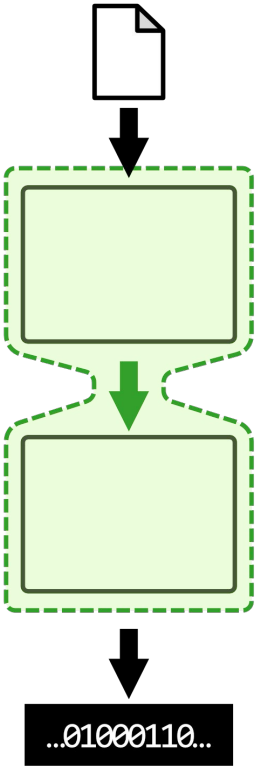
GHENT UNIVERSITY · UNIVERSITY OF CAMBRIDGE · THE UNIVERSITY OF EDINBURGH · UCL

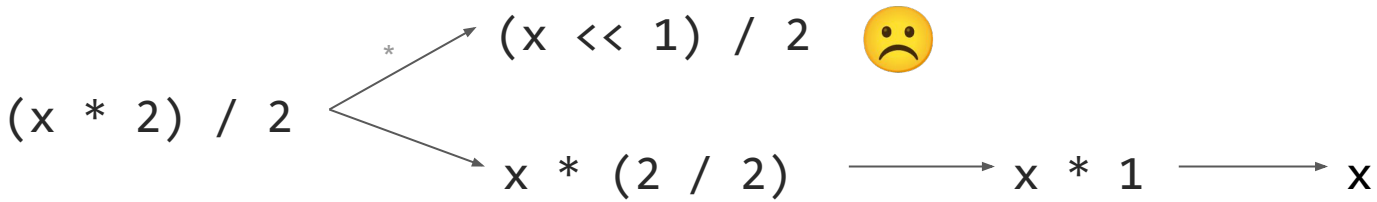
Tamagoyaki

E-Graphs as a Persistent Compiler Abstraction



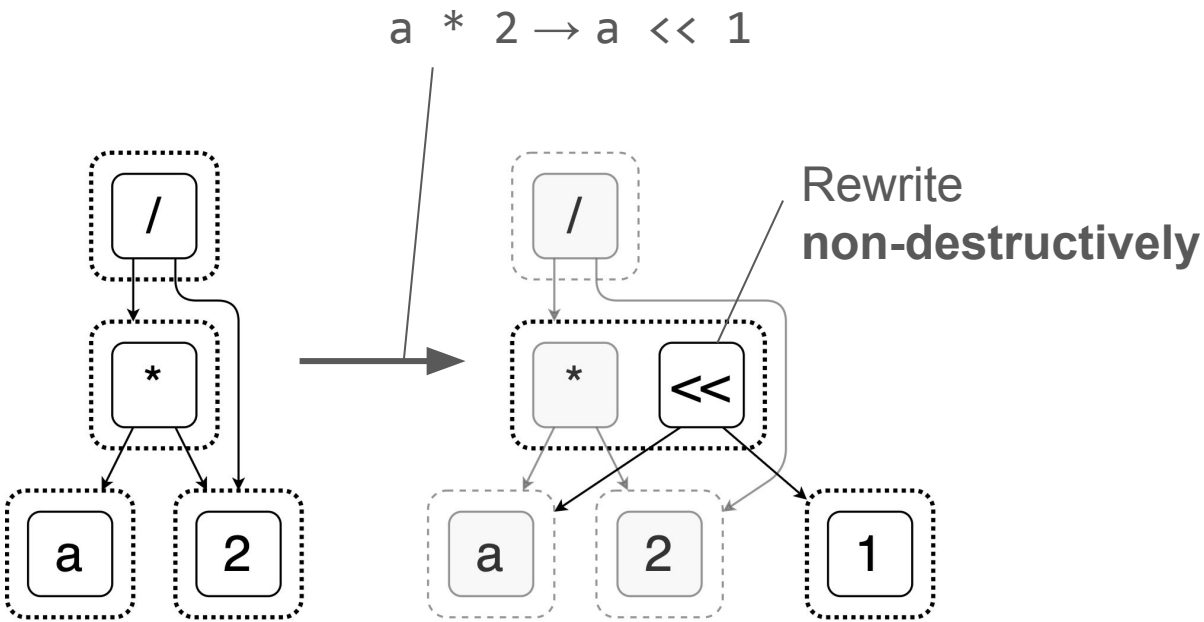
Jules Merckx · Alexandre Lopoukhine · Jianyi Cheng · Sam Coward · Bjorn De Sutter · Tobias Grosser

Which rewrite do you apply first?

