Equality Saturation:
Term Extraction and an Application to Network Synthesis

General Exam: Deyuan (Mike) He
April 16, 2024
1. Brief introduction to equality saturation
2. Term Extraction for equality saturation (Part A)
3. Applying equality saturation for network resource synthesis (Part B)
4. (If time permits) Ongoing project of invariant synthesis for distributed systems
Compiler optimizations are hard to design

- Inlining
- Code motion
- Constant folding
- Vectorize loop
- Load store forwarding
Compiler optimizations are hard to design

<table>
<thead>
<tr>
<th>Compiler Optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant folding</td>
</tr>
<tr>
<td>Vectorize loop</td>
</tr>
<tr>
<td>Inlining</td>
</tr>
<tr>
<td>Code motion</td>
</tr>
<tr>
<td>Load store forwarding</td>
</tr>
</tbody>
</table>

Which order to choose?

Phase Ordering Problem
Compiler optimizations are hard to design

GCC’s passes.def

500+ LoC to define the order

https://github.com/gcc-mirror/gcc/blob/master/gcc/passes.def
Compiler optimizations are hard to design

Observation: program transformations are destructive

\[
\begin{align*}
?V \times 2 & \quad (X \times 2) \div 2 \\
?V \times 2 \rightarrow ?V \ll 1 & \quad ?V \leftrightarrow X \\
& \quad (X \ll 1) \div 2 \\
\end{align*}
\]
Equality Saturation

Non-destructive rewriting

Equality Saturation

Convert to an e-graph

Input Program

Rewrite Rules

Rewrite till saturation / timeout


Equality Saturation

Input Program ➔ → Convert to an e-graph

Rewrite Rules

Rewrite till saturation / timeout

Term Extraction

Focus of our work (Part A)


Equality Saturation and E-Graphs

Converting terms to E-Graphs

\[(X \times 2) \div 2\]

E-Nodes: Function symbols, literals

E-Classes: Equivalent terms

Root E-Class


Equality Saturation and E-Graphs
Program Transformations with Syntactic Rewrites

Non-destructive rewriting

Equality Saturation and E-Graphs
Term Extraction

1. Assign a cost for each E-Node

1. Assign a cost for each E-Node

\[
\begin{align*}
\text{\( \times \)} & \quad \text{\( \div \)} \\
\text{\( \$8 \)} & \quad \text{\( \$10 \)} \\
\text{\( \times \)} & \quad \text{\( \ll \)} \\
\text{\( \$1 \)} & \quad \text{\( \$4 \)} \\
\end{align*}
\]
Equality Saturation and E-Graphs

Term Extraction

1. Assign a cost for each E-Node

2. Pick the min-cost term
   Attempt: Greedy

\[(X \ll 1) \div 2\]

Cost = 10 + 4 + 1 = 15

\[\frac{\div 10}{\times 8} \div 2\]

\[\times 4 \ll 1\]

\[X \times 1\]

\[2 \text{ FREE!}\]

\[1 \text{ FREE!}\]

Is That It?


Term Extraction
When Greedy Fails

Optimal:
11 + 9 + 5 = 25

Greedy:
11 + 8 + 5 + 5 = 29
Previous work: ILP-based extraction

Root Constraint: Extract at least one E-Node from the Root E-Class

Children Constraints: If an E-Node $n$ is extracted, then for all E-Class $C$, if $C$ is a child of $n$, then extract at least one E-Node from $C$

Objective: Minimize the sum of costs of extracted E-Node

Previous work: ILP-based extraction

Variables: $v_x$ for each e-node $x$

Objective: \[ \min \sum_x \text{cost}(x) \cdot v_x \]

Root Constraint: \[ \sum_{x \in \text{Root}} v_x \geq 1 \]

Children Constraints: \[-v_x + \sum_{y \in C_i} v_y \geq 1 \]

for each child $C_i$ of $x$
Previous work: ILP-based extraction

Cycles

$\alpha$ $\beta$

$\alpha$ $\beta$ 0

$\alpha$ $\beta$ 0

$\alpha$ $\beta$ 0

How to avoid infinite expansions?

