FlowFPX: Discovering and Tracking Exceptional Values

Ashton Wiersdorf, Taylor Allred, Xinyi Li, Ben Greenman, Ganesh Gopalakrishnan

Example: Killed NaNs

Two functions that compute the maximum in a list. The first will silently kill NaNs, while the second correctly propagates.

NaN Killer

```
function max_kill(lst)
  curr_max = 0.0
  for x in lst
    if curr_max < x
      curr_max = x
    end
  end
  curr_max
end
```

```
julia> max_kill([1, NaN, 4])
4.0
```

NaN Propagator

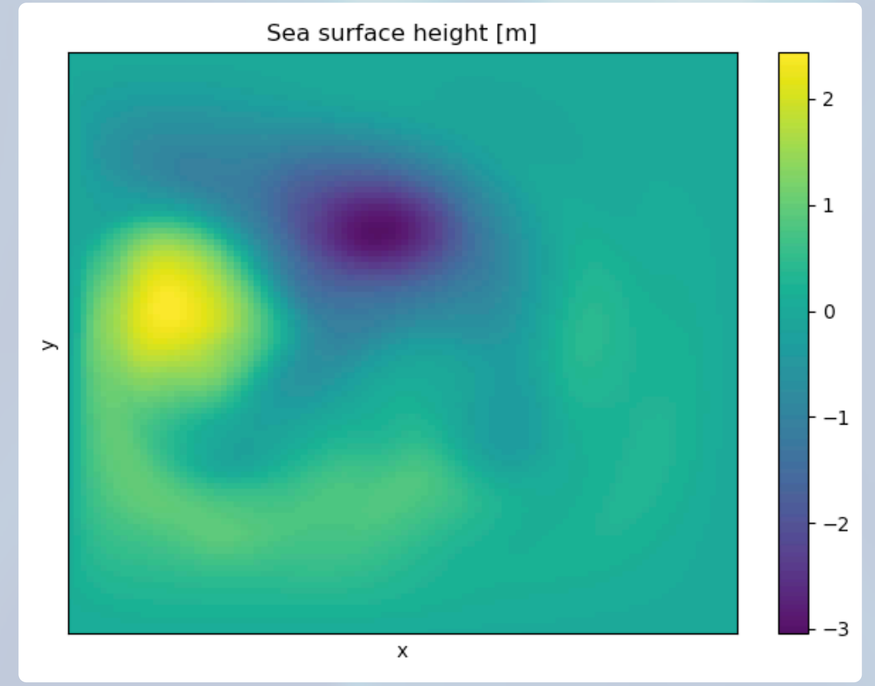
```
function max_prop(lst)
  foldl(max, lst)
end

# The `max' built-in
# correctly propagates
# NaNs, unlike <
```

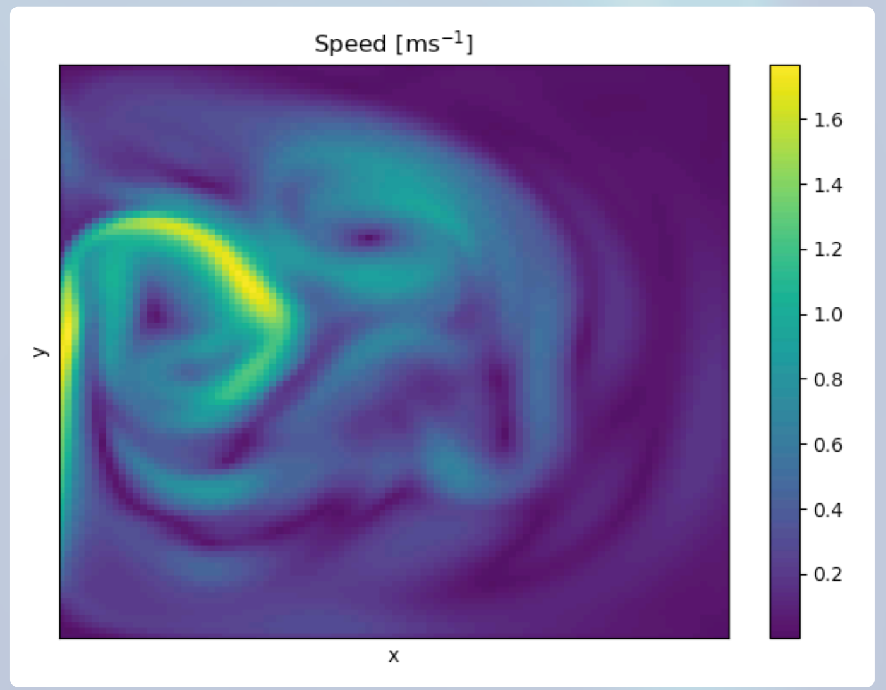
```
julia> max_prop([1, NaN, 4])
NaN
```

Example: Unexplained NaNs

NaNs can sometimes appear in the output of our programs and we don't know where they came from. FlowFPX provides tools to discover the source of NaNs.