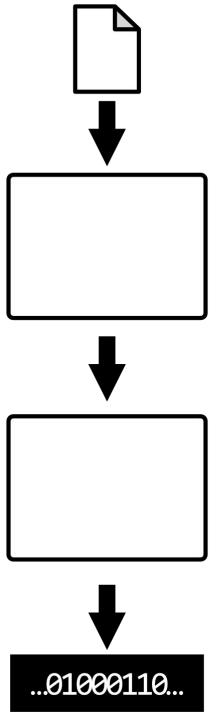


* Assume x is an unsigned integer with UB overflow

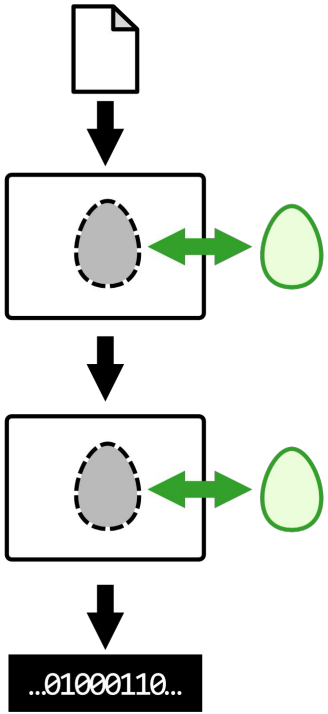
E-graphs store equivalent expressions



Code is transformed at different levels of abstraction



Egg is extensible but not integrated



Egg

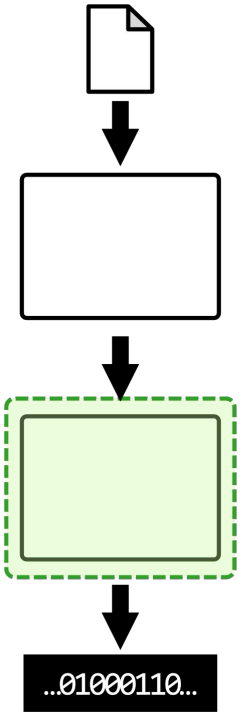
🙂 At different levels of abstraction

☹️ Equalities are lost

Cranelift offers tight compiler integration

☹ Equalities are lost

Egg

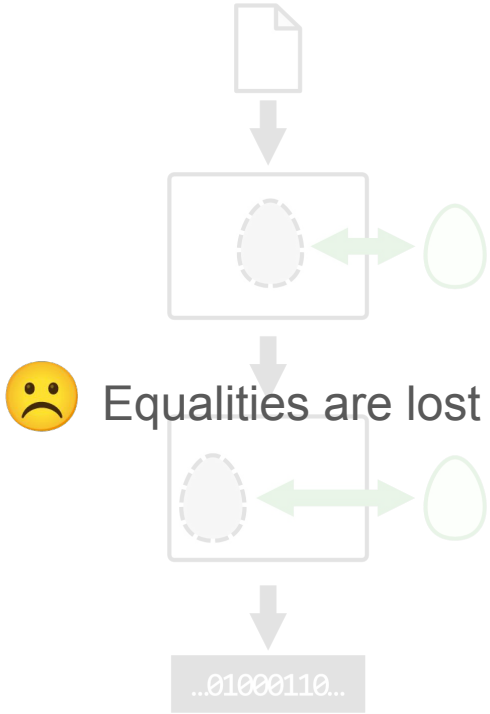


☹ Not at other levels of abstraction

🙂 Equality saturation is integrated

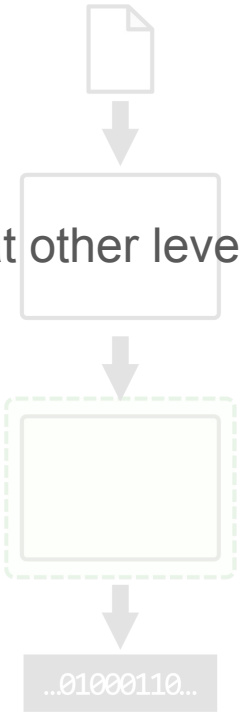
Cranelift

We aim to combine extensibility and integration



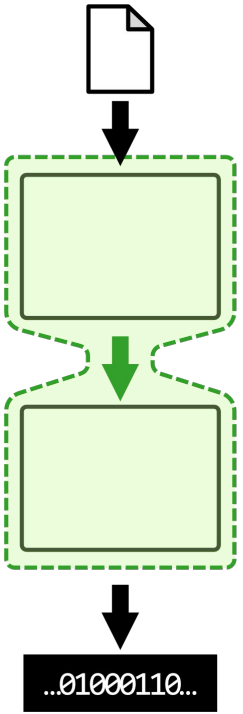
☹ Equalities are lost

Egg



☹ Not at other levels of abstraction

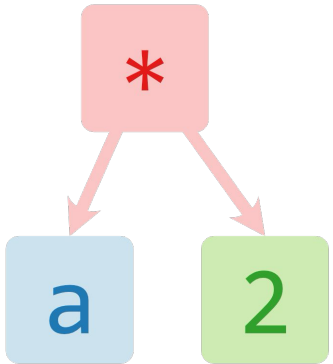
Cranelift



Tamagoyaki

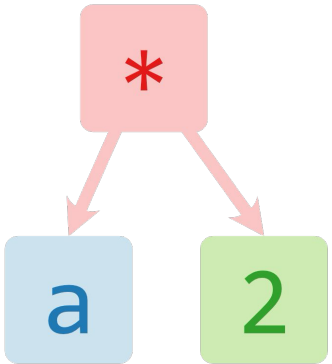
Start with arbitrary MLIR IR

```
func.func @f(%a : i64) -> i64 {

    %two = arith.constant 2 : i64

    %res = arith.muli %a, %two : i64


    func.return %res    : i64


}
```