Previous work: ILP-based extraction
Topological Order Constraints

Variables: $v_x$ for each e-node $x$

Objective:
\[
\min \sum_{x} \text{cost}(x) \cdot v_x
\]

Root Constraint:
\[
\sum_{x \in \text{Root}} v_x \geq 1
\]

Children Constraints:
\[
-v_x + \sum_{y \in C_i} v_y \geq 1
\]
for each child $C_i$ of $x$

Choose One!
Topological Order Constraints

Variables: $v_x, o_x$ for each e-node $x$

Objective: 
\[
\min \sum_x \text{cost}(x) \cdot v_x
\]

Root Constraint: 
\[
\sum_{x \in \text{Root}} v_x \geq 1
\]

Children Constraints: 
\[
-v_x + \sum_{y \in C_i} v_y \geq 1
\]
for each child $C_i$ of $x$

Topological order constraints: 
\[
o_y \geq o_x + 1 \quad (\text{if } v_x = 1), (y \text{ is in some children of } x)
\]

Previous work: ILP-based extraction

Previous work: ILP-based extraction

Topological Order Constraints

Variables: \( v_x, o_x \) for each e-node \( x \)

Objective: 
\[
\min \sum_x \text{cost}(x) \cdot v_x
\]

Root Constraint: 
\[
\sum_{x \in \text{Root}} v_x \geq 1
\]

Children Constraints: 
\[
- v_x + \sum_{y \in C_i} v_y \geq 1
\]

for each child \( C_i \) of \( x \)

Topological order constraints: 
\[
o_y + (1 - v_x) \cdot L \geq o_x + 1 \quad (y \text{ is in some children of } x)
\]

\( L \) is a large enough constant

# Variables: \( O(n) \)  # Constraints: \( O(n) \)  Search Space: \( O(2^n + n^n) \)
Our solution 1: ILP + Acyclicity constraints

Variables: $v_x$ for each e-node $x$

Objective: $\min \sum_x \text{cost}(x) \cdot v_x$

Root Constraint: $\sum_{x \in \text{Root}} v_x \geq 1$

Children Constraints: $-v_x + \sum_{y \in C_i} v_y \geq 1$

for each child $C_i$ of $x$

Acyclicity Constraints: Do not extract any cycle

Works well when number of cycles is reasonable
Acyclicity constraints

\[ (\neg x_1 \land \neg x_2) \land (\neg y_1 \land \neg y_2) \land \neg z_2 \]

Tseitin

\[ O_1 \leftrightarrow (\neg x_1 \land \neg x_2) \]
\[ O_2 \leftrightarrow (\neg y_1 \land \neg y_2) \]
\[ O_1 \lor O_2 \lor \neg z_2 \]

Acyclicity constraints in ILP formulation
Solution 1: ILP + Acyclicity constraints

Variables: $v_x$ for each e-node $x$

Objective: 
$$\min \sum_x \text{cost}(x) \cdot v_x$$

Root Constraint: 
$$\sum_{x \in \text{Root}} v_x \geq 1$$

Children Constraints: 
$$-v_x + \sum_{y \in C_i} v_y \geq 1$$
for each child $C_i$ of $x$

Acyclicity Constraints: 

# Variables: $O(n)$  # Constraints: $O(n \cdot \#\text{cycles})$  Search Space: $O(2^n)$
Solution 2: Weighted Partial MaxSAT

For each E-Node $x$, create a boolean variable $v_x$

$v_x \in \{0, 1\}$ if $x$ is in the extracted term

Must always be satisfied

**Hard Clauses**

- **Root Constraint:** \( \bigvee_{x \in \text{Root}} v_x \)
- **Children Constraints:** \( v_x \rightarrow \bigwedge_{C \in \text{children(x)}} \bigvee_{x' \in C} v_{x'} \)

**Acyclicity Constraints:**

Tseitin \( \bigvee_{C_i} \bigwedge_{x \in C_i \land \text{in_cycle}(x)} v_x \)

**Soft Clauses**

\( \neg v_x \) with weight $\text{cost}(x)$

**Objective:**

Maximizing weight of unextracted E-Nodes

# Variables: \( O(n) \)  
# Constraints: \( O(n \cdot \#\text{cycles}) \)  
Search Space: \( O(2^n) \)
**Term extraction**

**Complexity**

Solution 1 (*ILP-ACyc*): ILP formulation with acyclic constraints

Solution 2 (*WPMAXSAT*): Weighted partial MaxSAT formulation with acyclic constraints

Previous work (*ILP-Topo*): ILP with topological order constraints

<table>
<thead>
<tr>
<th>Encoding</th>
<th># Variables</th>
<th># Constraints</th>
<th>Search Space Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILP-ACyc WPMAXSAT</td>
<td>$O(n)$</td>
<td>$O(nk)$</td>
<td>$O(2^n)$</td>
</tr>
<tr>
<td>ILP-Topo</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(2^n + n^n)$</td>
</tr>
</tbody>
</table>

