Knuth-Bendix Completion for Program Optimization

Thesis Proposal

Michael Schifferer

20.08.2025

What Kind of Program Optimization?

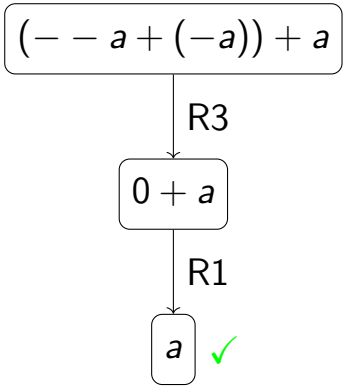
Rewrite rules:

R1: $0 + X \to X$

R2: $X + 0 \to X$

R3: $-X + X \to 0$

R4: $(X + Y) + Z \to X + (Y + Z)$



What's the Problem?

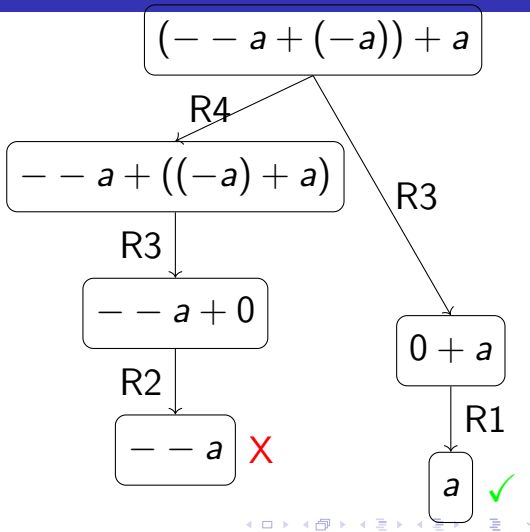
Rewrite rules:

R1: $0 + X \to X$

R2: $X + 0 \to X$

R3: $-X + X \to 0$

R4: $(X + Y) + Z \to X + (Y + Z)$



What's the Problem?

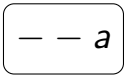
Rewrite rules:

R1: $0 + X \rightarrow X$

R2: $X + 0 \rightarrow X$

R3: $-X + X \rightarrow 0$

R4: $(X + Y) + Z \rightarrow X + (Y + Z)$



What's the Problem?

Rewrite rules:

R1: $0 + X \leftrightarrow X$

R2: $X + 0 \leftrightarrow X$

R3: $-X + X \leftrightarrow 0$

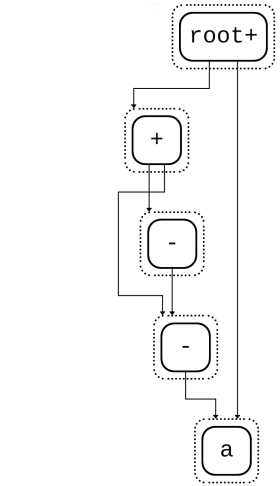
R4: $(X + Y) + Z \leftrightarrow X + (Y + Z)$

