Rewrite Rule Inference Using Equality Saturation

Chandrakana Nandi, Max Willsey, Amy Zhu, Yisu Remy Wang, Brett Saiki, Adam Anderson, Adriana Schulz, Dan Grossman, Zachary Tatlock

OOPSLA 2021
Rewrite Rules Are Ubiquitous!

Compilers
Program Synthesizers
Simplifiers / Optimizers
SMT Solvers
ML Frameworks
Rewrite Engines must be Efficient and Reliable!

Compilers
Program Synthesizers
Simplifiers / Optimizers
SMT Solvers
ML Frameworks

Performance and reliability are key for a TRS [Newcomb et al. OOPSLA'20]
But...Designing Rewrite Rules is still Hard!

Who *writes* the *rewrite* rules?
- Typically hand written by experts
- Time consuming, often takes years
- Too few / too many rules
- Unsound rules
A 3-Step Approach for Inferring Rewrite Rules

A 3-Step Approach for Inferring Rewrite Rules

Enumerate terms from a grammar

\[ a, b, 0, +, \ldots \]

\[ + \quad + \quad + \quad + \quad \ldots \]

\[ + \quad + \quad + \quad + \quad + \quad \ldots \]

\[ a \quad b \quad 0 \]
A 3-Step Approach for Inferring Rewrite Rules

Enumerate terms from a grammar

\[ a, b, 0, +, \ldots \]

Find candidates: interpret over concrete inputs

“Fingerprints”

A 3-Step Approach for Inferring Rewrite Rules

Enumerate terms from a grammar:
- a, b, 0, +, …

Find candidates: interpret over concrete inputs:
- “Fingerprints”

Equality:
- \((x + y) \leftrightarrow (y + x)\)

A 3-Step Approach for Inferring Rewrite Rules

Enumerate terms from a grammar

```
a, b, 0, +, ...
```

Find candidates: interpret over concrete inputs

```
(x + 0) ← x
```

A 3-Step Approach for Inferring Rewrite Rules

Enumerate terms from a grammar

a, b, 0, +, ...

Find candidates: interpret over concrete inputs

“Fingerprints”

Joshi et al. 2002, Bansal et al. 2006, Singh et al. 2016, Menendez et al. 2017, ...

\[(x + x) + (x + y) \leftrightarrow (x + x) + (y + x)\]
A 3-Step Approach for Inferring Rewrite Rules

Enumerate terms from a grammar

Find candidates: interpret over concrete inputs

“Fingerprints”

Filter candidates to get final ruleset

Remove redundant rules

A 3-Step Approach for Inferring Rewrite Rules

1. **Enumerate terms from a grammar**
   - Exponentially many terms!

2. **Find candidates: interpret over concrete inputs**
   - Too many candidates, some potentially unsound!

3. **Filter candidates to get final ruleset**
   - Hard to find a small, useful ruleset

---

Equality Saturation for Inferring Rewrite Rules

This Talk:
Inferring \textit{Small, Useful} Rulesets \textit{Faster}
using \textit{Equality Saturation}!
What is *Equality Saturation*?
What is *Equality Saturation*?

\[(a \times 2) / 2\]
What is *Equality Saturation*?

\[(a \times 2) / 2\]
What is *Equality Saturation*?

\[
\frac{(a \times 2)}{2}
\]

\[
\rightarrow a
\]
What is *Equality Saturation*?

\[(a \times 2) / 2 \rightarrow a\]

**Rewrite rules!**

\[(x \times y) / z \leftrightarrow x \times (y / z)\]

\[y / y \rightarrow 1\]

\[x \times 1 \leftrightarrow x\]
How to Apply Rewrite Rules?

\[(a \times 2) / 2 \rightarrow a \times (2 / 2)\]

\[(x \times y) / z \leftrightarrow x \times (y / z)\]
How to Apply Rewrite Rules?

