Rewrite Rule Synthesis: A Survey

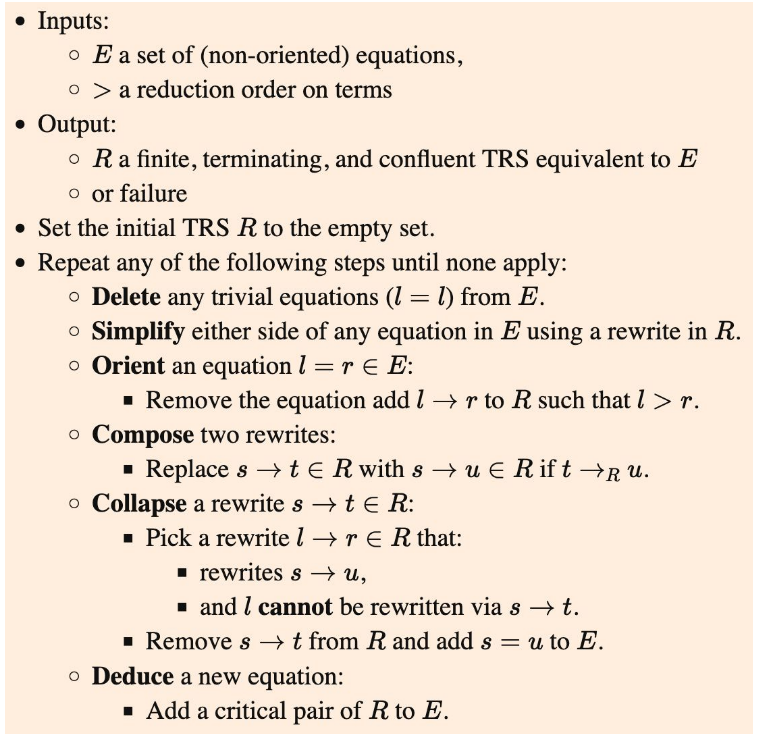
Charles Hong

April 24, 2024

What does it mean to synthesize rewrite rules?

Knuth-Bendix completion:

- Enumerate possible rules based on equations E and rewrite rules R; search order depends on how you apply Orient, Deduce
- Filter things like trivial pairs and subsumption by looping Simplify, Delete, Compose, Collapse
- If it succeeds, produces a verifiably terminating and confluent TRS



- Inputs:
  - E a set of (non-oriented) equations,
  - $>$ a reduction order on terms
- Output:
  - R a finite, terminating, and confluent TRS equivalent to E
  - or failure
- Set the initial TRS R to the empty set.
- Repeat any of the following steps until none apply:
  - **Delete** any trivial equations ($l = l$) from E.
  - **Simplify** either side of any equation in E using a rewrite in R.
  - **Orient** an equation $l = r \in E$:
    - Remove the equation add $l \to r$ to R such that $l > r$.
  - **Compose** two rewrites:
    - Replace $s \to t \in R$ with $s \to u \in R$ if $t \to_R u$.
  - **Collapse** a rewrite $s \to t \in R$:
    - Pick a rewrite $l \to r \in R$ that:
      - rewrites $s \to u$,
      - and l **cannot** be rewritten via $s \to t$.
    - Remove $s \to t$ from R and add $s = u$ to E.
  - **Deduce** a new equation:
    - Add a critical pair of R to E.

The beginnings of a framework for thinking about rewrite rule synthesis

Some typical attributes of rewrite rule synthesis:

