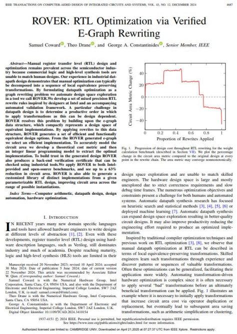
Equality Saturation and Industrial Circuit Design

Sam Coward - Postdoc with Alexandra Silva @ University College London

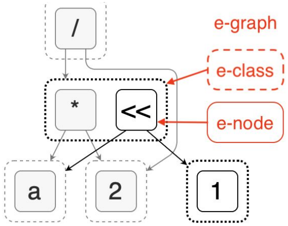
Disclaimer: not a computer scientist!

Introduction

IMPERIAL

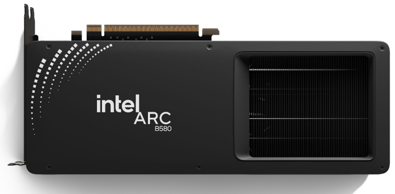

ROVER: RTL Optimization via Verified E-Graph Rewriting

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



intel

Numerical HW Group (April 2025)



Number Representation:

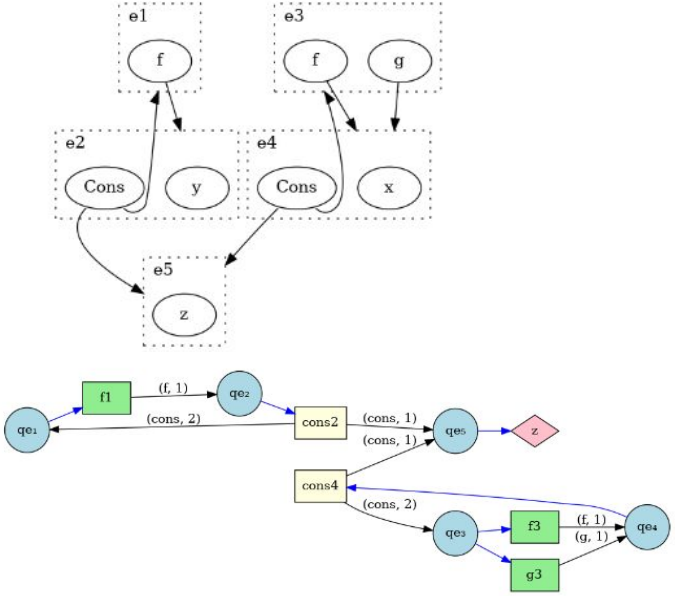
SystemVerilog

8-bit: E5M2, E4M3
4-bit: E3M0, E2M1

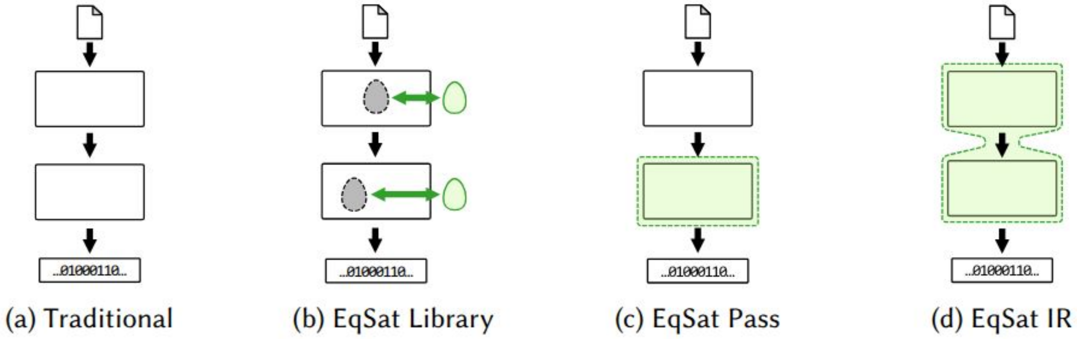
1. Landscape: E-Graphs and Equality Saturation
2. My Past: E-Graphs for Circuit Design
3. Now: Open-Source EDA