Turns our nice DAG into an infinite undirected graph

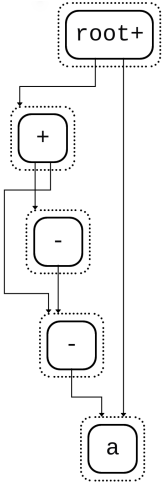
Equality Saturation

$$(--a + (-a)) + a$$

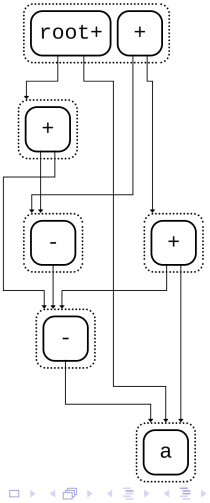
to e-graph



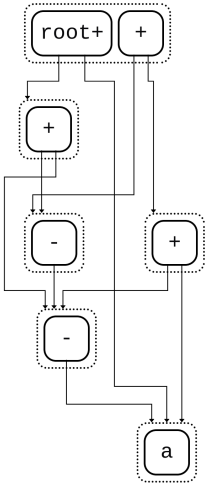
Equality Saturation



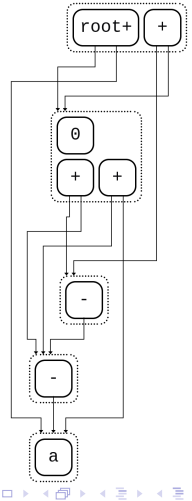
$$(X + Y) + Z \rightarrow X + (Y + Z)$$



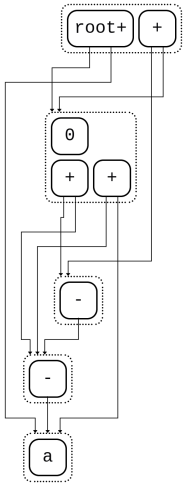
Equality Saturation



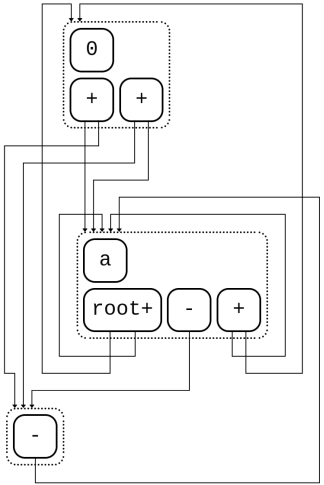
$$-X + X \rightarrow 0$$



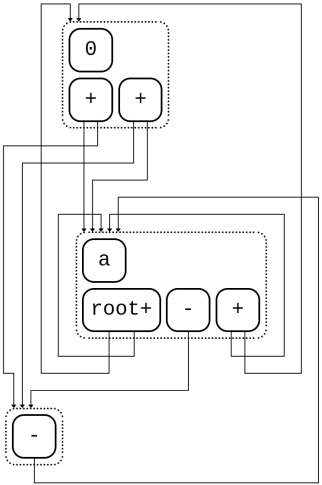
Equality Saturation



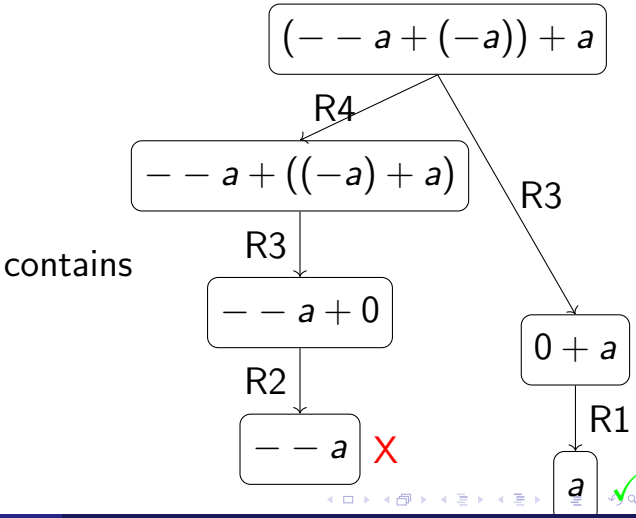
$$0 + X \to X \text{ and } X + 0 \to X$$



Equality Saturation



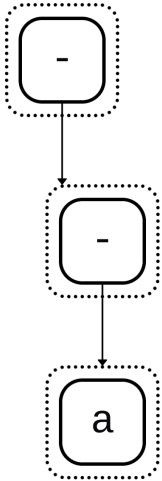
contains



Equality Saturation

$$- - a$$

to e-graph



Rewrite rules:
R1: $0 + X \to X$
R2: $X + 0 \to X$
R3: $-X + X \to 0$
R4: $(X + Y) + Z \to X + (Y + Z)$

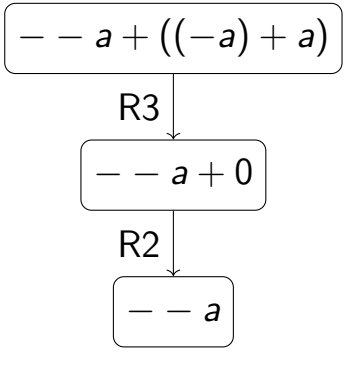
Knuth-Bendix Completion

Prove:

$$--a + ((-a) + a) = 0 + a$$

Rewrite rules:

R1: $0 + X \to X$
R2: $X + 0 \to X$
R3: $-X + X \to 0$
R4: $(X + Y) + Z \to X + (Y + Z)$



Superposition

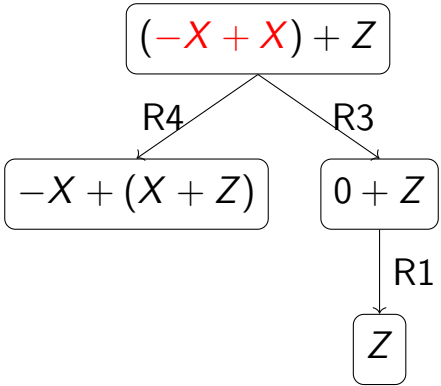
Rewrite rules:

R1: $0 + X \to X$

R2: $X + 0 \to X$

R3: $-X + X \to 0$

R4: $(X + Y) + Z \to X + (Y + Z)$



Superposition

Rewrite rules:

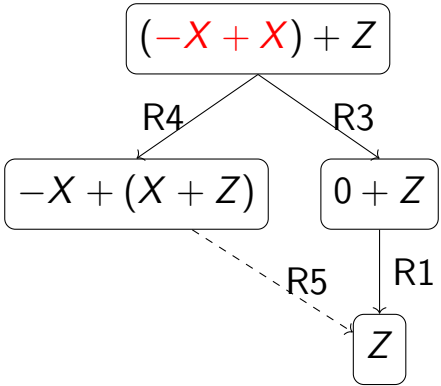
R1: $0 + X \to X$

R2: $X + 0 \to X$

R3: $-X + X \to 0$

R4: $(X + Y) + Z \to X + (Y + Z)$

R5: $-X + (X + Z) \to Z$



Superposition

Rewrite rules:

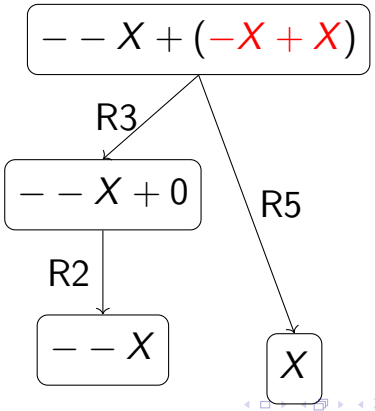
R1: $0 + X \to X$

R2: $X + 0 \to X$

R3: $-X + X \to 0$

R4: $(X + Y) + Z \to X + (Y + Z)$

R5: $-X + (X + Z) \to Z$



Superposition

Rewrite rules:

R1: $0 + X \to X$

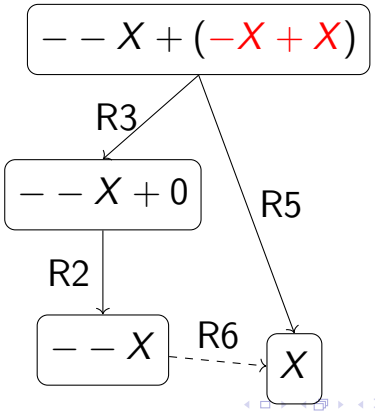
R2: $X + 0 \to X$

R3: $-X + X \to 0$

R4: $(X + Y) + Z \to X + (Y + Z)$

R5: $-X + (X + Z) \to Z$

R6: $- - X \to X$



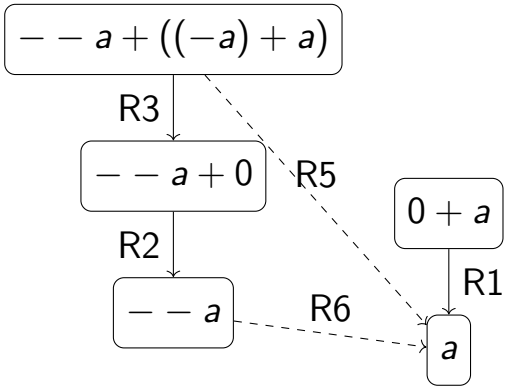
Knuth-Bendix Completion

Prove:

$$--a + ((-a) + a) = 0 + a$$

Rewrite rules:

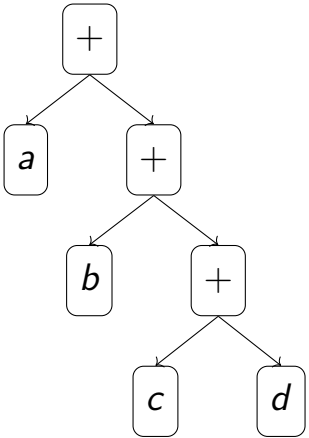
R1: $0 + X \to X$
R2: $X + 0 \to X$
R3: $-X + X \to 0$
R4: $(X + Y) + Z \to X + (Y + Z)$
R5: $-X + (X + Z) \to Z$
R6: $--X \to X$



Hypotheses

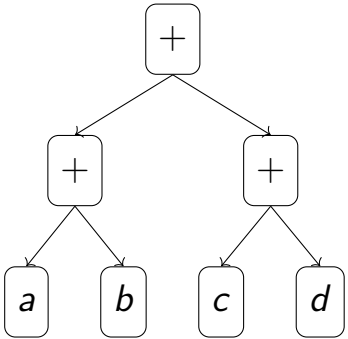
- H1: Extending rule sets by performing Knuth-Bendix Completion improves performance in terms of (a) saturation time and (b) quality of output for current implementations of Equality Saturation.
- H2: Greedy rewriting with KBC-extended rule sets is a viable alternative to Equality Saturation, in terms of output quality, when compile-time resources are limited.