1. Order of enumeration
2. Filtering
3. Verification

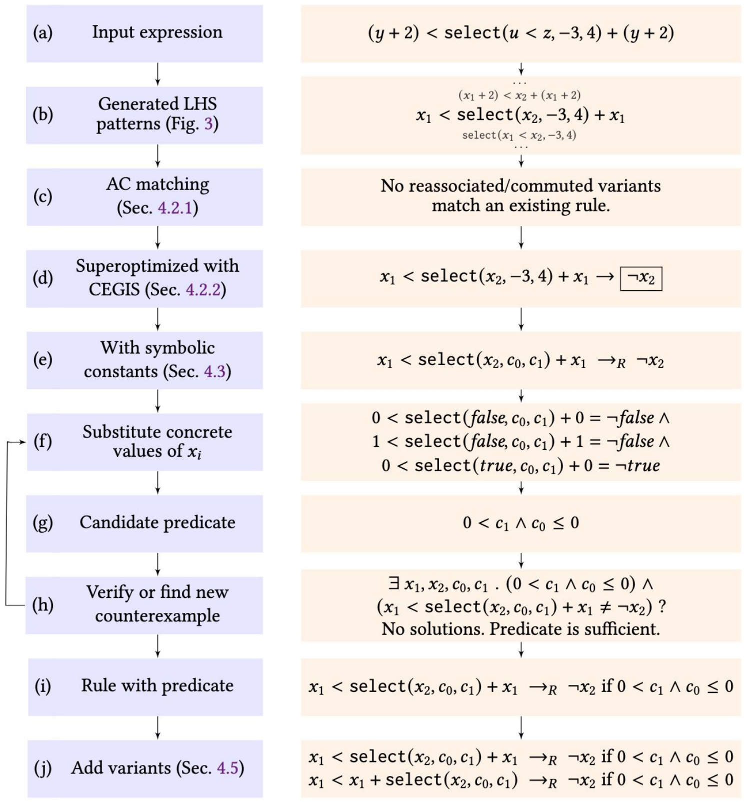
Not necessarily in that order.

What do rewrite rule synthesis tools look like in practice?

Rewrite rule synthesis for program optimization

Halide (OOPSLA 2020)

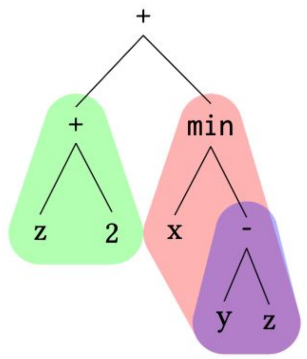
1. Order of enumeration
2. Filtering
3. Verification


(a) Input expression
$(y + 2) < \texttt{select}(u < z, -3, 4) + (y + 2)$
(b) Generated LHS patterns (Fig. 3)
$\cdots$
$(x_1 + 2) < x_2 + (x_1 + 2)$
$x_1 < \texttt{select}(x_2, -3, 4) + x_1$
$\texttt{select}(x_1 < x_2, -3, 4)$
$\cdots$
(c) AC matching (Sec. 4.2.1)
No reassociated/commuted variants match an existing rule.
(d) Superoptimized with CEGIS (Sec. 4.2.2)
$x_1 < \texttt{select}(x_2, -3, 4) + x_1 \rightarrow \boxed{\neg x_2}$
(e) With symbolic constants (Sec. 4.3)
$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \rightarrow_R \neg x_2$
(f) Substitute concrete values of x_i
$0 < \texttt{select}(false, c_0, c_1) + 0 = \neg false \wedge$
$1 < \texttt{select}(false, c_0, c_1) + 1 = \neg false \wedge$
$0 < \texttt{select}(true, c_0, c_1) + 0 = \neg true$
(g) Candidate predicate
$0 < c_1 \wedge c_0 \leq 0$
(h) Verify or find new counterexample
$\exists\, x_1, x_2, c_0, c_1 \;.\; (0 < c_1 \wedge c_0 \leq 0) \wedge$
$(x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \neq \neg x_2)$?
No solutions. Predicate is sufficient.
(i) Rule with predicate
$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \rightarrow_R \neg x_2 \text{ if } 0 < c_1 \wedge c_0 \leq 0$
(j) Add variants (Sec. 4.5)
$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \rightarrow_R \neg x_2 \text{ if } 0 < c_1 \wedge c_0 \leq 0$
$x_1 < x_1 + \texttt{select}(x_2, c_0, c_1) \rightarrow_R \neg x_2 \text{ if } 0 < c_1 \wedge c_0 \leq 0$

Rewrite rule synthesis for program optimization

Halide (OOPSLA 2020)