Current Projects

E-Graphs as Automata

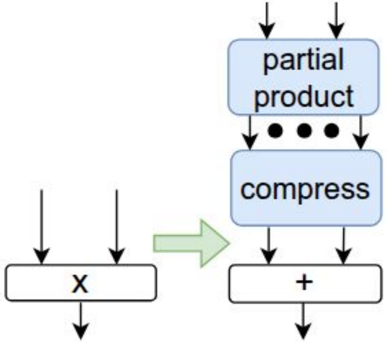


Equality Saturation & MLIR



Open-Source Circuit Compilers

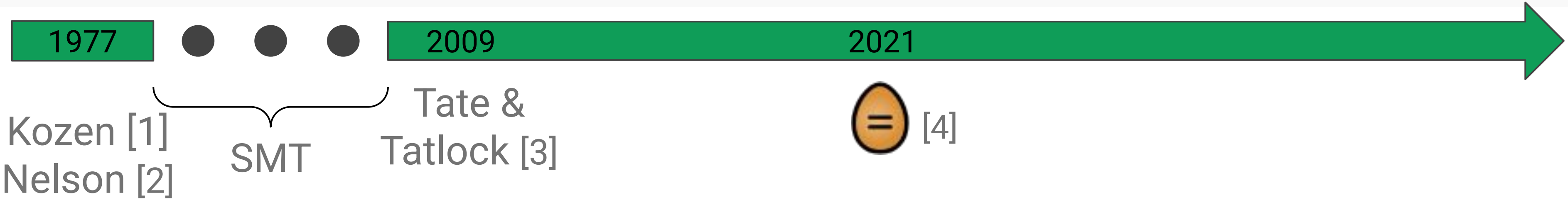
CIRCT



E-Graphs & Equality Saturation

data struct — program rewriting technique

[1] Complexity of Finitely Presented Algebras, Kozen
[2] Techniques for Program Verification, Nelson
[3] Equality Saturation: a new approach to optimization, Tate et al.
[4] egg: Fast and extensible equality saturation, Willsey et al.



E-Graphs & Equality Saturation

data struct

program rewriting technique

[1] Complexity of Finitely Presented Algebras, Kozen
[2] Techniques for Program Verification, Nelson
[3] Equality Saturation: a new approach to optimization, Tate et al.
[4] egg: Fast and extensible equality saturation, Willsey et al.

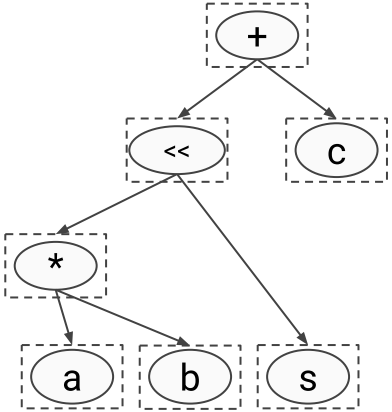
1977 ● ● ● 2009 2021

Kozen [1]
Nelson [2]

SMT

Tate &
Tatlock [3]

[4]



((a * b) << s) + c

E-Graphs & Equality Saturation

data struct — program rewriting technique

[1] Complexity of Finitely Presented Algebras, Kozen
[2] Techniques for Program Verification, Nelson
[3] Equality Saturation: a new approach to optimization, Tate et al.
[4] egg: Fast and extensible equality saturation, Willsey et al.

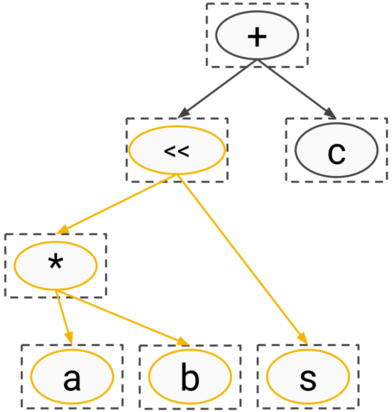
1977 ● ● ● 2009 2021

Kozen [1]
Nelson [2]

SMT

Tate & Tatlock [3]

[4]



(?x * ?y) << ?z →

((a * b) << s) + c

E-Graphs & Equality Saturation

E-Graphs: data struct

Equality Saturation: program rewriting technique

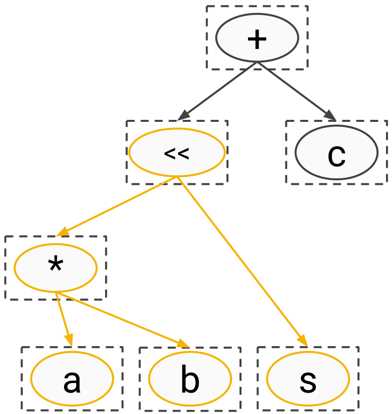
[1] Complexity of Finitely Presented Algebras, Kozen
[2] Techniques for Program Verification, Nelson
[3] Equality Saturation: a new approach to optimization, Tate et al.
[4] egg: Fast and extensible equality saturation, Willsey et al.

Kozen [1]
Nelson [2]

SMT

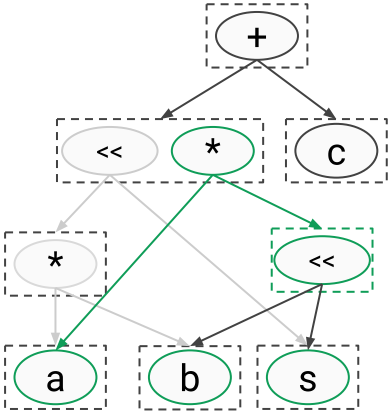
Tate & Tatlock [3]

[4]



((a * b) << s) + c

(?x * ?y) << ?z →
?x * (?y << ?z)



E-Graphs & Equality Saturation

data struct

program rewriting technique

[1] Complexity of Finitely Presented Algebras, Kozen
[2] Techniques for Program Verification, Nelson
[3] Equality Saturation: a new approach to optimization, Tate et al.
[4] egg: Fast and extensible equality saturation, Willsey et al.