$n$: number of E-Nodes  
$k$: number of E-Class cycles

Same solution space  
Potentially Exponential
Empirically

Implemented a prototype in the egg [1] framework

Workload: term extraction after equality saturation on tensor programs (DNNs) including


Rewrite rules from Glenside [2]

- **Image-to-column (im2col) only**
- **Image-to-column (im2col) + simplifications (operator fusion, reordering, etc.)**

---


## Term extraction
### Benchmark statistics

<table>
<thead>
<tr>
<th>Unit: 1,000</th>
<th>MobileNetV2</th>
<th>ResMLP</th>
<th>ResNet-18</th>
<th>ResNet-50</th>
<th>EfficientNet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Im2Col</td>
<td>Im2Col+SIMPL</td>
<td>Im2Col</td>
<td>Im2Col+SIMPL</td>
<td>Im2Col</td>
</tr>
<tr>
<td># E-Nodes</td>
<td>50</td>
<td>20</td>
<td>40</td>
<td>8</td>
<td>35</td>
</tr>
<tr>
<td># E-Classes</td>
<td>25</td>
<td>6</td>
<td>20</td>
<td>2.5</td>
<td>25</td>
</tr>
<tr>
<td># Cycles</td>
<td>17</td>
<td>17</td>
<td>15</td>
<td>4</td>
<td>14</td>
</tr>
</tbody>
</table>

Statistics of saturated E-Graphs (Unit: 1k)
Term extraction

Evaluation results

**Upper**: Image-to-column rewrite rule only

**Lower**: Image-to-column + simplifications including Operator fusion, reordering, etc.

- **WPMAXSAT**
- **ILP-ACyc**
- **ILP-Topo**
- **Overhead**

**ILP-Topo timeouts (300s)**

Solving WPMAXSAT and ILP-ACyc is ~3x faster than solving ILP-Topo

For a larger input, solving ILP-Topo (previous work) timeouts after 300s while solving WPMAXSAT and ILP-ACyc takes a few seconds

**Optimality is guaranteed by all encodings**
EGRAPHS’23 Workshop paper

CatsTail: P4 Resource Synthesis using Equality Saturation
Programmable switches

Match-action tables

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_ip</td>
<td>10.0.1.1</td>
<td>DROP</td>
</tr>
<tr>
<td>count</td>
<td>1</td>
<td>INCR(num_pkt)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

10.0.1.1

Mapping to programmable switches is hard

// process packets
if match(p.src)
    p.filtered = 0
    p.ciallo = ID

Abstracts away hardware details
Arbitrary computes
Control flows
Any number of logical stages

Doesn’t Fit!
Rewrite your program

P4 Compiler

Parser

Deparser

Fixed-size tables
Fixed-function ALUs
Fixed number of physical stages
Etc...

Stage 1 Stage 2 Stage 3

Match-action tables

P4 Compiler

Parser

Deparser
Mapping to programmable switches is hard

Challenge 1: Limited # of Stages
Challenge 2: Table Dependencies
Challenge 3: Targeting different backends
Mapping to programmable switches is hard

Challenge 1: Limited # of Stages

\((f_1 + f_2) + f_3) + f_4\)

Stage 1

Stage 2

\[ f_1 \quad S \quad f_2 \quad f_3 \quad R \]
Mapping to programmable switches is hard
Challenge 1: Limited # of Stages

\[(f_1 + f_2) + (f_3 + f_4)\]
Mapping to programmable switches is hard

Challenge 2: Table Dependencies

R/W Dependencies

(this example) Write-after-Read
Read-after-Write
Write-after-Write

Table 1
Reads: f1

Table 2
Writes: f1

Stage X

Stage Y

$X < Y$
Mapping to programmable switches is hard
Challenge 3: Targeting different backends

// process packets
If match(p.src)
  p.filtered = 0
  p.ciallo = ID

P4 Compiler

Intel Tofino
Domino
SmartNIC
Previous work: CaT

Figure 1: The workflow of the CaT compiler.

Previous work: CaT

Our Focus

CaT Compiler

Phase 1: Resource Transformation
Phase 2: Resource Synthesis (Sketch synthesis)
Phase 3: Resource Allocation

P4-16 → P4-16 → Generated Code → Allocation Result → P4-14 Program → Tofino Compiler

Configuration File → Menshen Simulator

Figure 1: The workflow of the CaT compiler.