1. Order of enumeration: LHS searched bottom-up from input expressions, RHS searched with associative/commutative matching and CEGIS



$(z+2)+\texttt{min}(x, y-z)$ $\quad\texttt{min}(x, y-z)$ $\quad z+2$ $\quad y-z$

$v_1+\texttt{min}(x, y-z)$ $\quad\texttt{min}(x, v_2)$

$v_1+\texttt{min}(x, v_3)$

$(z+2)+\texttt{min}(x, v_3)$

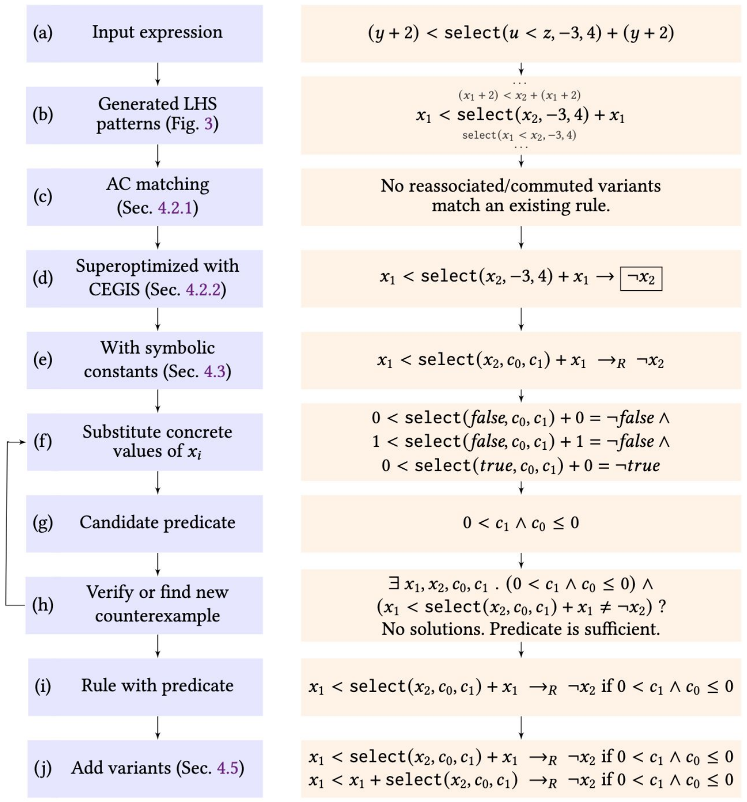
v_1+v_2

Fig. 3. Given the input expression $(z+2)+\texttt{min}(x, y-z)$, we find all possible LHS patterns by substituting fresh variables for subterms, for all valid combinations. Then, we repeat the process for each individual subterm. This process yields the list of candidate LHS terms on the right.

Rewrite rule synthesis for program optimization

Halide (OOPSLA 2020)

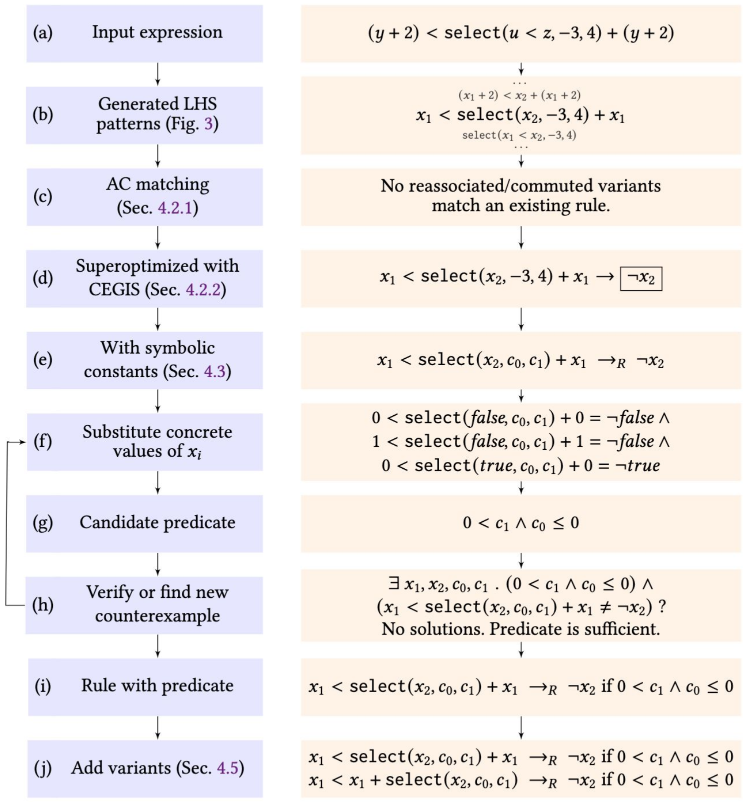
1. Order of enumeration
2. Filtering:
   a. Heuristic pruning based on term size
   b. Generated rules are checked for redundant LHS/predicate combos, as well as if a more general rule already exists