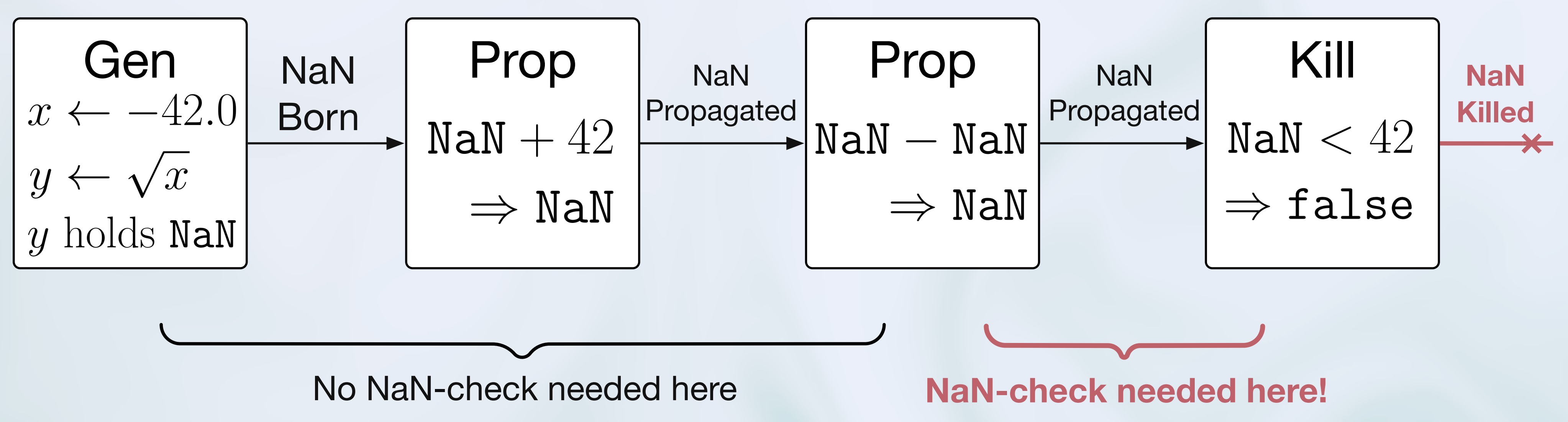
ShallowWaters.jl is a fluid physics simulator. It takes a map of a sea floor elevation, like the graph shown at left. ShallowWaters simulates the flow of fluid and produces the graph below.



This simulation is resource-intensive. Several parameters let us adjust the precision to get faster render times. However, some of these parameters (e.g. CFL) at certain values cause the computation to crash or produce bad results because of spurious NaNs. Debugging is slow and error-prone without better tools to detect where the NaNs came from.

Image credits: ShallowWaters.jl library https://milankl.github.io/ShallowWaters.jl/dev/ Background image credits: ShallowWaters.jl repository https://github.com/milankl/shallowwaters.jl

The Life-Cycle of a NaN



Methods

Collecting NaN Life-Cycles

FloatTracker is a Julia library that provides a wrapper type floating-point values which emits call stacks at interesting points, such as when a NaN flows into an operation but does not flow out. FloatTracker makes use of Julia's type-based dispatch mechanism and metaprogramming capabilities to make the TrackedFloat type a drop-in replacement for any Float type.