\[
\frac{(a \times 2)}{2} \quad \rightarrow \quad a \times \left(\frac{2}{2}\right)
\]

\[
\frac{(x \times y)}{z} \quad \leftrightarrow \quad x \times \left(\frac{y}{z}\right)
\]

\[
a \times \left(\frac{2}{2}\right) \quad \rightarrow \quad a \times 1
\]

\[
y \div y \quad \rightarrow \quad 1
\]
How to Apply Rewrite Rules?

\[(a \times 2) / 2 \rightarrow a \times (2 / 2)\]

\[(x \times y) / z \leftrightarrow x \times (y / z)\]

\[a \times (2 / 2) \rightarrow a \times 1\]

\[y / y \rightarrow 1\]

\[a \times 1 \rightarrow a\]

\[x \times 1 \leftrightarrow x\]
Destructively, In a Specific Order

Order of rule application affects result

Missed opportunities for optimizations

Same order may not work for all inputs

Old expression is lost

e.g., supporting commutativity is hard without additional tricks to ensure termination!
Equality Saturation Mitigates Phase Ordering!

Initial term \rightarrow E-graph \rightarrow Optimized term

Extract e.g., AST size

Apply all rewrite rules!
How Does Equality Saturation Work?

\[(a \times 2) / 2\]
How Does Equality Saturation Work?

\[(a \times 2) / 2\]
How Does Equality Saturation Work?

\[(a \times 2) / 2\]
How Does Equality Saturation Work?

\[ \frac{a \times 2}{2} \]

E-node

E-class

\[ \frac{x \times y}{z} \leftrightarrow x \times \frac{y}{z} \]
How Does Equality Saturation Work?

\[(a * 2) / 2\]

E-node

E-class

\[(x * y) / z \leftrightarrow x * (y / z)\]

\[(a * 2) / 2, a * (2 / 2)\]
How Does Equality Saturation Work?

(a * 2) / 2

\[
\frac{a \times 2}{2}
\]

E-node

E-class

\[
\frac{(x \times y)}{z} \leftrightarrow x \times \left(\frac{y}{z}\right)
\]

(a * 2) / 2, a * (2 / 2)

Represents both terms!
Equality Saturation for Inferring Rewrite Rules

Equality Saturation for not just applying rewrites, but to also **infer** them!

Ruler

Grammar
\[ e ::= x, 0, e + e, e * e, ... \]

Interpreter
\[
\text{match } e \{
\quad \text{const } \Rightarrow \text{const}
\quad \text{var } (v) \Rightarrow \text{lookup } (v)
\quad e1 + e2 \Rightarrow \text{eval } (e1) + \text{eval } (e2)
\quad e1 * e2 \Rightarrow \text{eval } (e1) * \text{eval } (e2)
\quad ...
\}
\]

Validator
SMT / model check / fuzz

Term Enumeration
Modulo Equivalence
Candidate Rule Generation
Rule Selection

Rewrites
\[
\begin{align*}
    x + 0 &= x \\
    x * 1 &= x \\
    x - 0 &= x \\
    x / 1 &= x \\
    x + y &= y + x \\
    x + (y + z) &= (x + y) + z \\
    x * (y * z) &= (x * y) * z
\end{align*}
\]

Enumeration
Candidate Generation
Rule Selection
Ruler

Grammar

\[ e ::= \text{x, 0, } e + e, e * e, \ldots \]

Interpreter

\[
\text{match } e \{ \\
\quad \text{const } \Rightarrow \text{const} \\
\quad \text{var } (v) \Rightarrow \text{lookup } (v) \\
\quad e_1 + e_2 \Rightarrow \text{eval}(e_1) + \text{eval}(e_2) \\
\quad e_1 * e_2 \Rightarrow \text{eval}(e_1) * \text{eval}(e_2) \\
\quad \ldots \\
\}
\]

Validator

SMT / model check / fuzz

Enumeration

Candidate Generation

Rule Selection

Rewrites

\[
\begin{align*}
\text{x + 0} &= \text{x} \\
\text{x + 1} &= \text{x} \\
\text{x - 0} &= \text{x} \\
\text{x / 1} &= \text{x} \\
\text{x + y + z} &= (\text{x + y}) + \text{z} \\
\text{x * (y * z)} &= (\text{x * y}) * \text{z}
\end{align*}
\]
Enumeration Modulo Equality Saturation

\[ a, b, 0, +, \ldots \]

Exponentially many terms!
Enumeration Modulo Equality Saturation

Exponentially many terms!