(a) Input expression
$(y+2) < \texttt{select}(u < z, -3, 4) + (y+2)$
(b) Generated LHS patterns (Fig. 3)
…
$(x_1+2) < x_2 + (x_1+2)$
$x_1 < \texttt{select}(x_2, -3, 4) + x_1$
$\texttt{select}(x_1 < x_2, -3, 4)$
…
(c) AC matching (Sec. 4.2.1)
No reassociated/commuted variants match an existing rule.
(d) Superoptimized with CEGIS (Sec. 4.2.2)
$x_1 < \texttt{select}(x_2, -3, 4) + x_1 \to \boxed{\neg x_2}$
(e) With symbolic constants (Sec. 4.3)
$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \to_R \neg x_2$
(f) Substitute concrete values of x_i
$0 < \texttt{select}(\mathit{false}, c_0, c_1) + 0 = \neg \mathit{false} \wedge$
$1 < \texttt{select}(\mathit{false}, c_0, c_1) + 1 = \neg \mathit{false} \wedge$
$0 < \texttt{select}(\mathit{true}, c_0, c_1) + 0 = \neg \mathit{true}$
(g) Candidate predicate
$0 < c_1 \wedge c_0 \leq 0$
(h) Verify or find new counterexample
$\exists\, x_1, x_2, c_0, c_1 \,.\, (0 < c_1 \wedge c_0 \leq 0) \wedge$
$(x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \neq \neg x_2)$?
No solutions. Predicate is sufficient.
(i) Rule with predicate
$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \to_R \neg x_2$ if $0 < c_1 \wedge c_0 \leq 0$
(j) Add variants (Sec. 4.5)
$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \to_R \neg x_2$ if $0 < c_1 \wedge c_0 \leq 0$
$x_1 < x_1 + \texttt{select}(x_2, c_0, c_1) \to_R \neg x_2$ if $0 < c_1 \wedge c_0 \leq 0$

Rewrite rule synthesis for program optimization

Halide (OOPSLA 2020)

1. Order of enumeration
2. Filtering
3. Verification:
   a. An SMT solver and proof assistant are used to prove soundness of rewrites
   b. The authors provide a termination proof based on their reduction order