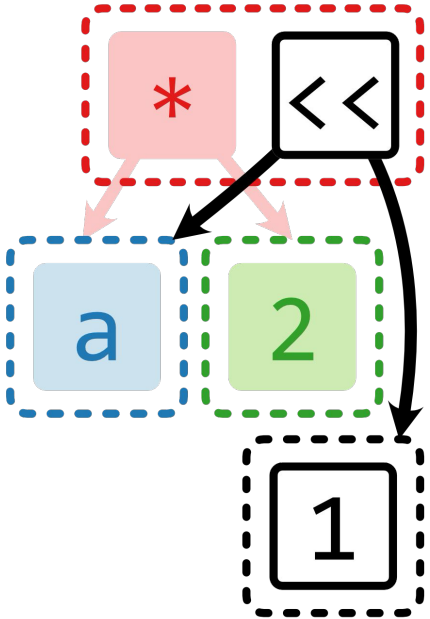
Wrap operations in a graph

```
func.func @f(%a : i64) -> i64 {
  %res = equivalence.graph {
    %two = arith.constant 2 : i64

    %mul = arith.muli %a, %two : i64


    equivalence.yield %mul    : i64
  }
  func.return %res : i64
}
```



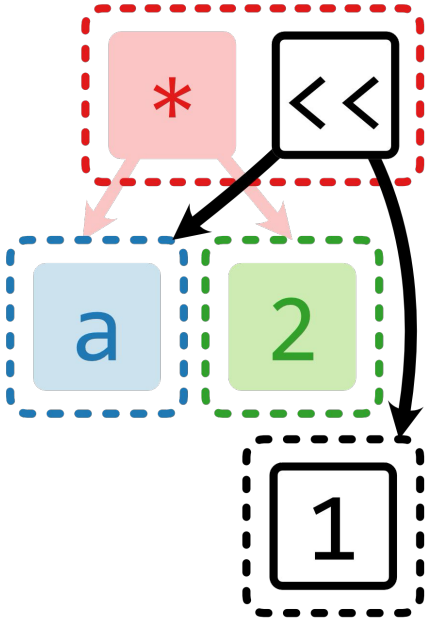
New e-nodes are introduced as operands to e-classes

```
func.func @f(%a : i64) -> i64 {
  %res = equivalence.graph {
    %two = arith.constant 2 : i64
    %one = arith.constant 1 : i64
    %mul = arith.muli %a, %two : i64
    %shl = arith.shli %a, %one : i64

    equivalence.yield %mul : i64
  }
  func.return %res : i64
}
```

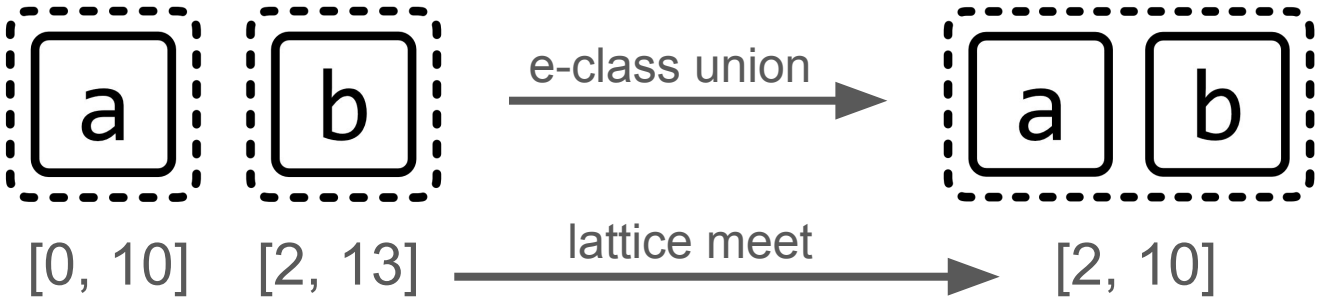


New e-nodes are introduced as operands to e-classes

```
func.func @f(%a : i64) -> i64 {
  %res = equivalence.graph {
    %two = arith.constant 2 : i64
    %one = arith.constant 1 : i64
    %mul = arith.muli %a, %two : i64
    %shl = arith.shli %a, %one : i64
    %c = equivalence.class %mul, %shl : i64
    equivalence.yield %c : i64
  }
  func.return %res : i64
}
```