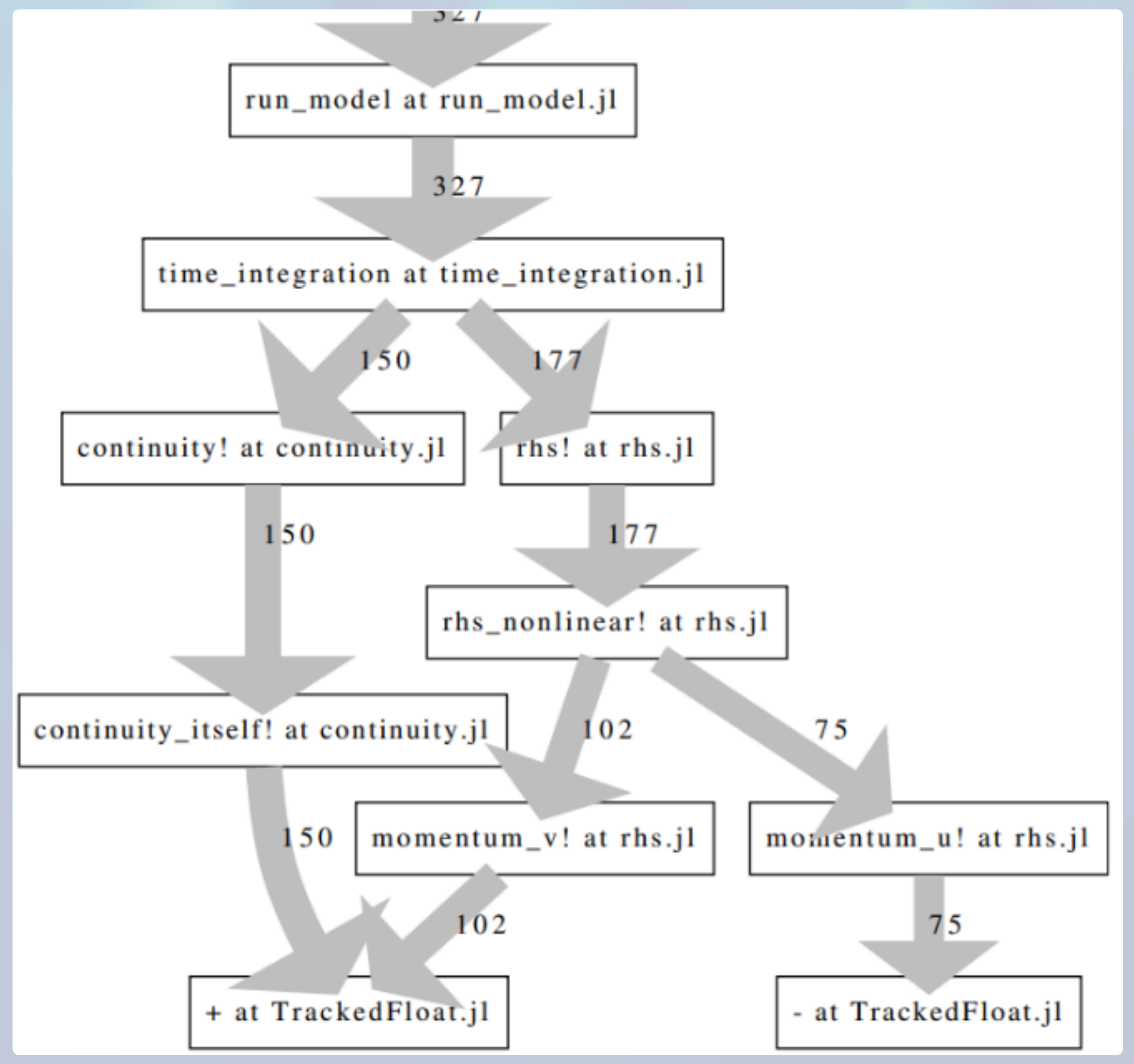
```
@eval function Base.max(x::TrackedFloat64, y::TrackedFloat64)
  # Call to `max' on a TrackedFloat type captured here...
end
```

Julia's type-based dispatch in action

FloatTracker can also inject NaNs to examine how resilient programs are against improper NaN handling. This way, we can test programs for which we have no examples of input data that produce NaNs, but which may encounter NaNs in production.

Making Sense of the Stack Traces

CSTG is a stack trace visualization tool that we can use to understand how NaNs flow through the computation. (Pictured at right) CSTG also allows us to diff two stack trace graphs to find locations where bugs crop up.