Enumerate over an E-graph
Enumeration Modulo Equality Saturation

Exponentially many terms!

Enumerate over an E-graph

E-classes

\[(x + x) + (x + y)\]
\[\equiv\]
\[(x + x) + (y + x)\]

Apply current ruleset
\[(x + y) \leftrightarrow (y + x)\]
Enumeration Modulo Equality Saturation

Exponentially many terms!

E-classes

Merge equivalent terms

Enumerate over an E-graph

Apply current ruleset

\((x + y) \leftrightarrow (y + x)\)
Enumeration Modulo Equality Saturation

Shrinks the term space by applying rewrites as they are learned!
Ruler

Grammar
\[ e ::= x, 0, e + e, e * e, \ldots \]

Interpreter
\[
\text{match } e \{ \\
\quad \text{const } \Rightarrow \text{const} \\
\quad \text{var } (v) \Rightarrow \text{lookup } (v) \\
\quad e1 + e2 \Rightarrow \text{eval } (e1) + \text{eval } (e2) \\
\quad e1 * e2 \Rightarrow \text{eval } (e1) * \text{eval } (e2) \\
\ldots
\}
\]

Validator
SMT / model check / fuzz

Enumeration
Candidate Generation
Rule Selection

Rewrites
\[
\begin{align*}
x + 0 &= x \\
x * 1 &= x \\
x - 0 &= x \\
x / 1 &= x \\
x + y &= y + x \\
x + (y + z) &= (x + y) + z \\
x * (y * z) &= (x * y) * z
\end{align*}
\]
Candidate Generation by Characteristic Vector Matching

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>-2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>-7</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-5</td>
<td>0</td>
</tr>
</tbody>
</table>

Seed initial E-classes with concrete values (cvecs) from the domain
Candidate Generation by Characteristic Vector Matching

Seed initial E-classes with concrete values (cvecs) from the domain

Compute the cvecs for newly enumerated E-classes
Candidate Generation by Characteristic Vector Matching

Compute the cvecs for newly enumerated E-classes

Seed initial E-classes with concrete values (cvecs) from the domain

(x + y) <-> (y + x)
Candidate Generation by Characteristic Vector Matching

Seed initial E-classes with concrete values (cvecs) from the domain:

\[
\begin{array}{cccc}
2 & 4 & 4 & 6 \\
-4 & -3 & -3 & 10 \\
14 & 0 & 0 & -14 \\
8 & -1 & -1 & -10 \\
\end{array}
\]

Compute the cvecs for newly enumerated E-classes:

\[
\begin{array}{cccc}
1 & 3 & 0 & 0 \\
-2 & 5 & 0 & 0 \\
7 & -7 & 0 & 0 \\
4 & -5 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c}
(x + y) \leftrightarrow (y + x) \\
(x + 0) \leftrightarrow x \\
\end{array}
\]
Candidate Generation by Characteristic Vector Matching

Seed initial E-classes with concrete values (cvecs) from the domain

Compute the cvecs for newly enumerated E-classes

Validate candidates using SMT, fuzzing, model checking

\[(x + y) \leftrightarrow (y + x)\]

\[(x + 0) \leftrightarrow x\]
Ruler

Grammar

```
e ::= x, 0, e + e, e * e, ...
```

Interpreter

```
match e {
  | const => const
  | var (v) => lookup (v)
  | e1 + e2 => eval (e1) + eval(e2)
  | e1 * e2 => eval (e1) * eval(e2)
  | ...
}
```

Validator

SMT / model check / fuzz

**Enumeration**

**Candidate Generation**

**Rule Selection**

Rewrites

```
x + 0 = x
x + 1 = x
x - 0 = x
x / 1 = x
x + y = y + x
x + (y + z) = (x + y) + z
x * (y * z) = (x * y) * z
```
Rule Selection with Equality Saturation

\[ C = \]