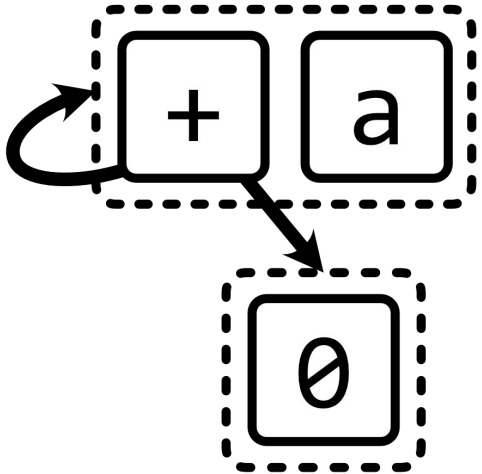


Reuse lattices for e-class analyses



Graph is necessary to handle cycles within expressions

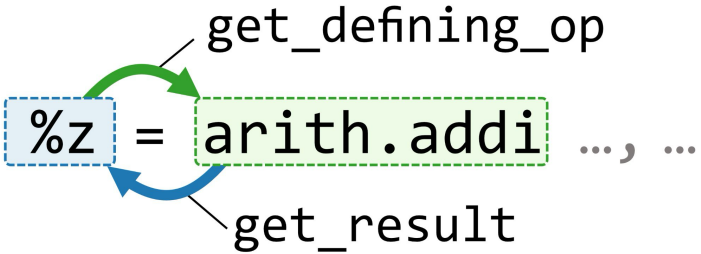
```
equivalence.graph {
    %c_zero = equivalence.class %zero
    %sum = arith.addi %c_a, %c_zero
    %c_a = equivalence.class %sum, %a
}
```



a = a + 0 = (a + 0) + 0 = ...