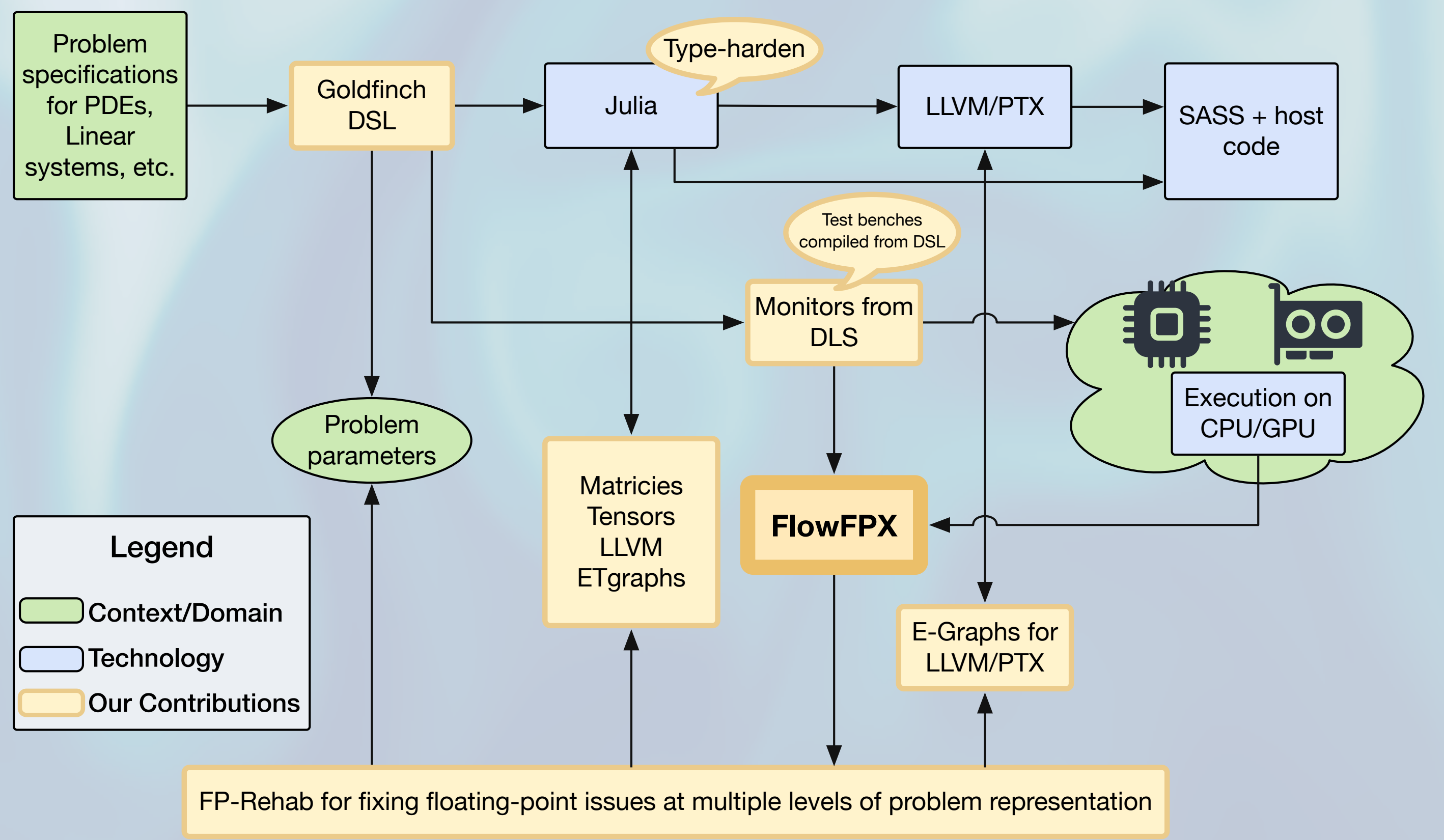
Background

Reliable numerical computations are central to HPC and ML. These computations often use IEEE floating-point to represent real numbers efficiently. This representation has some quirks that can destroy the correctness or the usefulness of a computation. FlowFPX is a tool to track the lifetime of NaNs (values representing "not-a-number") in computation.

NaNs arise from nonsensical computations (such as taking the square root of a negative number) or unrepresentable computations (due to floating-point quirks). NaNs propagate: almost all operations with a NaN produce a NaN regardless of what the other arguments were.

However, there are times when NaNs do not propagate. We call these "kills", as the NaN disappears, but not before changing the computation in potentially fatal ways! For example, NaN < 42 returns false, and 1.0^{NaN} produces 1.0.

Further Work


Problem specifications for PDEs, Linear systems, etc.
Goldfinch DSL
Julia
Type-harden
LLVM/PTX
SASS + host code
Test benches compiled from DSL
Monitors from DLS
Execution on CPU/GPU
Problem parameters
Matricies Tensors LLVM ETgraphs
FlowFPX
E-Graphs for LLVM/PTX
Legend
Context/Domain
Technology
Our Contributions
FP-Rehab for fixing floating-point issues at multiple levels of problem representation

FloatTracker incurs significant run-time costs due to its interception of every function call operating on TrackedFloat values. We would eventually like to move to a static, ahead-of-time solution that incurs no run-time overhead. We will investigate novel type-systems to track areas where floating-point exceptions can occur and inject a minimal set of checks to ensure safe and sound floating-point calculations.

FlowFPX is just one component of a larger system designed to make floating-point computation more robust, easier to debug, and more reliable. Our contributions are highlighted in yellow.