- \( (x + y) \leftrightarrow (y + x) \)
- \( (x + 0) \leftrightarrow (0 + x) \)
- \( (y + 0) \leftrightarrow (0 + y) \)
- \( (x * y) \leftrightarrow (y * x) \)
- \( (x * 1) \leftrightarrow (1 * x) \)
- \( (y * 1) \leftrightarrow (1 * y) \)
Rule Selection with Equality Saturation

C =

(x + 0) ↔ (0 + x)
(y + 0) ↔ (0 + y)
(x * 1) ↔ (1 * x)
(y * 1) ↔ (1 * y)
Rule Selection with Equality Saturation

Rank sound candidates based on generality and pick top-k (2)

(x + y) ←→ (y + x)
(x * y) ←→ (y * x)
(x + 0) ←→ (0 + x)
(y + 0) ←→ (0 + y)
(x * 1) ←→ (1 * x)
(y * 1) ←→ (1 * y)

Instantiate and add to rule E-graph
Rule Selection with Equality Saturation

Rank sound candidates based on generality and pick top-k (2)

Instantiate and add to rule E-graph
Rule Selection with Equality Saturation

\[(x + y) \leftrightarrow (y + x)\]
\[(x * y) \leftrightarrow (y * x)\]
\[(x + 0) \leftrightarrow (0 + x)\]
\[(y + 0) \leftrightarrow (0 + y)\]
\[(x * 1) \leftrightarrow (1 * x)\]
\[(y * 1) \leftrightarrow (1 * y)\]

Rank sound candidates based on generality and pick top-k (2)

Instantiate and add to rule E-graph
Rule Selection with Equality Saturation

Instatiate and add to rule E-graph

Run equality saturation
Rule Selection with Equality Saturation

All four rules are redundant and therefore discarded!

Instantiate and add to rule E-graph

Run equality saturation
Rule Selection with Equality Saturation

Continue processing until candidate set is empty or has only unsound ones left!

All four rules are redundant and therefore discarded!

Instantiate and add to rule E-graph

Run equality saturation

(x + y) ↔ (y + x)
(x * y) ↔ (y * x)
Rule Selection with Equality Saturation

- Larger top-k makes Ruler faster
- Smaller top-k gives smaller rulesets
- See paper for detailed comparison!

Inequality Saturation

(x + y) \leftrightarrow (y + x) 
(x * y) \leftrightarrow (y * x)

Instantiate and add to rule E-graph

(x + 0) \leftrightarrow (0 + x) 
(y + 0) \leftrightarrow (0 + y) 
(x * 1) \leftrightarrow (1 * x) 
(y * 1) \leftrightarrow (1 * y)
Rule Selection with Equality Saturation

Shrinks the *candidate space* by applying rewrites as they are learned!
Ruler

Grammar

\[ e ::= x, 0, e + e, e * e, \ldots \]

Interpreter

```java
match e {
  | const => const
  | var (v) => lookup (v)
  | e1 + e2 => eval(e1) + eval(e2)
  | e1 * e2 => eval(e1) * eval(e2)
  ...
}
```

Validator

SMT / model check / fuzz

Term Enumeration
Modulo Equivalence

Candidate Rule Generation

Rule Selection

Rewrites

\[
\begin{align*}
x + 0 &= x \\
x * 1 &= x \\
x - 0 &= x \\
x \div 1 &= x \\
x + y &= y + x \\
x + (y + z) &= (x + y) + z \\
x * (y * z) &= (x * y) * z
\end{align*}
\]
Equality Saturation “Soundiness”

Equality Saturation amplifies unsoundness!
Equality Saturation “Soundiness”

Equality Saturation *amplifies* unsoundness!
Equality Saturation “Soundiness”

Equality Saturation *amplifies* unsoundness!

![Diagram](image)

\[(y \times 0) \leftrightarrow 0\]
\[(y \times 0) \leftrightarrow 1\]
Equality Saturation “Soundiness”

Equality Saturation amplifies unsoundness!

current ruleset

Run equality saturation on term E-graph

(y * 0) ←→ 0
(y * 0) ←→ 1

a

0

1
Equality Saturation “Soundiness”

Equality Saturation *amplifies* unsoundness!