Adapting rewrite patterns to equality saturation

We have to take into account the use-def indirection



We have to take into account the use-def indirection

New equivalence.class ops require special handling

C++

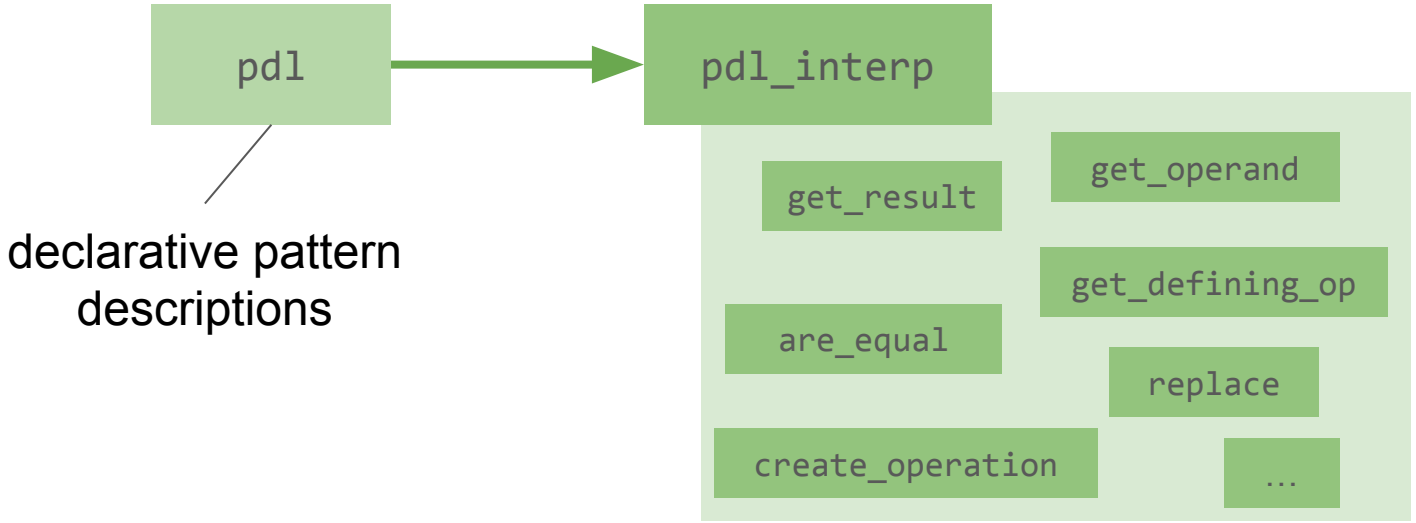
PDL

PDL: MLIR's declarative rewrite representation

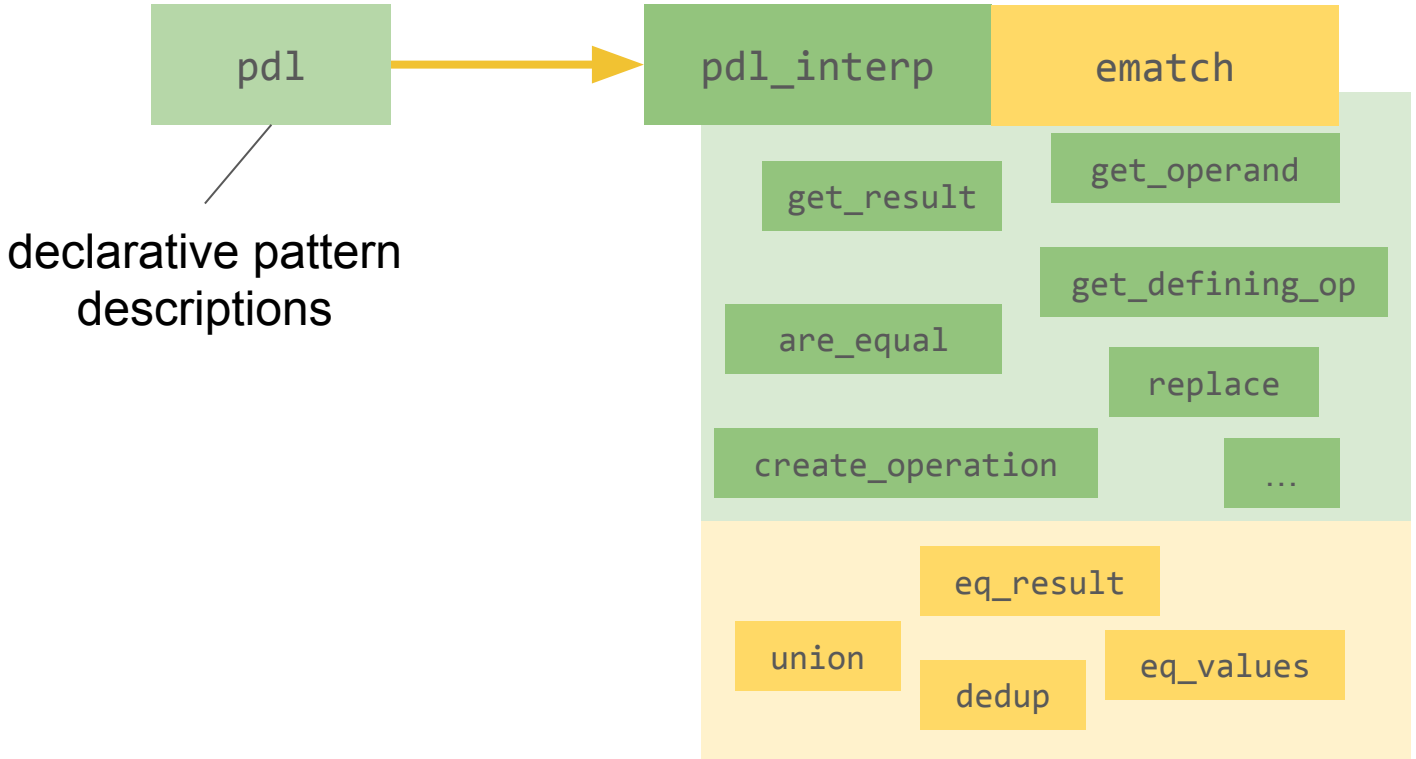
$$a \times 2 \longrightarrow a \ll 1$$

```
pdl.pattern @times_two_shift_one : benefit(2) {
 %t = pdl.type
 %a = pdl.operand
 %c2_attr = pdl.attribute = 2 : i64
 %c2_op = pdl.operation "arith.constant" {"value" = %c2_attr} -> (%t : !pdl.type)
 %c2_res = pdl.result 0 of %c2_op
 %a_mul_2_op = pdl.operation "arith.muli" (%a, %c2_res : !pdl.value, !pdl.value) -> (%t : !pdl.type)
 pdl.rewrite %a_mul_2_op {
   %c1_attr = pdl.attribute = 1 : i64
   %c1_op = pdl.operation "arith.constant" {"value" = %c1_attr} -> (%t : !pdl.type)
   %c1_res = pdl.result 0 of %c1_op
   %a_shift_1_op = pdl.operation "arith.shli" (%a, %c1_res : !pdl.value, !pdl.value) -> (%t : !pdl.type)
   pdl.replace %a_mul_2_op with %a_shift_1_op
 }
}
```

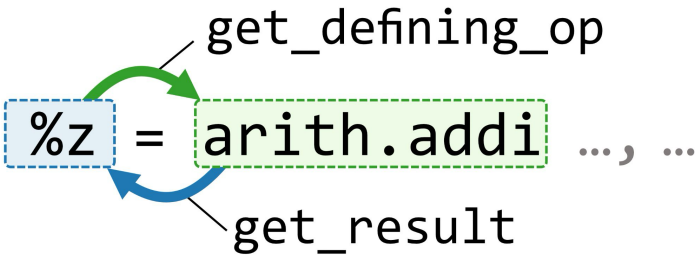
PDL: MLIR’s declarative rewrite representation



We implement e-matching with small modifications



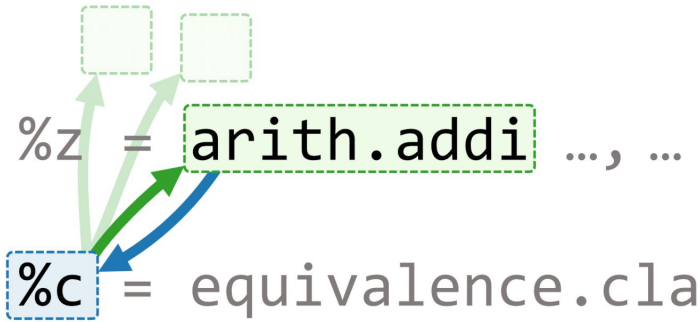
We have to take into account the use-def indirection



get_result

get_defining_op

We have to take into account the use-def indirection



introduce backtracking

get_result
eq_result

eq_values
get_defining_op

~~replace~~
union

create_operation
dedup

maintain hashcons

Case study: Herbie in MLIR

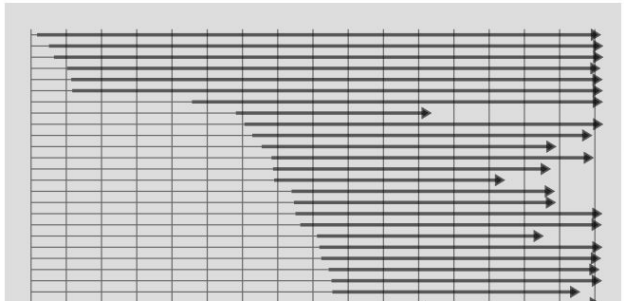
The Herbie Project Try • Install • Learn