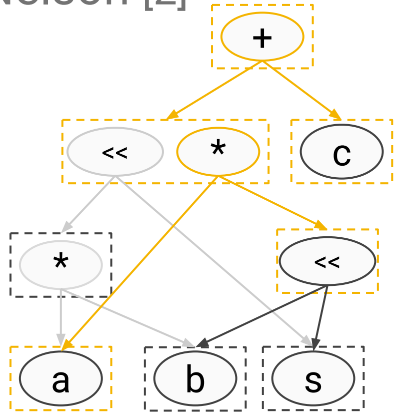
1977 ● ● ● 2009 2021

Kozen [1]
Nelson [2]

SMT

Tate & Tatlock [3]

[4]



(?x * ?y) + ?z →

((a * b) << s) + c

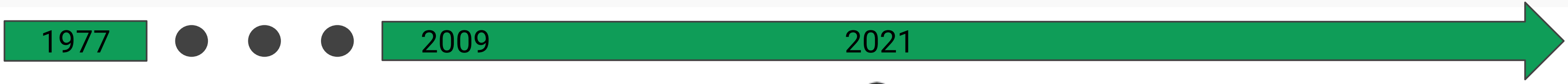
(a*(b<<s)) + c

E-Graphs & Equality Saturation

data struct

program rewriting technique

[1] Complexity of Finitely Presented Algebras, Kozen
[2] Techniques for Program Verification, Nelson
[3] Equality Saturation: a new approach to optimization, Tate et al.
[4] egg: Fast and extensible equality saturation, Willsey et al.

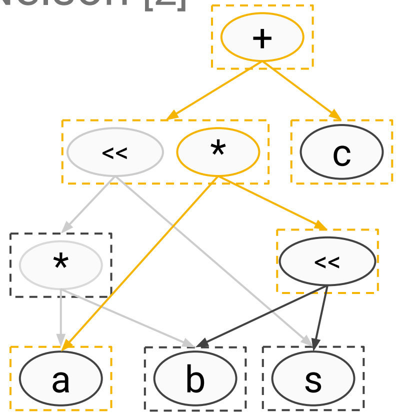


Kozen [1]
Nelson [2]

SMT

Tate & Tatlock [3]

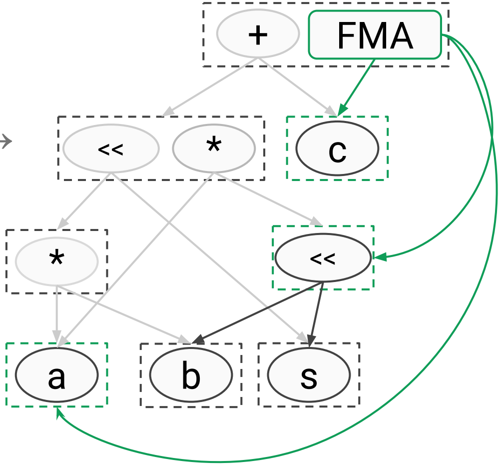
[4]



((a * b) << s) + c

(a*(b<<s)) + c

(?x * ?y) + ?z →
FMA(?x,?y,?z)



Congruence Closure:
x=y ⇒ f(x) = f(y)

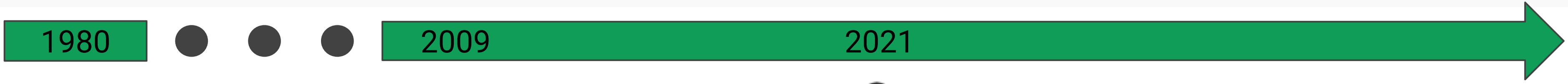
Applications:

➢ Proof tactics
➢ Numerical stability
➢ Compilers
➢ Synthesis tasks

E-Graphs & Equality Saturation

data struct — program rewriting technique

[1] Techniques for Program Verification, Nelson
[2] Complexity of Finitely Presented Algebras, Kozen
[3] Equality Saturation: a new approach to optimization, Tate et al.
[4] egg: Fast and extensible equality saturation, Willsey et al.

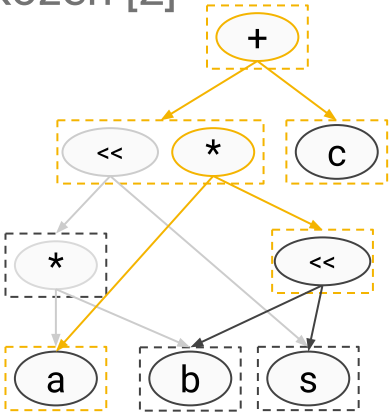


Nelson [1]
Kozen [2]

SMT

Tate & Tatlock [3]

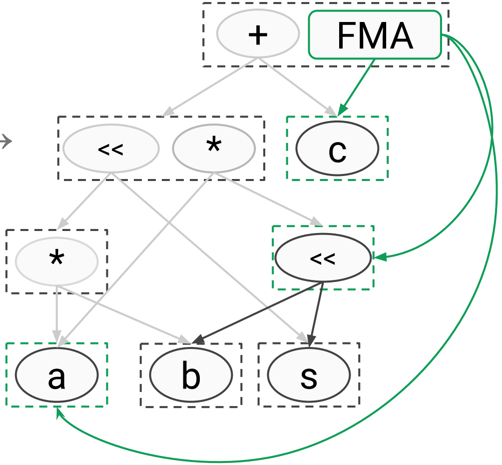
[4]



