 `EquationText` class

Holds a `MathExpression`. It should have the following:

- `public EquationText(MathExpression innerMath)`
- `public MathExpression getInnerMath()` as public getter.

2.2.4 `Document` class

A collection of `DocElements`, with a title. It should have the following:

- `public Document(String title)`: constructs a document with no elements, with `title` as title.
- `public ArrayList<DocElement> getElements()`: a getter of all the elements, in the order they were added.
- `public void setElements(ArrayList<DocElement> arr)`: the collection is reset to the elements inside `arr`, as if they were added in the order as they are in `arr`.
- `public void add(DocElement de)`: adds `de` to the collection.
- `void setTitle(String title),String getTitle()` as public getter and setter for the title.

2.3 `MathVisitor` classes

2.3.1 `MathVisitor<T>` interface

It should have the following:

- `T visit(Op op)`
- `T visit(Num num)`
- `T visit(Sym sym)`
- `T visit(Var var)`

2.3.2 `CountAtomicsVisitor implements MathVisitor<Integer>`

Let `visitor` be of this type. Let `me` be a `MathExpression`. Calling `me.accept(visitor)` should return the how many atomics there are (`Sym,Var,Num`) in `me`.

2.3.3 `ClearVarsVisitor implements MathVisitor<Void>`

Let `visitor` be of this type. Let `me` be a `MathExpression`.
Calling `me.accept(visitor)` should call `setPreviousMatch(null)` in all rule variables.

2.3.4 `PrintMathMLVisitor implements MathVisitor<String>`

Let `visitor` be of this type. Let `me` be a `MathExpression`. Calling `me.accept(visitor)` should return the MathML string of the expression. Refer to Table 1 for treating the types:

`MathExpression`	MathML	Comments
`Num`	`<mrow><mn> Value </mn></mrow>`	
`Sym`	`<mrow><mi> Value </mi></mrow>`	
`Var` with no previous match	`<mrow><msub><mi>V</mi><mn> Id </mn></msub></mrow>`	prints "V_{Id}"
`Var` with previous match	`<mrow><msub><mi>V</mi><mn> Id </mn></msub><mo>[</mo> Prev <mo>]</mo></mrow>`	prints "$V_{Id}[Prev]$"
`Op` (not Division)	`<mrow><mo>(</mo> First <mo> Op </mo> Second <mo>)</mo></mrow>`	prints "$(First\ Op\ Second)$." Use `"+","&times;","&vdash;"` for addition, multiplication and entails respectively.
`Op` (Division)	`<mrow><mfrac> First Second </mfrac></mrow>`	prints $\frac{First}{Second}$

Table 1: MathML representations.

2.4 TextVisitor classes

2.4.1 `TextVisitor<T>` interface

It should have the following:

- `T visit(Document document)`
- `T visit(EquationText equationText)`
- `T visit(Paragraph paragraph)`

2.4.2 `PrintHTMLVisitor implements TextVisitor<String>`

Let `visitor` be of this type. Let `de` be a `DocElement`. Calling `de.accept(visitor)` should return the HTML string of the element. Refer to Table 2 for treating the types:

2.5 Rule Classes

All the classes here should implement the `Rule` interface.

DocElement	HTML	Comments
Paragraph	Text	
EquationText	$InnerMath$	InnerMath should be the MathML string of the MathExpression.
Document	<html><head><title> Title </title></head><body> Elements </body></html>	Elements is the HTML representations of all the elements in the collection, in the order they were added.

Table 2: HTML representations.

2.5.1 Rule interface

It should have the following:

- `void clear()`: clears all rule variables in the premise and entailing expressions. *You can implement this as a default method. How?*
- `boolean apply(MathExpression me)`: clears all rule variables in the premise and entailing expressions, and then attempts matching `me` to the premise. If the match fails, the premise and the entailing expression should stay clear. If the match succeeds, the rule variables in the premise and the entailing expression should be updated to show what was matched. Returns the result of the matching process. **You can safely assume that `me` will not contain any `Vars`**, although if your implementation is good, then your method will not have a problem with that as well. *You can implement this as a default method. How?*
- `MathExpression getPremise()`: returns the premise expression.
- `MathExpression getEntails()`: returns the entailing expression.
- `MathExpression entails(MathExpression me)`: calls `apply(me)` and then returns `getEntails()`. *You can implement this as a default method - obviously.*

2.5.2 XDotYDivXIsYRule class

Implements the rule $\frac{(V_x \times V_y)}{V_x} \vdash V_y$. It should have the following:

- `XDotYDivXIsYRule(Var x, Var y)`: constructs the premise and entailing expressions with the rule variables given as argument.
- `Var getX(),Var getY()` as public getters.