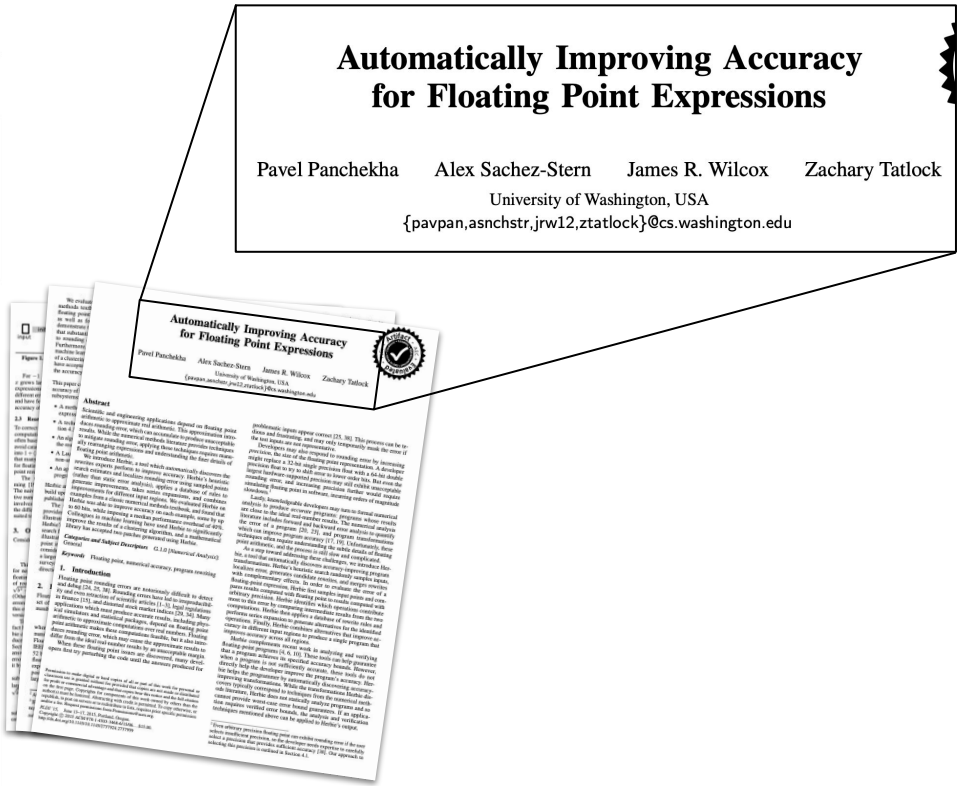
Find and fix floating-point problems:

`sqrt(x+1) - sqrt(x)` → `1/(sqrt(x+1) + sqrt(x))`

Herbie detects inaccurate expressions and finds more accurate replacements. The red expression is inaccurate when $x > 1$; Herbie's replacement, in blue, is accurate for all x.

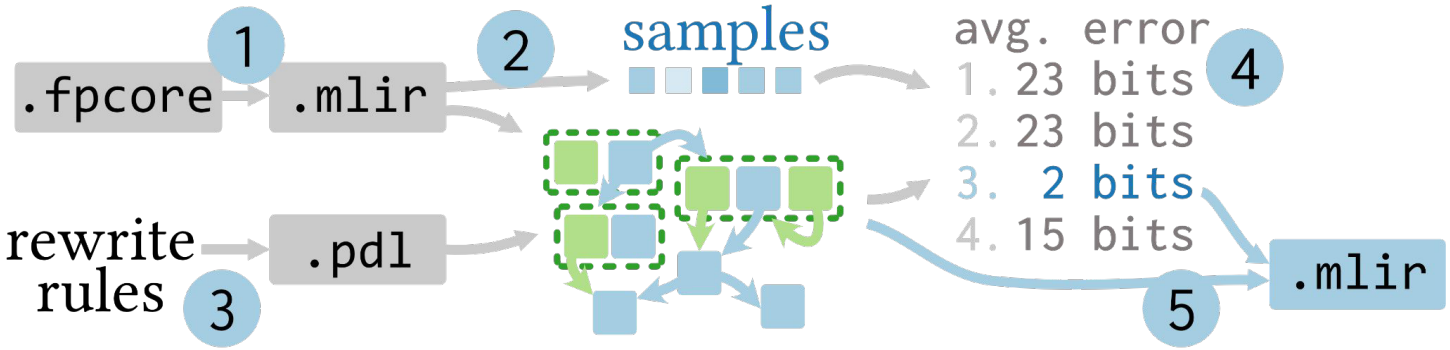
Use	Learn	Contribute
Web demo	Tutorial	Source Code
Install	Release Notes	Report a Bug
News	Documentation	License
Blog	Papers about Herbie	