((a * b) << s) + c

(a*(b<<s)) + c

(?x * ?y) + ?z →
FMA(?x,?y,?z)



E-Graphs:

- ★ Datalog
- ★ Context
- ★ Efficient extraction

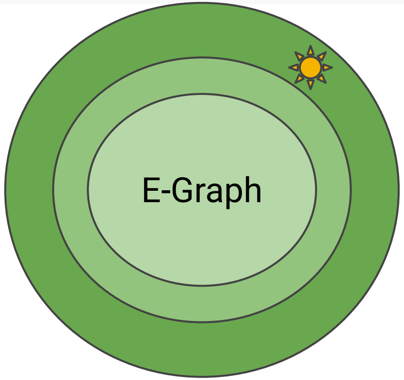
Applications:

- ➢ Proof tactics
- ➢ Numerical stability
- ➢ Compilers
- ➢ Synthesis tasks

Why so popular? What's left?

https://egraphs.org/
Monthly Seminars

- Removes scheduling
- Fast enough…
- Well built library



Scalability

- Destructive rewriting
- Iterative equality saturation

Approximate Equivalence?

sigmoid(x) -> ax^2 + bx + c

Accuracy vs performance?

Cycles and Analyses?

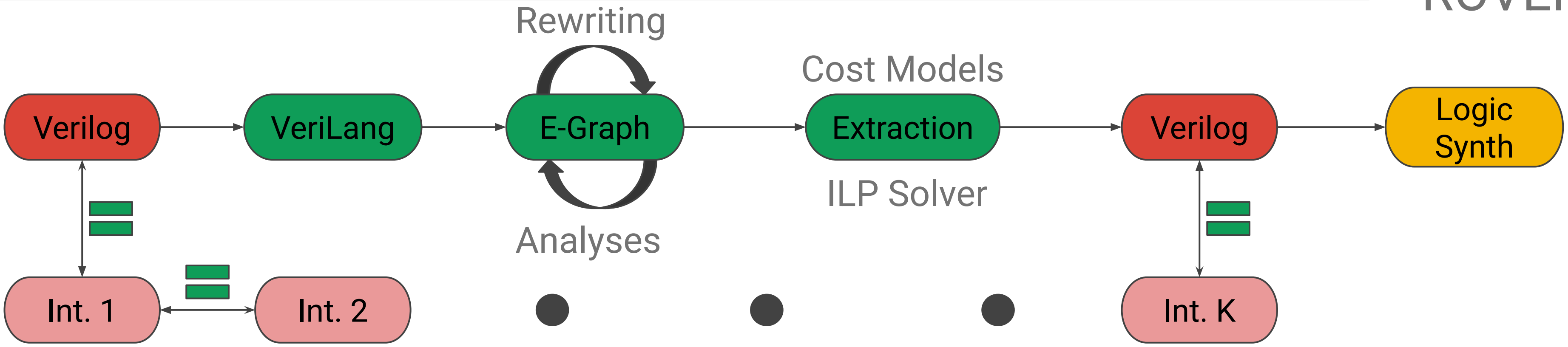
$$\frac{1}{1-x} \rightarrow 1 + x \times \left(\frac{1}{1-x}\right)$$

Lift analysis to e-classes

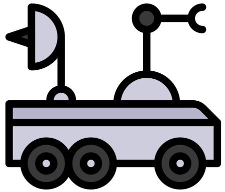
Past:
E-Graphs for Industrial Circuit Design

E-Graphs & Circuit Design

ROVER



E-Graphs & Circuit Design



Verilog:

```
assign z[8:0] = x[7:0] + y[4:0]
```

VeriLang:

```
(+ 9 8 unsign x 5 unsign y)
```

CIRCT Project:

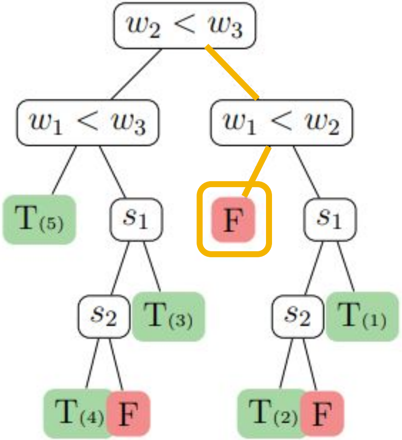
```
%c0_i4 = hw.constant 0 : i4
%false = hw.constant false
%0 = comb.concat %false, %x : i1, i8
%1 = comb.concat %c0_i4, %y : i4, i5
%2 = comb.add %0, %1 : i9
```

