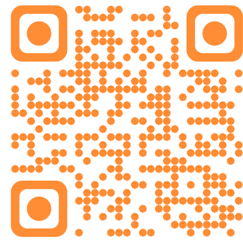
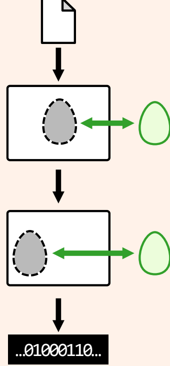
Tamagoyaki

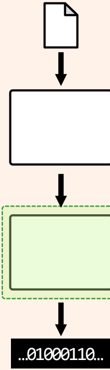
E-Graphs as a Persistent Compiler Abstraction

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

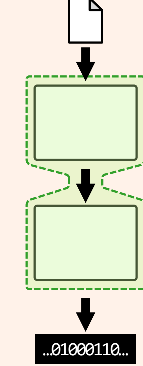


Existing work leverages equality saturation for code optimization by converting parts of the program into an e-graph and using external frameworks. This requires (lossy) conversions from code to e-graph and back. Importantly, ***equivalence information is lost*** when an optimized expression is extracted inserted back in the program.





Cranelift integrates equality saturation more tightly in the compiler backend. This allows for reusing compiler infrastructure and subsuming some compiler passes. However, ***Cranelift is not extensible***. Users cannot easily add their own operations, and general cyclic equality saturation is not supported.



Tamagoyaki is an ***extensible equality saturation framework that embeds equivalences in MLIR IR***. By representing equivalences explicitly in code, they can persist throughout compilation. Furthermore, with MLIR, users can easily run equality saturation on their own custom dialects. Tamagoyaki can also represent cycles in the e-graph.

operation name · block argument · attribute · type

```
func.func @function(%a: i64) -> i64 {
  %zero = arith.constant 0 : i64
  %res = arith.addi %0, %zero : i64
  func.return %res : i64
}
```

region · result · dialect name · operands

We build on MLIR which offers an extensible IR format.

```
func.func @f(%a : i64) → i64 {
  %res = equivalence.graph {
    %two = arith.constant 2
    %mul = arith.muli %a, %two
    equivalence.yield %mul
  }
  func.return %res
}
```

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

We introduce the **equivalence** dialect which contains operations to represent an e-graph. First, **graph** and **yield** operations are used to enclose the e-graph.

Rewrites can be applied that introduce new operations to the IR. For example: x * 2 ⟶ x << 1 leads to the insertion of a shift operation. The result value of this shift is added as an operand to a **class** operation. Every operand to such a class is considered equivalent.

```
%a = class(%x, …)        %a = class(%x, …)        %a = class(%x, %y, …)        %a = class(%x, %y, …)
%b = f(%a)               %b = f(%a)               %b = f(%a)                   %b = f(%a)
%c = class(%y, …)        %c = class(%a, %y, …)    %c = class(%a, %y, …)
%d = f(%c)               %d = f(%c)               %d = f(%a)                   %d = f(%a)
```

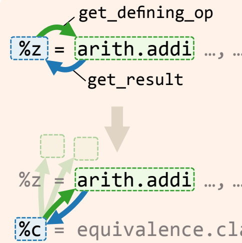
The relation between e-class and e-nodes, as well as the union-find data structure, is ***encoded directly in SSA IR***.

We build on **pdl**, an MLIR dialect for declaratively expressing rewrite rules.

This means that ***existing patterns can be used to do both destructive and non-destructive rewriting***.

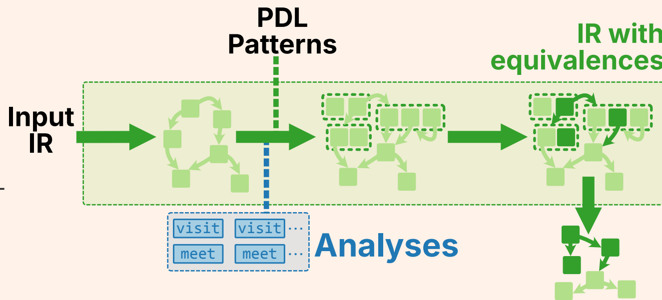
```
a & a ⟶ a

pdl.pattern : benefit(1) {
  %0 = pdl.type
  %a = pdl.operand
  %1 = pdl.operation "arith.andi"(
         %a, %a: !pdl.value, !pdl.value
       ) → (%0: !pdl.type)
  pdl.rewrite %1 {
    pdl.replace %1 with (%a: !pdl.value)
  }
}
```



equivalence.class operations bring indirections in the defining-operation-to-result relation. We extend lowering from **pdl** to **pdl_interp** to insert operations from our ***ematch dialect***. These operations ensure that every equivalent value in classes is considered.
Additionally, we adapt the pdl lowering heuristics to prevent exponential blowup in matching complexity.

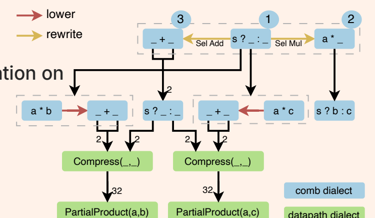
Classical steps in equality saturation correspond to compiler passes on IR containing equivalence operations, using user-provided rewrite patterns.
Different steps include e-graph insertion, e-matching and rebuilding, extraction, and e-class analyses.

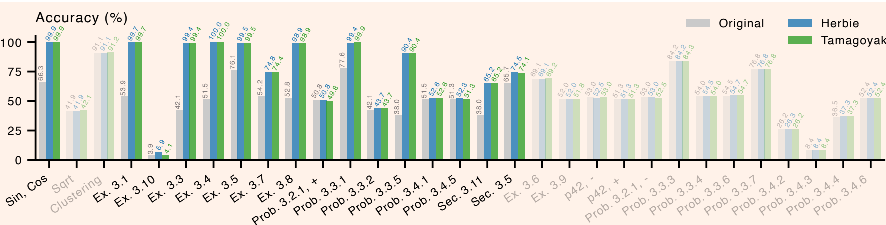


	circt-synth		ROVER			Tamagoyaki		
	Area	Delay	Area	Delay	EqSat	Area	Delay	EqSat
FirFilter	**230**	963	302	**807**	15	302	**807**	18
AdpcmDecoder	159	357	**89**	**286**	14	**89**	**286**	19
ShiftedFma	910	819	910	819	13	**857**	**727**	15
ShiftMult	1339	741	825	771	14	**775**	**687**	17

Area (µm²), Delay (ns), EqSat Time (ms). Bold = best.

Rover is an existing tool that uses equality saturation for optimizing RTL circuit designs. We build on CIRCT, an MLIR project for hardware design to replicate this work. CIRCT contains dialects that go beyond what is representable at RTL level, this allows us to ***find optimized designs that outperform ROVER***.
Additionally, we show that running additional passes such as canonicalization on the persistent e-graph leads to more accurate cost modeling.





We replicate the core procedure of Herbie, a floating point accuracy optimizer that uses equality saturation.
We are able to reach similar accuracy to Herbie. Currently, we have a geomean runtime slowdown of 1.56x.
This proof-of-concept implementation shows that ***techniques such as Herbie can be integrated more tightly in MLIR***.