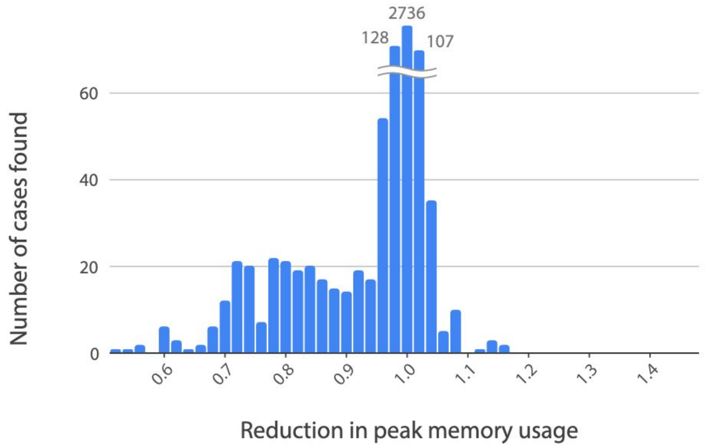
(a) Input expression
$(y+2) < \texttt{select}(u < z, -3, 4) + (y+2)$
(b) Generated LHS patterns (Fig. 3)
$\cdots$
$(x_1+2) < x_2 + (x_1+2)$
$x_1 < \texttt{select}(x_2, -3, 4) + x_1$
$\texttt{select}(x_1 < x_2, -3, 4)$
$\cdots$
(c) AC matching (Sec. 4.2.1)
No reassociated/commuted variants match an existing rule.
(d) Superoptimized with CEGIS (Sec. 4.2.2)
$x_1 < \texttt{select}(x_2, -3, 4) + x_1 \rightarrow \boxed{\neg x_2}$
(e) With symbolic constants (Sec. 4.3)
$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \rightarrow_R \neg x_2$
(f) Substitute concrete values of x_i
$0 < \texttt{select}(\mathit{false}, c_0, c_1) + 0 = \neg \mathit{false} \wedge$
$1 < \texttt{select}(\mathit{false}, c_0, c_1) + 1 = \neg \mathit{false} \wedge$
$0 < \texttt{select}(\mathit{true}, c_0, c_1) + 0 = \neg \mathit{true}$
(g) Candidate predicate
$0 < c_1 \wedge c_0 \leq 0$
(h) Verify or find new counterexample
$\exists\, x_1, x_2, c_0, c_1 \;.\; (0 < c_1 \wedge c_0 \leq 0) \wedge$
$(x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \neq \neg x_2)$?
No solutions. Predicate is sufficient.
(i) Rule with predicate
$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \rightarrow_R \neg x_2$ if $0 < c_1 \wedge c_0 \leq 0$
(j) Add variants (Sec. 4.5)
$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \rightarrow_R \neg x_2$ if $0 < c_1 \wedge c_0 \leq 0$
$x_1 < x_1 + \texttt{select}(x_2, c_0, c_1) \rightarrow_R \neg x_2$ if $0 < c_1 \wedge c_0 \leq 0$

Rewrite rule synthesis for program optimization

Halide (OOPSLA 2020)

- What's the benefit of doing all this?
  - The authors find bugs in the existing TRS
  - They present a case study of 5 PRs where synthesized rules are more general than hand-written ones
  - However, they can't synthesize all the existing rules
  - Also, most of the synthesized rules don't help (although many do)

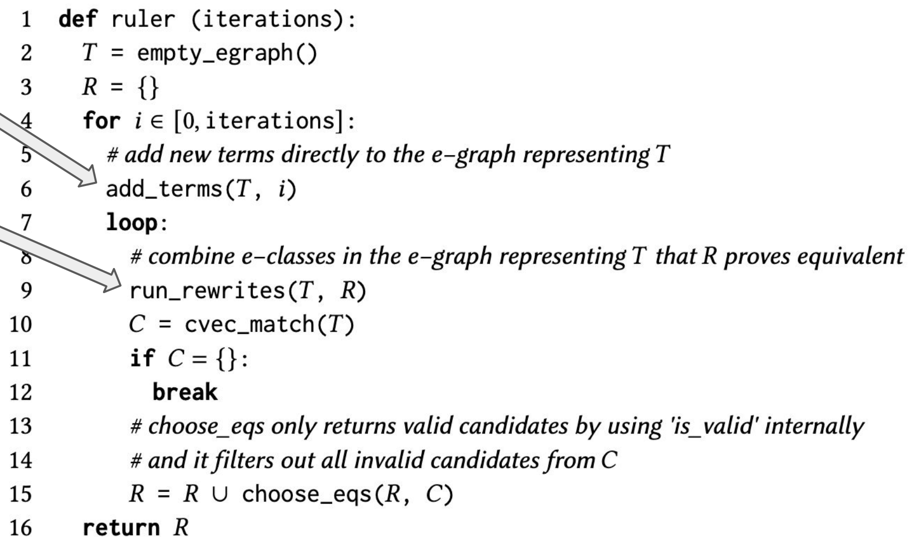
"We have found that human experts can best leverage the strengths of the synthesis tool by using it as an assistant: for example by synthesizing a rule and then generalizing or simplifying it by hand, or by writing a rule and asking the tool to synthesize a valid predicate."