Run equality saturation on term E-graph
Equality Saturation “Soundiness”

Equality Saturation *amplifies* unsoundness!

Run equality saturation on term E-graph

Unsound merge, 0 != 1
Implementation

https://github.com/uwplse/ruler

Implemented in Rust

Uses egg for equality saturation
Evaluation

Ruler vs Other tools (CVC4)
How do the rulesets compare?
# Comparison with CVC4

<table>
<thead>
<tr>
<th>Domain</th>
<th># Conn</th>
<th>Time (s)</th>
<th># Rules</th>
<th>Drv</th>
<th>Time (s)</th>
<th># Rules</th>
<th>Drv</th>
<th>Time</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>2</td>
<td>0.01</td>
<td>20</td>
<td>1</td>
<td>0.13</td>
<td>53</td>
<td>1</td>
<td>0.06</td>
<td>0.38</td>
</tr>
<tr>
<td>bool</td>
<td>3</td>
<td>0.06</td>
<td>28</td>
<td>1</td>
<td>0.82</td>
<td>293</td>
<td>1</td>
<td>0.07</td>
<td>0.10</td>
</tr>
<tr>
<td>bv4</td>
<td>2</td>
<td>0.14</td>
<td>49</td>
<td>1</td>
<td>4.47</td>
<td>135</td>
<td>0.98</td>
<td>0.03</td>
<td>0.36</td>
</tr>
<tr>
<td>bv4</td>
<td>3</td>
<td>4.30</td>
<td>272</td>
<td>1</td>
<td>372.26</td>
<td>1978</td>
<td>1</td>
<td>0.01</td>
<td>0.14</td>
</tr>
<tr>
<td>bv32</td>
<td>2</td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
<td>18.53</td>
<td>126</td>
<td>0.93</td>
<td>0.70</td>
<td>0.37</td>
</tr>
<tr>
<td>bv32</td>
<td>3</td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
<td>1199.53</td>
<td>1782</td>
<td>0.91</td>
<td>0.53</td>
<td>0.11</td>
</tr>
</tbody>
</table>
## Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Ruler</th>
<th>CVC4</th>
<th>Ruler / CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Time (s)</td>
<td># Rules</td>
<td>Drv</td>
</tr>
<tr>
<td>bool</td>
<td>0.01</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>bool</td>
<td>0.06</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>0.14</td>
<td>49</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>4.30</td>
<td>272</td>
<td>1</td>
</tr>
<tr>
<td>bv32</td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
</tr>
<tr>
<td>bv32</td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
</tr>
</tbody>
</table>
## Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Ruler</th>
<th>CVC4</th>
<th>Ruler / CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td># Rules</td>
<td>Drv</td>
</tr>
<tr>
<td>bool</td>
<td>0.01</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>bool</td>
<td>0.06</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>0.14</td>
<td>49</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>4.30</td>
<td>272</td>
<td>1</td>
</tr>
<tr>
<td>bv32</td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
</tr>
<tr>
<td>bv32</td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
</tr>
</tbody>
</table>
## Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Ruler</th>
<th>CVC4</th>
<th>Ruler / CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain</td>
<td># Conn</td>
<td>Time (s)</td>
<td># Rules</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>bool</td>
<td>2</td>
<td>0.01</td>
<td>20</td>
</tr>
<tr>
<td>bool</td>
<td>3</td>
<td>0.06</td>
<td>28</td>
</tr>
<tr>
<td>bv4</td>
<td>2</td>
<td>0.14</td>
<td>49</td>
</tr>
<tr>
<td>bv4</td>
<td>3</td>
<td>4.30</td>
<td>272</td>
</tr>
<tr>
<td>bv32</td>
<td>2</td>
<td>13.00</td>
<td>46</td>
</tr>
<tr>
<td>bv32</td>
<td>3</td>
<td>630.09</td>
<td>188</td>
</tr>
</tbody>
</table>
### Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Ruler</th>
<th></th>
<th>CVC4</th>
<th></th>
<th>Ruler / CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td># Rules</td>
<td>Drv</td>
<td>Time (s)</td>
<td># Rules</td>
</tr>
<tr>
<td>bool</td>
<td>0.01</td>
<td>20</td>
<td>1</td>
<td>0.13</td>
<td>53</td>
</tr>
<tr>
<td>bool</td>
<td>0.06</td>
<td>28</td>
<td>1</td>
<td>0.82</td>
<td>293</td>
</tr>
<tr>
<td>bv4</td>
<td>0.14</td>
<td>49</td>
<td>1</td>
<td>4.47</td>
<td>135</td>
</tr>
<tr>
<td>bv4</td>
<td>4.30</td>
<td>272</td>
<td>1</td>
<td>372.26</td>
<td>1978</td>
</tr>
<tr>
<td>bv32</td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
<td>18.53</td>
<td>126</td>
</tr>
<tr>
<td>bv32</td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
<td>1199.53</td>
<td>1782</td>
</tr>
</tbody>
</table>