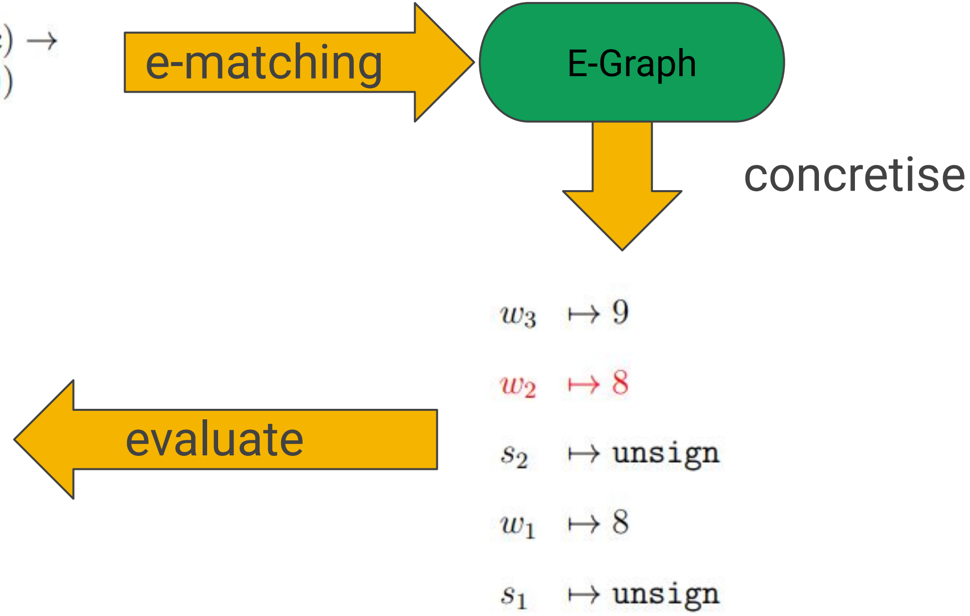
VeriLang Rewriting: Associativity

Rewrite:

$$(+\ w_3\ w_2\ s_2\ (+\ w_2\ w_1\ s_1\ \mathbf{a}\ w_1\ s_1\ \mathbf{b})\ w_1\ s_1\ \mathbf{c}) \rightarrow$$
$$(+\ w_3\ w_1\ s_1\ \mathbf{a}\ w_2\ s_2\ (+\ w_2\ w_1\ s_1\ \mathbf{b}\ w_1\ s_1\ \mathbf{c}))$$

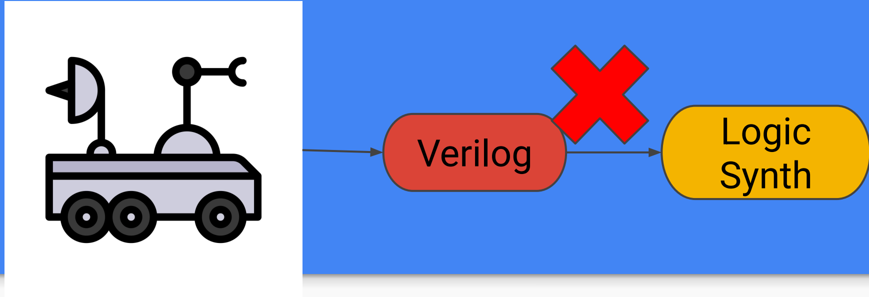
Predicate:



