# FPEAnalysis System: Developer's Manual

An Independent Reading and Research Project (Fall 2015)
under Professor David Bindel (bindel@cs.cornell.edu)
Cornell University

Humam A Alwassel (haa54@cornell.edu)

Revision 1, updated December 21, 2015

This document is the developer's manual for FPEAnanlysis system. It explains the main ideas and techniques that the system uses. This document is technical in nature and is aimed towards developers with a sufficient computer science background, specifically in numerical analysis. Please read the "User's Manual" first before reading this document.

## 1 Source Code Files

The system source code files are:

- **analysis.ml**: a library for computing error estimates to analyzing stability of expressions
- **ast.ml**: type definitions for the abstract syntax tree and the data structures used by FPEAnalysis
- **augment.ml**: a library for augmenting expression with (1+ $\delta$)'s
- **deriv.ml**: a library for computing partial derivatives of expressions
- **eval.ml**: a library for evaluating expressions using interval arithmetics
- **FPEAnalysis.ml**: the main system program
- **lexer.mll**: the lexer for the input files
- **parser.mly**: the parser for the input files
- **pprint.ml**: a pretty print library for data types used by the system
- **util.ml**: a utility library with functions used across the system, such as functions for simplifying expressions and interval

## 2 The (1+$\delta$) Model

Suppose that *expr* is an arbitrary arithmetic expression with an exact value $v$. Let fl(*expr*) be the correctly rounded floating point representation of *expr*'s value. The (1+$\delta$) model states that fl(*expr*) = $(1 + \delta) * v$, where $|\delta| < \epsilon_{mach}$. This means that the absolute and relative floating point errors are $\delta * v$ and $1 + \delta$, respectively. This is a linear model for estimating the floating point error.

# 3 Interval Arithmetics

Read about the subject on https://en.wikipedia.org/wiki/Interval_arithmetic

# 4 How FPEAnalysis Works?

FPEAnalysis takes the following main steps to produce its final analysis report:

1. Parse the input file and divide it into a $(bindings, conditions, body)$ tuple, where
   - *bindings* is the list of $(var, val)$ pairs from the first curly braces list
   - *conditions* is the list of $(subexpr, val)$ pairs from the second curly braces list
   - *body* is the program body converted to an *Ast.aexp* type
2. Convert the lists *bindings* and *conditions* to an $Ast.BindingMap$ and an $Ast.ConditionMap$ maps, respectively (call these maps *bind_map* and *cond_map*, respectively)
3. Augment *body* and each of its subexpressions with $(1 + \delta)$'s. Augment each subexpression with a different $\delta$ variable (use $\delta_0, \delta_1, \delta_2, ...$) with the following exceptions:
   - Augment expressions that are equivalent with the same $\delta$ because equivalent expressions should have the same error. For example, the two expressions $(a + b)$ and $(b + a)$ are equivalent and should have the same $\delta$ (see the *Ast.expCompare* compare function for the equivalence rules)
   - Do not augment a subexpression $(a - b)$ if $a$ and $b$ are within a factor of 2 from each other (use the *bind_map* to decide this). There is no floating point error in such cases and the computation is exact

   Store each subexpression and it augmented version in an $Ast.Input2AugmentedMap$ map (call it $in2aug\_map$)
4. Take the partial derivatives of each augmented subexpression with respect to each $\delta$. Linearize each partial derivative about $\delta = 0$. Store the results in an $Ast.Augmented2DerivMap$ map (call it $aug2deriv\_map$)
5. Compute the worst absolute and relative errors for each subexpression. Let $eval(expr)$ donates the interval value of *expr* under the variable bindings in *bind_map* and after applying any applicable conditions from *cond_map*. The worst absolute error is

$$w\_abs\_err = \max\left(\sum_i eval\left(\left[\frac{\partial\ expr}{\partial\ \delta_i}\right]_{linearized}\right)\right) * \epsilon_{mach}.$$

   The worst relative error is

$$w\_rel\_err = \frac{w\_abs\_err}{\min eval(expr)}.$$

6. Process the computed worst relative errors for each subexpression to find critical subexpressions. Consider an expression critical if it has a high relative error but its children subexpressions have low relative error. Compile the analysis report and return it to the user