IS593 Project Proposal:

Topic: Floating point Imprecision Identification

- Check if the imprecision of a less precise floating point representation is above a certain threshold.

Background and Motivation:

Floating point numbers are widely used in real world applications. After all, they are one of the most natural units of measurement. Although integers will help us iterate through items, and count the unrealistic discrete objects, they fail when we want to represent values for continuous measurement. In theory, floating point numbers should have no problems doing what Integers do and more, but in the computing world, they are usually the programmer's last resort. The most important reason for this is the unpredictability of rigorous computations on them. And this in turn is the fault of today's defacto representation, namely, the IEEE 754 floating point standard.

Floating point numbers are notoriously known for propagating imprecision which can easily happen a series of multiplications, divisions and most notoriously exponentiations. This has caused many real world tragedies in the past and has surely been a source of headaches to countless number of programmers. Current most common IEEE 754 floating point number formats are Single Precision (32 bits), Double Precision (64 bits) and Extended Precision (80 bits). These are also intuitively in order of increasing precision and representable range.

Many applications will have to identify which representation to use based on their own preferences as there are tradeoffs to the options. 32 bit floats would save memory but are less precise. The most common 'double' float type will have more precision, but comes at a larger cost for space. The speed tradeoff between the two depends on the host machines hardware configuration. The third option, Extended precision, 'long double' in c, will cost more space and speed but comes at a greater precision. Which option to choose for an application is not always straightforward because of the reasons explained above. Hence, finding an automated way to locate imprecision propagation would help get rid of this problem.

Approach:

For this project, I will focus on comparing 32 and 64 bit floats. I will use similar abstract values as we learned in the homework (more specifically the interval domain). Users will input the threshold a precision offset that they will tolerate in their program, say 5*10(-5). The analyzer would then be able to note out the code locations that the difference between 32-bit computation value and the 64-bit computation value of the floating point operation would exceed the threshold 5*10^(-5).

The analyzer will keep track of two abstract values for every float-32 definition, one in float-32 and another in float-64 representation of that number. Up on any floating point operation, the analyzer will compute the abstract values on both of those representation domains. After following a flow-sensitive computation, the analyzer would then detect from the final abstract memory entries which positions will make the assertion fail.

Challenges:

- Ocaml does not support 32-bit floats. For this problem, I will either use external C functions for all the 32-bit operations or find other libraries that can provide the functionality.
- The analyzer would only be able to work on a predefined threshold. To make the tool completely autonomous, I need to do some in depth analysis to come up with safety relationship of floating point imprecision. (eg, for f > 10 * 10^(10) threshold should be 0.001, …)
- Growing the tool to support multiple floating point representations and finding the best one for users needs, taking into account the range of possible values and desired threshold of precision.

Expected Result and Evaluation:

The tool should be able to run on real world applications since it is only concerned with a very specific program elements – floats. Many real world C programs contain 32-bit floating point numbers and testing them will not be difficult. In addition, finding imprecise locations for a program that uses 32-bit floats is almost guaranteed since we can always lower the threshold to find even the smallest discrepancy. This way we can even compare safety status of two programs that have the same functionality by comparing each of their highest discrepancies. The correctness of the tool, for the time being, will be tested through inspection until I find another automated way of testing.