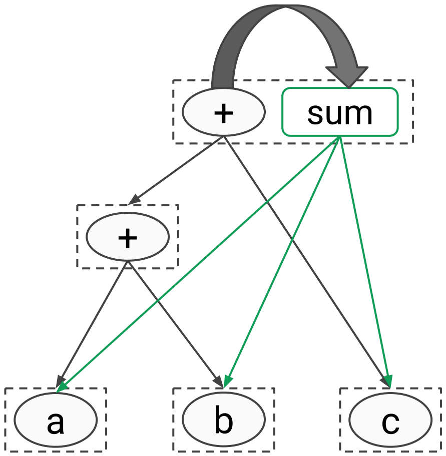


Goal: necessary & sufficient predicate ⇒ no missed opportunities & correct

Downstream Correlation



47% faster, same area

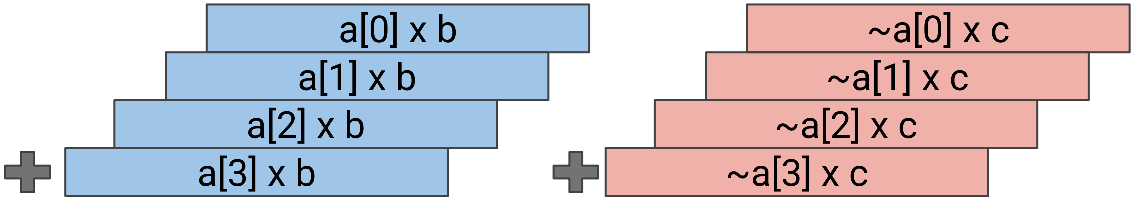


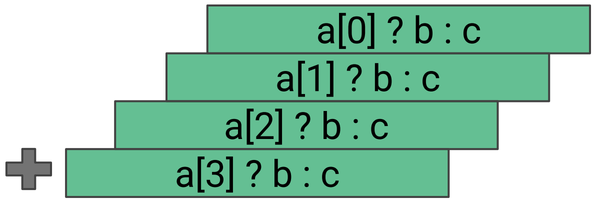
a+b+c
a+b+d

Localise cost modeling via abstraction

Graphics Blend: 2 multipliers, 1 adder

$$a \times b + \sim a \times c$$





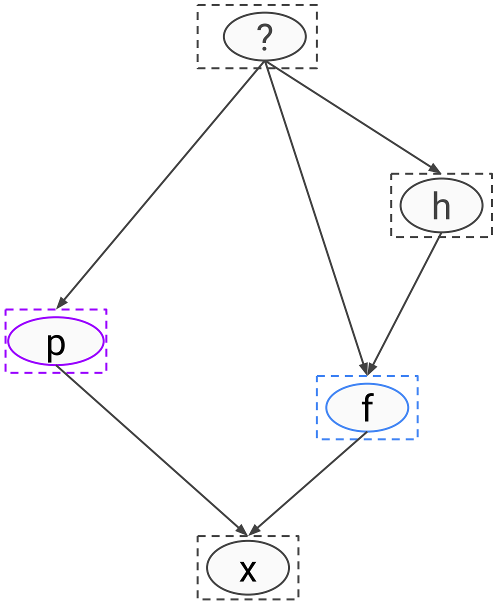
1 multiplier

Context-Aware E-Graph Rewriting

y = p(x) ? f(x) : h(f(x))

Compression ⇒ treat all uses of f(x) the same

p(x) ⇒ f(x) → c

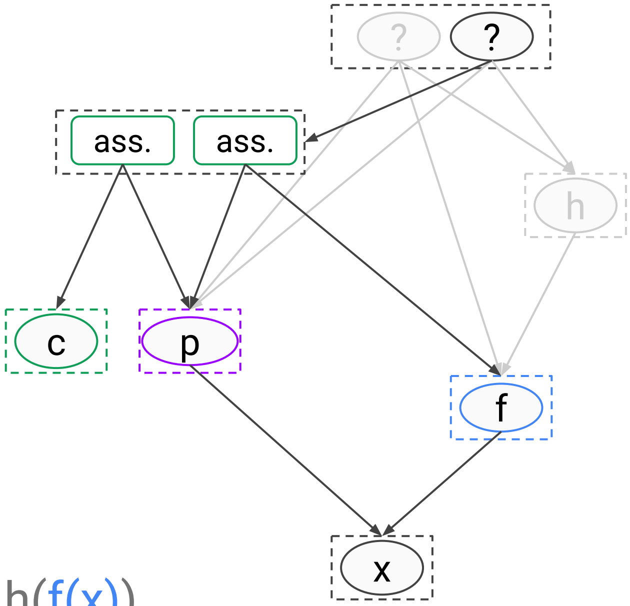


Context-Aware E-Graph Rewriting

y = p(x) ? f(x) : h(f(x))

Compression ⇒ treat all uses of f(x) the same

p(x) ⇒ f(x) → c



Solution:

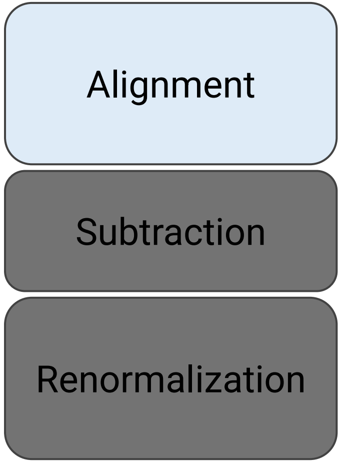
p(x) ? f(x) : h(f(x)) → p(x) ? assume(f(x), p(x)) : h(f(x))

assume(f(x), p(x)) → assume(c, p(x))

Floating-Point Subtraction Circuit

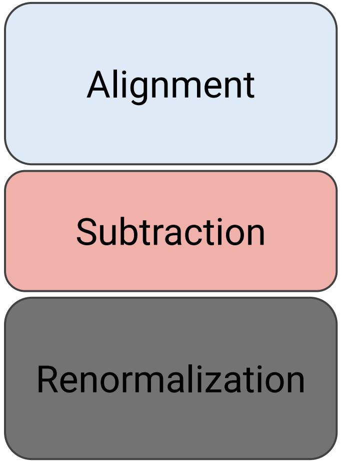
wlog a > b

$$2^{ea} \times 1.ma - 2^{eb} \times 1.mb = 2^{ea}\left(1.ma - \frac{1.mb}{2^{ea-eb}}\right) = 2^{ec} \times 1.mc$$



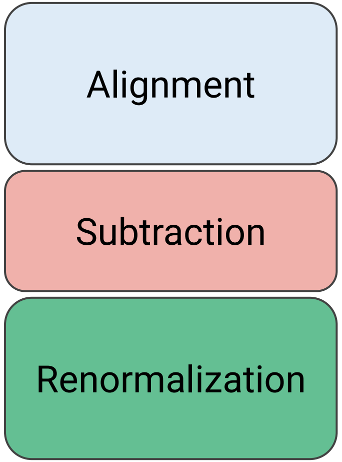
Floating-Point Subtraction Circuit

$$2^{ea} \times 1.ma - 2^{eb} \times 1.mb = 2^{ea}\left(1.ma - \frac{1.mb}{2^{ea-eb}}\right) = 2^{ec} \times 1.mc$$



Floating-Point Subtraction Circuit

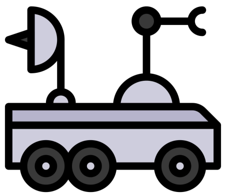
$$2^{ea} \times 1.ma - 2^{eb} \times 1.mb = 2^{ea}\left(1.ma - \frac{1.mb}{2^{ea-eb}}\right) = 2^{ec} \times 1.mc$$



Floating-Point Subtraction Circuit

$$2^{ea} \times 1.ma - 2^{eb} \times 1.mb = 2^{ea}\left(1.ma - \frac{1.mb}{2^{ea-eb}}\right) = 2^{ec} \times 1.mc$$

assume Alignment | assume Alignment
assume Subtraction | assume Subtraction
assume Renormalization | assume Renormalization
ea-eb ≤ 1 ?