Resource synthesis via Equality Saturation

Frontend Transformations

- ITE Transformations
  - Path Condition
  - Conditional Assignments
  - Field Assignments

- Symbolic Execution

- Loop-free programs; Ifs, assigns

Equality Saturation

- Mio IR
  - Seq
  - Table 1
  - Table 2
  - Table 3

- Rewrite rules addressing the 3 challenges

Synthesis Extraction

- Min-depth-min-cost extraction
  - Valid program $P$
    - $C(P) < \infty$
  - Invalid program $P$
    - $C(P) = \infty$

- Target-dependent mapping rules
- Domino Stateful Rewrite Rules
- Domino Stateless Rewrite Rules

- Min-depth-min-cost extraction: minimizing stage utilization

41
Frontend transformation

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>hdr.f3 = ite(hdr.f2 == 0, e1, e2)</td>
</tr>
<tr>
<td></td>
<td>hdr.f4 = ite(hdr.f1 == 0, e3, e4)</td>
</tr>
</tbody>
</table>

K

A1

Assign

Ite(f2 == 0, e1, e2)
f3

A2

Assign

Ite(f1 == 0, e3, e4)
f4
Frontend transformation

Introduce Table operators to allow table transformations

T1 must be placed before T2

T1 and T2 are put in the same stage
Rewrite rules

Challenge 1: Limited resource

General-purpose program transformations

\[ ?x + ?y \Rightarrow ?y + ?x \]
\[ (?x + ?y) + ?z \Leftrightarrow ?x + (?y + ?z) \]
\[ ?x + O \Rightarrow ?x \]
\[ \neg(?x \& ?y) \Rightarrow \neg?x \lor \neg?y \]
\[ ?x \& ?x \Rightarrow ?x \]

ite

\[ \text{ite}(\text{true}, \ ?x, \ ?y) \Rightarrow \ ?x \]
\[ \text{ite}(\text{false}, \ ?x, \ ?y) \Rightarrow \ ?y \]

Etc...

52 Rules

Challenge 2: Table Dependencies

Table Transformations

Table parallelization
Subexpression lifting
Table merging
Etc...

10 Rules

Challenge 3: Different backends

Synthesis rewrites

1-1 to sketch grammars in CaT (Gao et al.)

\[ ?x + ?y \Rightarrow \text{alu_add} \ ?x \ ?y \]
\[ \text{if mapped}(?x) \& \text{mapped}(?y) \]

\[ ?V = \text{ite}(?x == ?y, \ ?x + ?z, \ ?x) \Rightarrow \]
\[ \text{stateful_alu}(\text{if}, \ ?V, \ ?x == ?y, \ ?x + z, \ ?x) \]
\[ \text{if} \ ...

Tofino: 11 Rules
Domino: 21 Rules

Table transformations

Goals:
• Explores different topological orders of applying tables
• Parallelizing table placements
• Decomposing computations
• Eliminate table dependencies
Table transformations

Decomposing computations

 hdr.value = \textbf{ite}( hdr.f1 + hdr.f2 + C > hdr.f3, e1, e2)
Table transformations

Decomposing computations

\[
\text{Seq} \\
\text{T1} \rightarrow \text{T2} \\
hdr.tmp = hdr.f1 + hdr.f2 \\
hdr.value = \text{ite}(hdr.tmp + C > hdr.f3, e1, e2)
\]

Lift computes with depth > 3
Can be done if split computation does not involve global variables
Synthesis rewrites

Target-dependent rewrite rules
Based on ALU Grammars used for Sketch-guided synthesis in CaT (Gao et al.)

Stateless ALUs
Pure computations

Stateful ALUs
May modify a register file in the ALU (representing global variables)

SKETCH: a Syntax-guided Synthesis-based technique; Program sketches with holes


Synthesis rewrites
Stateless ALUs

Inductively defined based on Sketch grammars

Base Case: X and Y are literals or PHV field variable

Induction Step: X and Y represent stateless ALU computations
Limitations: a global variable is not read/written by two different tables

With conditions that
1. ?gvar is a global variable or 0
2. ?v1 and ?v2 are PHV fields or constants
Rewrite rules

Efficiently explores the space of candidate mappings by composing the rewrite rules via Equality Saturation

General-purpose program transformations

- ?x + ?y => ?y + ?x
- (?x + ?y) + ?z <=> ?x + (?y + ?z)
- ?x + 0 => ?x
- ~(?x & ?y) => ~?x | ~?y
- ?x & ?x => ?x
- \text{ite}(true, ?x, ?y) => ?x
- \text{ite}(false, ?x, ?y) => ?y

Etc...

Table Transformations

- Table parallelization
- Subexpression lifting
- Table merging
- Etc...