Knuth-Bendix vs. Compiler Canonicalizations



$a + (b + (c + d))$

vs.



$(a + b) + (c + d)$

Knuth-Bendix vs. Compiler Canonicalizations

What to do?

- Keep corresponding rules bidirectional
  - Causes e-graph growth
  - Only works for H1
- Introduce canonicalization step
  - Improves generalizability

Rule Selection

KBC is unlikely to terminate.

What to do?

- Limit rules based on
  - number
  - term size
- Interrupt KBC when no good rules are generated anymore(?)

Rule Selection

KBC is unlikely to terminate.

What to do?

- Limit rules based on
  - number → Bonus: How does the number impact performance?
  - term size
- Interrupt KBC when no good rules are generated anymore(?)

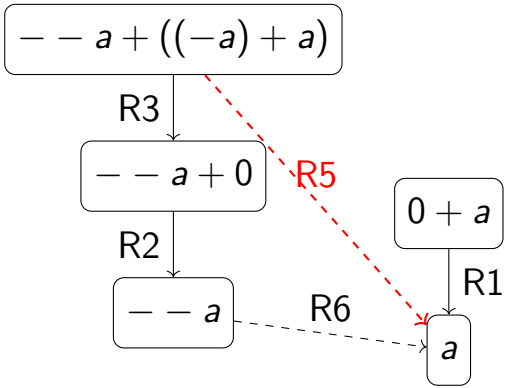
Example: KBC Transformation Revisited

Prove:

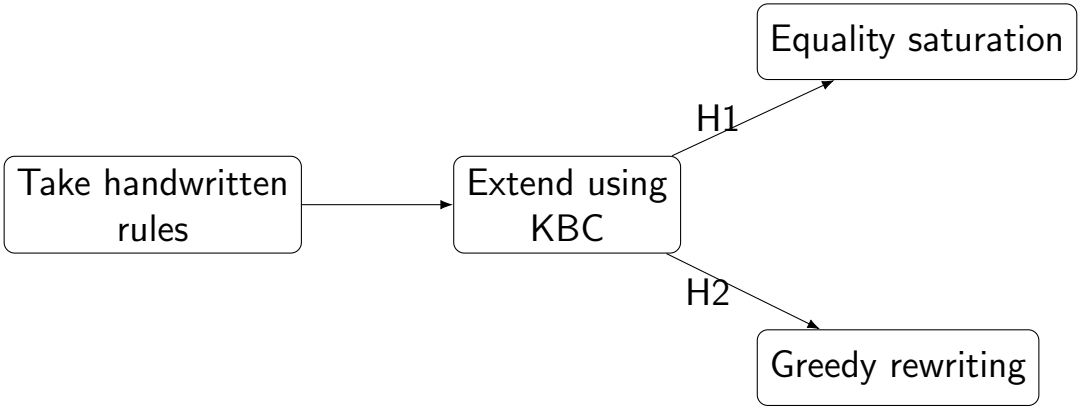
$$--a + ((-a) + a) = 0 + a$$

Rewrite rules:

R1: $0 + X \to X$
R2: $X + 0 \to X$
R3: $-X + X \to 0$
R4: $(X + Y) + Z \to X + (Y + Z)$
R5: $-X + (X + Z) \to Z$
R6: $--X \to X$



Basic Workflow



Tools

- egg (Equality saturation)
  - Easy to use
  - Has example rule sets
  - Used in practice
- Twee (KBC-based theorem prover)
  - Good for generating rule sets
  - Features for KBC termination
  - Simple implementation of Knuth-Bendix Ordering
  - Allows conditional rewrite rules

Validation

- Generate rule sets from egg example rules
- Test on arithmetic terms
- Use egg example rules as benchmark

Validation

- Generate rule sets from egg example rules
- Test on arithmetic terms
- Use egg example rules as benchmark

$\Rightarrow$ Accept H1 if KBC improves output quality and execution time
$\Rightarrow$ Accept H2 if output quality is not significantly worse

Possible Extensions

- Additional domains
  - Boolean algebra
  - Bitvector algebra
- Alternative equality saturation implementations
  - egglog
  - ægraphs
- Finding heuristics for rule selection

Summary

- Extend rewrite rule systems with Knuth-Bendix Completion
- Evaluate with
  - equality saturation
  - greedy rewriting
- Expected contribution:
  - Semi-automated rule set generation for rewrite-based program optimization
  - Insights into the impact of rule set size on equality saturation
  - Enabling cheap optimization through greedy rewriting