Rewrite rule synthesis for program optimization

Ruler (OOPSLA 2021) applies equality saturation to rewrite rule synthesis

- Given a grammar, enumerate all terms up to size i in an e-graph
- Find e-classes by identifying terms with equivalent fingerprints, and collapse e-graph
- Extract rewrite rules from newly identified e-classes



```
def ruler (iterations):
  $T$ = empty_egraph()
  $R$ = {}
  for $i \in [0,$ iterations]:
    # add new terms directly to the e-graph representing $T$
    add_terms($T$, $i$)
    loop:
      # combine e-classes in the e-graph representing $T$ that $R$ proves equivalent
      run_rewrites($T$, $R$)
      $C$ = cvec_match($T$)
      if $C = \{\}$:
        break
      # choose_eqs only returns valid candidates by using 'is_valid' internally
      # and it filters out all invalid candidates from $C$
      $R$ = $R \cup$ choose_eqs($R$, $C$)
  return $R$
```

Rewrite rule synthesis for program optimization

Ruler (OOPSLA 2021)

1. Order of enumeration
   - "Complete enumeration" of terms in a grammar with up to n connectives
   - Connectives: a has 0, $(a + b)$ has 1, and $(a + (b + c))$ has 2
2. Filtering
   - At each step, keep only a subset of identified rewrite rules that can ideally be used to infer other candidate rules
   - Based on heuristics: more distinct variables, fewer constants, shorter larger side (between the two terms forming the candidate), shorter smaller side, and fewer distinct operators
3. Verification
   - Different backends applied depending on the domain (booleans, bitvector-4, bitvector-32, rationals)

Rewrite rule synthesis for program optimization

Ruler (OOPSLA 2021)

- Able to synthesize a ruleset equivalent to Herbie’s 52 hand-written rules
- Identified a missing rule that led to a Herbie bugfix
- Despite e-graph efficiency, can’t see this scaling to large term or grammar sizes

Rewrite rule synthesis for PL (and PL for rewrite rule synthesis)

Enumo (OOPSLA 2023): Use Ruler's approach, but programmatically guide ruleset construction

1. Order of enumeration
   - Begins with a workload rather than a full grammar
   - "Allowed" and "forbidden" operators specified by the user, forbidden operators can only be added to the e-graph via a set of "exploratory rules"
   - Can use different subsets of rules (allowed, exploratory, all) to grow/collapse e-graph
2. Filtering
3. Verification

Rewrite rule synthesis for program optimization

Isaria (ASPLOS 2024): Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors

- Provides a direct comparison between synthesized ruleset and hand-written ruleset from Diospyros (e-graphs for autovectorization for DSP)

Rewrite rule synthesis for SMT solvers

Syntax-Guided Rewrite Rule Enumeration for SMT Solvers (SAT 2019)

- Simplifying input expressions for ease of verification
- Formal people apply their own techniques (SyGuS) to rewrite rule synthesis

Theory exploration

- Identify useful lemmas for synthesizing proofs
- TheSy (CAV 2021): Theory Exploration Powered By Deductive Synthesis

Why synthesize rewrite rules?

- It depends on the setting
- Could be performance, verifiability, engineering effort

Why not synthesize rewrite rules?

- Need a starting point or a heuristic for rule beneficiality - search won't scale otherwise
- Might be less work to hand-write, if you just want a few basic optimizations

Future Directions

- Priorities are so different in each domain, it’s difficult to suggest general directions for research
- Techniques from across PL-related research have been applied to the problem (equality saturation, CEGIS, SyGuS, SMT solvers)
    - Time to try ML? :)
- Scalability is a big challenge
- Still an early direction!
    - Haven’t seen any sophisticated use of cost models
    - Haven’t seen any TRS or e-graph projects built from the ground up with rewrite rule synthesis