ROVER

(10 seconds)

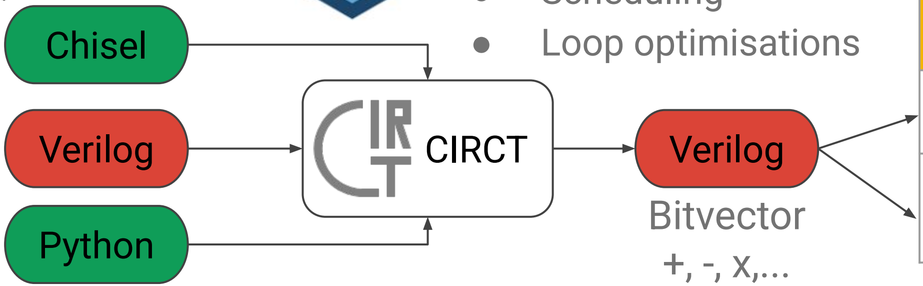
Near-Path | Far-Path
Alignment | Alignment
Subtraction | Subtraction
Renormalization | Renormalization
ea-eb ≤ 1 ?

20% faster!!!

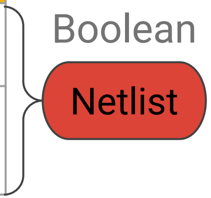
Now:
Open-Source Datapath Synthesis

High-Performance Datapath Synthesis (ASIC)

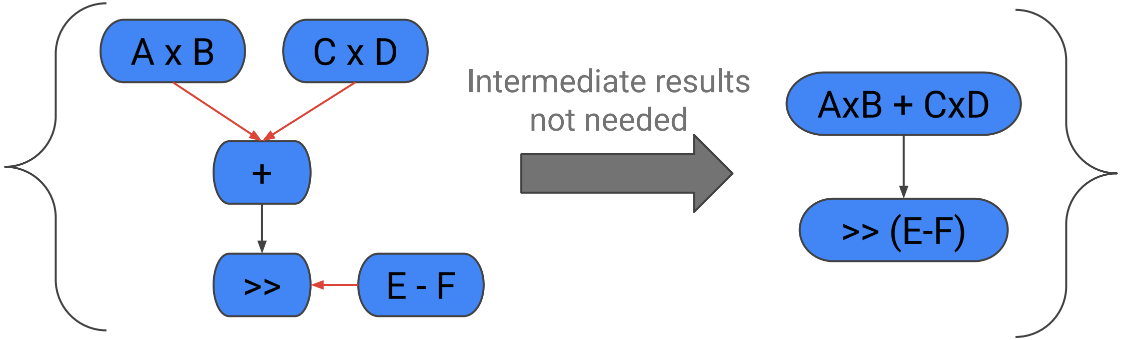
- Verification generators
- Scheduling
- Loop optimisations



Tool	Cost	Productivity	PPA
Yosys & ABC			
Design Compiler / Genus			



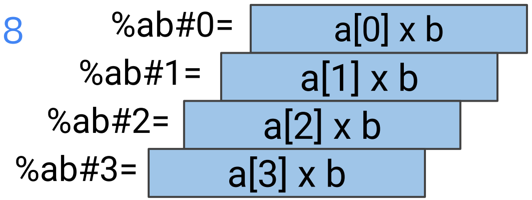
Open-Source



Closed-Source

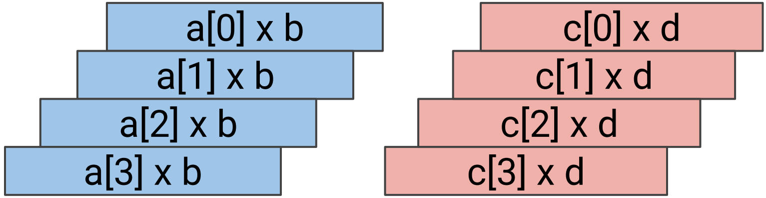
Datapath Dialect: (a x b) + (c x d)

```
%ab:4 = datapath.partial_product %a, %b : i8
```