Some matching examples:

- $(x \times (y+3)) \times (\frac{z}{(x \times (y+3))})$: The topmost operator is multiplication. The match fails.
- $\frac{(x \times (y+3))+z}{(x \times (y+3))}$: The topmost operator is division. But the first element is not multiplication. The match fails.
- $\frac{(x \times (y+3)) \times z}{(x \times (y+2))}$: The topmost operator is division. For the first expression, it is a multiplication. For the first element of multiplication, V_x could match $[(x \times (y+3)]$, and for the second element V_y could match $[z]$. Then the first element of the division is OK. The second element of the division needs to match V_x, but it has previously matched $[(x \times (y+3)]$. Unfortunately $[(x \times (y+2))]$ doesn't match that. The match fails.
- $\frac{(x \times (y+3)) \times z}{(x \times (y+3))}$: The topmost operator is division. For the first expression, it is a multiplication. For the first element of multiplication, V_x could match $[(x \times (y+3)]$, and for the second element V_y could match $[z]$. Then the first element of the division is OK. The second element of the division needs to match V_x, but it has previously matched $[(x \times (y+3)]$. Luckily $[(x \times (y+3))]$ matches that. The match succeeds.

2.5.3 `XDotZeroIsZeroRule` class

Implements the rule $(V_x \times 0) \vdash 0$. It should have the following:

- `XDotZeroIsZeroRule(Var x)`: constructs the premise and entailing expressions with the rule variables given as argument.
- `Var getX()` as public getter.

Some matching examples:

- 0: It is not a multiplication, the match fails.
- $1 \times \frac{0}{2}$: The topmost operation is multiplication, V_x could match 1, but the second element is not 0, the match fails.
- 0×0: The topmost operation is multiplication. V_x could match 0, and the second element is zero. It matches.

2.5.4 `XPlusXIs2XRule` class

Implements the rule $(V_x + V_x) \vdash (2 \times V_x)$ It should have the following:

- `XPlusXIs2XRule(Var x)`: constructs the premise and entailing expressions with the rule variable given as argument.
- `Var getX()` as public getter.

3 UML Class Diagram

You are required to provide a UML Class Diagram of all the classes (even the ones provided) and the relations between them. If it makes it hard to see what is going on, you may omit aggregation links, but be sure to verbally show those by indicating the type of the attribute in the UML diagram of that class. Your UML Class Diagram should be a pdf file.

4 Other Specifications

- Your library will be blackbox tested against a custom `main` function where various `MathExpressions` and `DocElements` are instantiated, against which the integrity of your visitors and rules are checked. You can find a sample in ODTUCLASS.
- Your OO design will also affect your grade. You are allowed to create additional concrete classes and methods to these, but you should be able to implement everything without any addition to the interfaces described in this text, in an almost pure OO way (except the `match` method). If you need to create another interface, ask yourself if it is really required before shooting yourself in the foot. **Points may be deducted for especially adding new methods to the `DocElement/MathExpression` interfaces, or cheating out by creating a new interface and adding it to each concrete class**. Use custom visitors instead, that is one of the main points of this homework! If you have any doubts, you can use ODTUCLASS or mail me regarding your design choice.
- **Again, you are not allowed to use `instanceof` operator anywhere except the `match` method. Stay OO.**
- **This is an individual assignment. Using any piece of code that is not your own is strictly forbidden and constitutes as cheating. This includes friends, previous homeworks, or the Internet. The violators will be punished according to the department regulations.**

- **Late Submission: As indicated in the syllabus, $5day^2$ for at most 3 days.**
- Follow the course page on ODTUClass for any updates and clarifications. Please ask your questions on ODTUClass instead of e-mailing if the question does not contain code or solution.

5 Submission

Put all of your classes and interface definitions under package "`hw1`". Submission will be done via ODTUClass. You will submit a single tar file called "`hw1.tar.gz`". This should contain "`uml.pdf`", the uml diagram, and the `hw1` directory, containing all your source files. Your homework should compile with

```
> tar -xf hw1.tar.gz
> cd hw1
> javac *.java
```