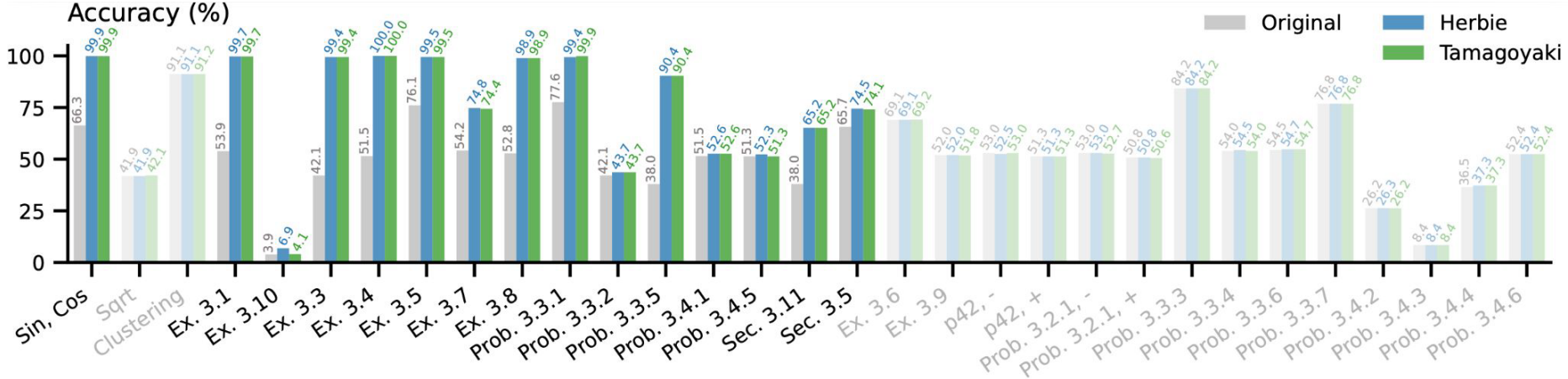


herbie.uwplse.org

We implement Herbie's core in Tamagoyaki

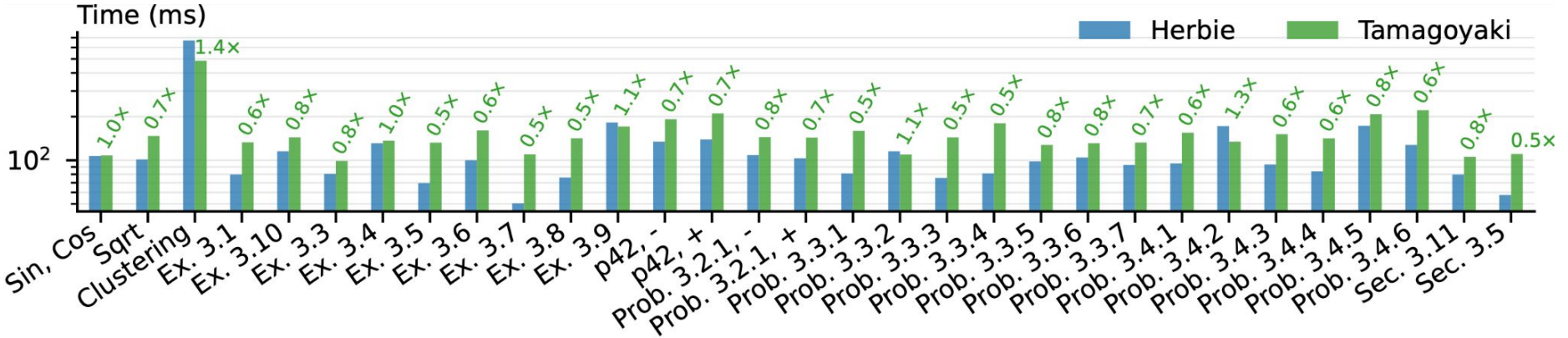


We match Herbie accuracy-wise*



* math target, single iteration, no series expansion, no regime selection

With comparable performance



Case study: RTL Optimization

ROVER: RTL Optimization via Verified E-Graph Rewriting

Samuel Coward, Theo Drane, and George A. Constantinides, *Senior Member, IEEE,*

+

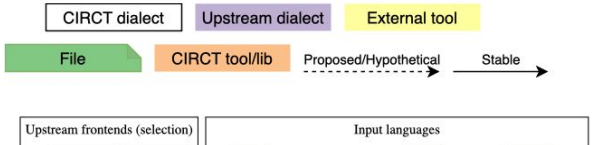
⚡ "CIRCT" / Circuit IR Compilers and Tools

"CIRCT" stands for "Circuit IR Compilers and Tools". One might also interpret it as the recursively as "CIRCT IR Compiler and Tools". The T can be selectively expanded as Tool, Translator, Team, Technology, Target, Tree, Type, ... we're ok with the ambiguity.