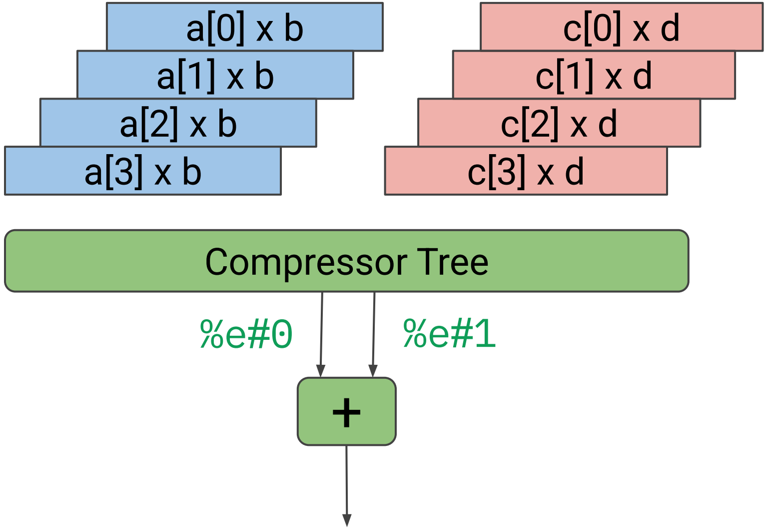
Datapath Dialect: (a x b) + (c x d)

```
%ab:4 = datapath.partial_product %a, %b : i8
%cd:4 = datapath.partial_product %c, %d : i8
```



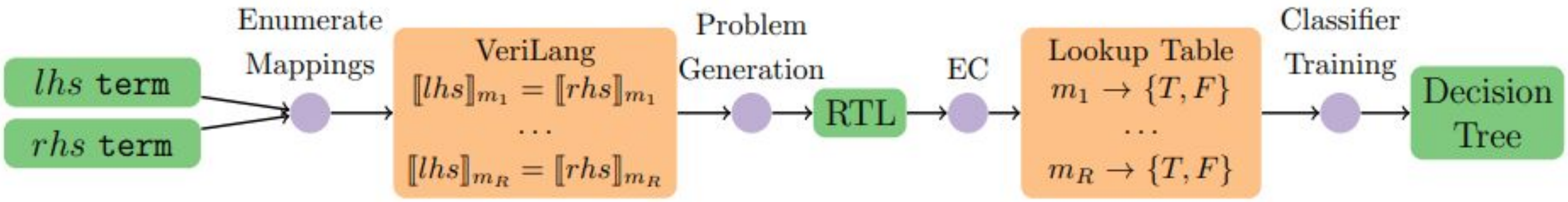
Datapath Dialect: (a x b) + (c x d)

```
%ab:4 = datapath.partial_product %a, %b : i8
%cd:4 = datapath.partial_product %c, %d : i8

// Compress to carry-save form
%e:2 = datapath.compress %ab#0, …, %ab#3,
                         %cd#0, …, %cd#3
                        : i8 [8 -> 2]

// (carry + save) = a*b + c*d
%result = comb.add %e#0, %e#1 : i8
```

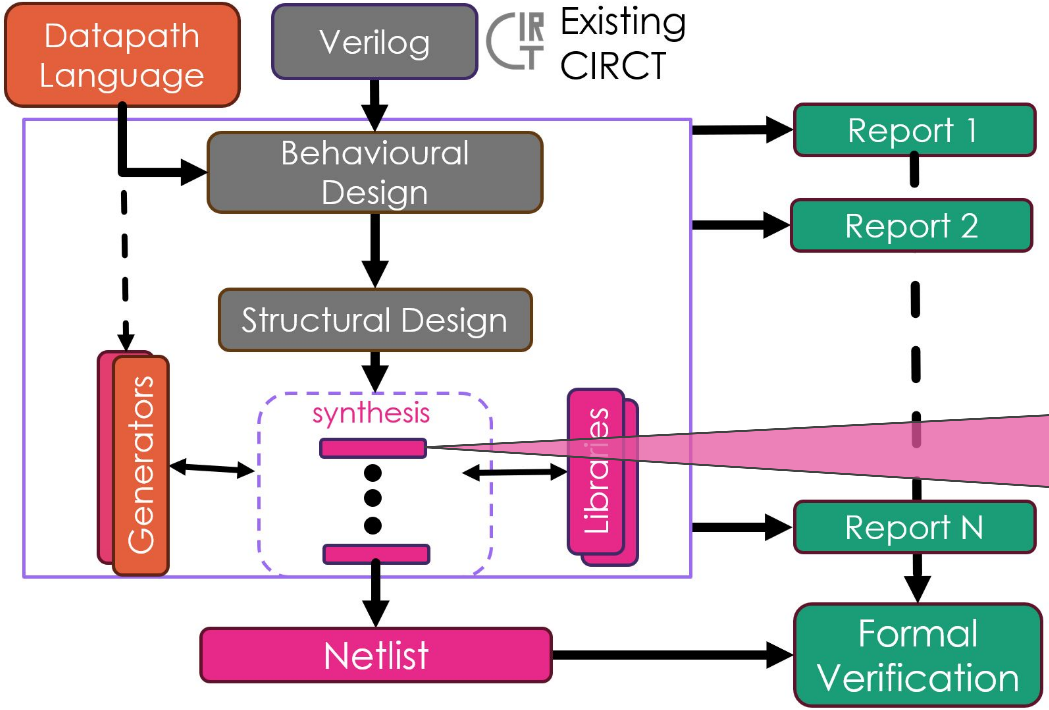


Backup

Condition Synthesis



Vision: Open-Source Datapath Synth



- Outperform commercial tools?
- Interactive design flow
- Formal guarantees

Progress To-Date

- Datapath dialect
- Datapath analyses
- Industrial/Academic buy-in