Fraction of the 1782 rules from CVC4 that the 188 rules from Ruler can derive via equality saturation
Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Ruler</th>
<th>CVC4</th>
<th>Ruler / CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Time (s)</td>
<td># Rules</td>
<td>Drv</td>
</tr>
<tr>
<td>bool</td>
<td>0.01</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>bool</td>
<td>0.06</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>0.14</td>
<td>49</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>4.30</td>
<td>272</td>
<td>1</td>
</tr>
<tr>
<td>bv32</td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
</tr>
<tr>
<td>bv32</td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
</tr>
</tbody>
</table>

Ruler infers a **smaller, useful ruleset faster**
Evaluation

Ruler vs Other tools (CVC4)
How do the rulesets compare?

Ruler vs Humans (Herbie)
Can Ruler compete with experts?
Comparison with Human-written Rules

\[ \sqrt{x+1} - \sqrt{x} \rightarrow \frac{1}{\sqrt{x+1} + \sqrt{x}} \]

Herbie detects inaccurate expressions and finds more accurate replacements. The red expression is inaccurate when \( x > f \); Herbie's replacement, in blue, is accurate for all \( x \).
Comparison with Human-written Rules

52 rational rules, designed by the developers over 6 years

55 / 155 benchmarks are purely over rational arithmetic
Comparison with Human-written Rules

52 **rational** rules, designed by the developers over 6 years

55 / 155 benchmarks are purely over rational arithmetic

Herbie can generate more-complex expressions that aren't more precise #261
Comparison with Human-written Rules

| x * y |
| x | * | y |

| x * x |

52 rational rules, designed by the developers over 6 years

55 / 155 benchmarks are purely over rational arithmetic

Herbie can generate more-complex expressions that aren't more precise #261

Discovered by Ruler, resolved the GitHub issue!
End-to-End: Rational Herbie

None: Remove all rules
Herbie: Herbie without any changes
Ruler: Herbie with Ruler’s rules
Both: Herbie with both original and Ruler’s rules
Rational Herbie: Comparing Accuracy

None: Remove all rules
Herbie: Herbie without any changes
Ruler: Herbie with Ruler’s rules
Both: Herbie with both original and Ruler’s rules

Ruler’s rules are at least as good as the original Herbie rules
Rational Herbie: Comparing AST Size

- **None**: Remove all rules
- **Herbie**: Herbie without any changes
- **Ruler**: Herbie with Ruler’s rules
- **Both**: Herbie with both original and Ruler’s rules

Ruler’s rules are at least as good as the original Herbie rules.
Rational Herbie: Comparing AST Size

None: Remove all rules
Herbie: Herbie without any changes
Ruler: Herbie with Ruler’s rules
Both: Herbie with both original and Ruler’s rules

Ruler’s rules are at least as good as the original Herbie rules

See paper for more results!
Rewrite Rule Inference Using Equality Saturation

Equality Saturation improves all three steps!

**Ruler:** https://github.com/uwplse/ruler