The CIRCT community is an open and welcoming community. If you'd like to participate, you can do so in a number of different ways:

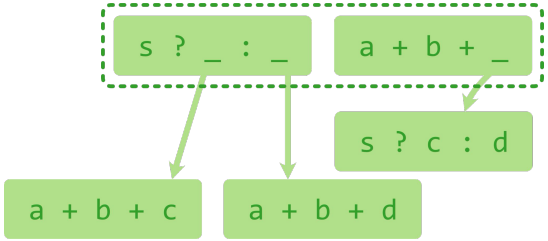
1. Join our Discourse Forum on the LLVM Discourse server. To get a "mailing list" like experience click the bell icon in the upper right and switch to "Watching". It is also helpful to go to your Discourse profile, then the "emails" tab, and check "Enable mailing list mode".
2. For real-time discussion join the CIRCT channel of the LLVM discord server.
3. Join our weekly video chat. Please see the meeting notes document for more information.
4. Contribute code. CIRCT follows all of the LLVM Policies: you can create pull requests for the CIRCT repository, and gain commit access using the standard LLVM policies.

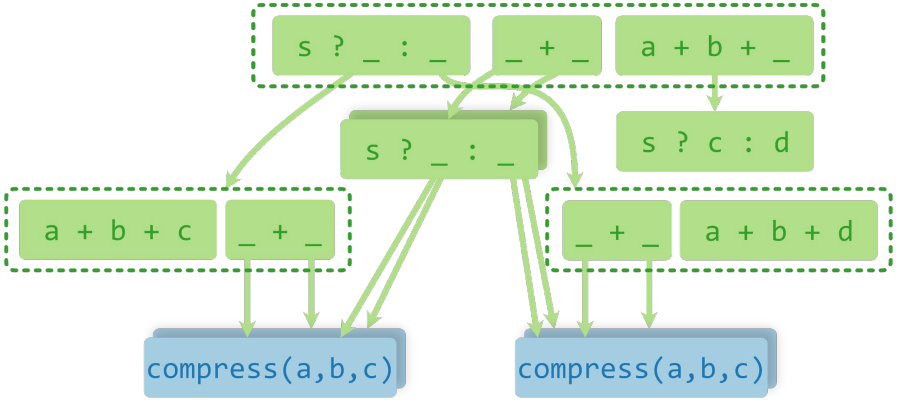
Also take a look at the following diagram, which gives a brief overview of the current dialects and how they interact:



circt.llvm.org

CIRCT has lower-level dialects than Verilog



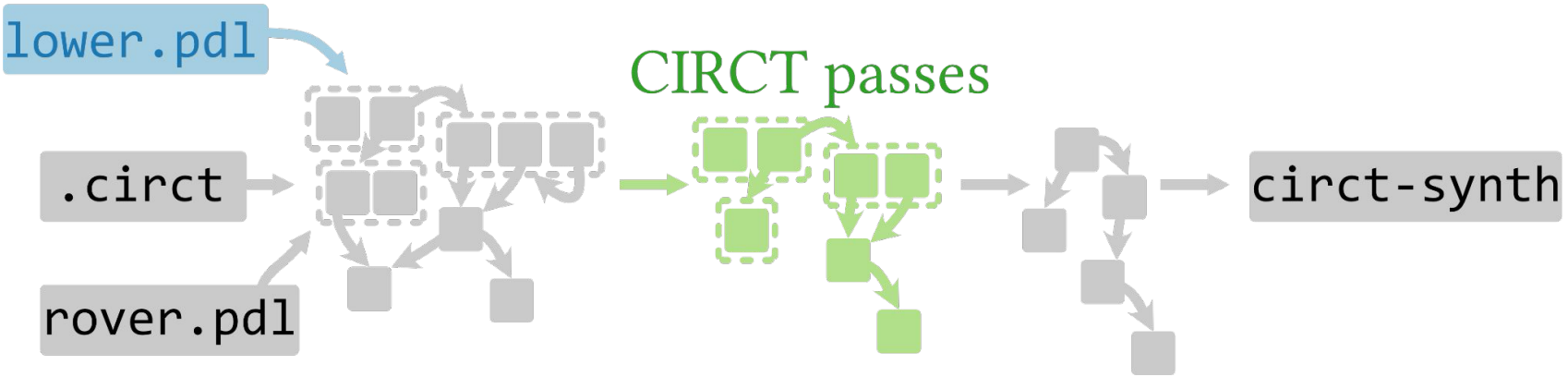


equivalence.class

comb dialect

datapath dialect

We also run existing CIRCT passes on the e-graph



This allows finding better hardware designs

Benchmark	No EqSat		Single-Level			**Multi-Level**			**Multi-Level + CIRCT Passes**		
	Area	Delay	Area	Delay	Opt. Time	Area	Delay	Opt. Time	Area	Delay	Opt. Time
FirFilter	**230**	963	330	**674**	4	330	**674**	4	330	**674**	10
Adpcm	159	357	**89**	**286**	6	**89**	**286**	5	**89**	**286**	13
ShiftedFma	910	819	910	819	4	**857**	**727**	4	**857**	**727**	17
ShiftMult	1339	741	825	771	4	1339	741	7	**775**	**687**	38

Try out Tamagoyaki in your projects!

Your PDL Rewrites +  + **Your Cost Model**