Synthesis rewrites

1-1 to sketch grammars in CaT (Gao et al.)

- ?x + ?y => \text{alu_add} ?x ?y
- if mapped(?x) & mapped(?y)

- ?V = \text{ite}(?x == ?y, ?x + ?z, ?x)
- \text{stateful_alu}(if, ?V, ?x == ?y, ?x + z, ?x)
- if ...

52 Rules

10 Rules

Tofino: 11 Rules  Domino: 21 Rules

Extraction

Goal: Extract min-depth computation tree

**Seq**

\[ \text{cost}(T_1) + \text{cost}(T_2) \]

**Par**

\[ \max \left( \text{cost}(T_1), \text{cost}(T_2) \right) \]
Extraction
Goal: Extract min-depth computation tree

max \(_i\) (Cost(A\(_i\)))
Extraction

Goal: Extract min-depth computation tree

\[
\text{ite} \left( \text{mapped}(f(X,Y)), \max \left( \text{Cost}(X), \text{Cost}(Y) \right) + 1, \infty \right)
\]

Only allow extracting computations that are already mapped to target backends.
$C(P) = \text{Minimum number of stages required to map } P$
Evaluations

RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)

RQ2: Efficacy of CatsTail: stage utilization compared with CaT

RQ3: Does the extraction always succeed?
RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)

**Experiments setup:**

**Target Backends:** Intel Tofino and Domino (Banzai) ALUs

**Input programs:** 8 P4 programs with real-word applications, including:

- Rate control protocol, Packet sampling, Flowlet Switching,
- Stateful firewall, Blue increase/decrease, Marple flow

**Rewrite Rules:**

For the Tofino backend, we enable all the synthesis rewrite

For the Domino backend, we ran two sets of experiments:

1. Full: All synthesis rewrite rules
2. Sk: synthesis rewrite rules corresponding to the sketch grammar

CaT used in their benchmark
RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)

Evaluations

(b) Synthesis time comparison for Intel Tofino ALUs

- X: Benchmark cases.
- Y: Synthesis time (ms), in log-scale

Successfully synthesized

~an order of magnitude faster in synthesis
Evaluations

RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)

Successfully synthesized
Orders of magnitude faster
Evaluations

RQ2: Efficacy of CatsTail: stage utilization compared with CaT

Table 1. Comparison of the number of stages required to map the synthesized program given by CATSTAIL and CaT [Gao et al. 2023] to Intel Tofino switches and Domino switches.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># Stages on Domino</th>
<th># Stages on Tofino</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CATSTAIL</td>
<td>CaT</td>
</tr>
<tr>
<td>RCP</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Sampling</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Blue Increase</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Flowlet Switching</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Marple Flow NMO</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Marple New Flow</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Stateful Firewall</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Learn Filter</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Same numbers of stage required**

Nested ifs not supported by Tofino switch
Evaluations

RQ3: Does the extraction always succeed?

// process packets
If some_func(p.src)
  p.ciallo = 1
else
  P.drop = 1

CatsTail

$C(P) = \infty$

Incompleteness of general purpose / table transformation rules
// process packets
if some_func(p.src)
    p.ciallo = 1
else
    P.drop = 1

\[
C(\mathcal{P}) = \infty
\]

Evaluations

RQ3: Does the extraction always succeed?
Evaluations

RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)

Orders of magnitude faster compared with CaT, thanks to the scalability of egg

RQ2: Efficacy of CatsTail: stage utilization compared with CaT

Stage utilization is as good as CaT

RQ3: Does the extraction always succeed?

No, but we can work around
Outline

1. Brief introduction to equality saturation
2. Term Extraction for equality saturation **(Part A)**
3. Applying equality saturation for network resource synthesis **(Part B)**
4. (If time permits) Ongoing project of invariant synthesis for distributed systems
Recent project: PInfer
Learning invariants for distributed systems from traces

Recent project: PInfer
Learning invariants for distributed systems from traces

Recent project: PInfer
Learning invariants for distributed systems from traces

Invariant learning: Related works

Protocol Definition + Invariants checking

Enumerate combinations of predicates and connectives

Ivy
DistAI
DuoAI
SWISS
PInfer

Invariants about messages / events

Invariants about states (Ivy-style)


Invariant learning

PInfer

Challenges:

1. Huge search space: many valid predicates over events and payloads
   Brute-force enumeration leads to vacuously true/false invariants, which are not useful for production systems

2. Efficiency: enumerating logical connectives is computationally intractable

Trace Grammar that focuses useful predicates

Formulate invariant learning as a boolean function learning problem