Equality Saturation for Tensor Graph Superoptimization

Yichen Yang, Pitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, and Jacques Pienaar (2021)

Presented by Sarah Zhao, R244 (11.16.2022)

Motivation

- Tensor graph super-optimization via **graph substitutions** (aka **rewrite rules**)
- **TASO**: generation of graph substitutions
  - Searching space of resulting graphs
    - Greedy algorithm and sequential application of substitutions
    - Limits search space and not necessarily optimal
    - Inefficient (exponential scaling)

How to improve?

Tensat!

- Builds on TASO graph substitutions and improves search algorithm
- **Equality Saturation** [Tate et al., 2009]
  - Idea: explore everything compactly first, then extract optimal solution
  - Uses a data structure called **e-graphs** for compact storage [Gregory Nelson PhD Thesis, 1980]
  - 2 main phases: **exploration** and **extraction**
- Faster run-time and shorter search time compared to TASO

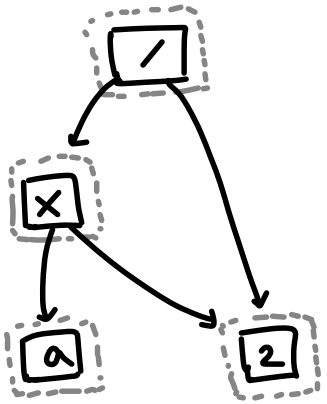
Equality Saturation

All about equivalences

e-graphs: set of equivalence classes (e-classes) each of which contain e-nodes

e-nodes: an operator with children that are **e-classes**

Input: $(a \times 2)/2$



Example - Exploration

Input: $(a \times 2)/2 = (a \ll 1)/2$

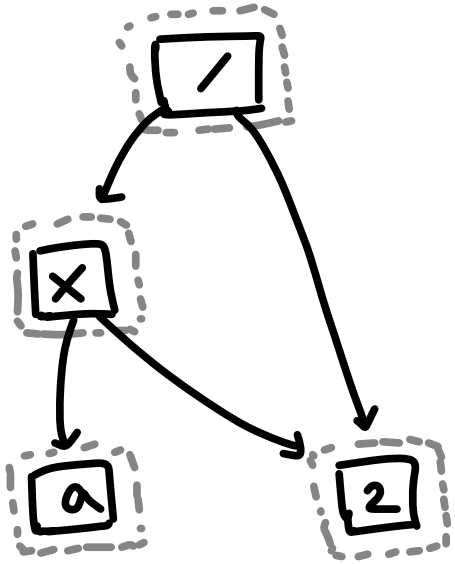
Substitutions:

$x \times 2 \rightarrow x \ll 1$

$(x \times y)/z \rightarrow x \times (y/z)$

$x/x \rightarrow 1$

$x \times 1 \rightarrow x$



Initial e-graph

Example - Exploration

Input: $(a \times 2)/2 = (a \ll 1)/2$

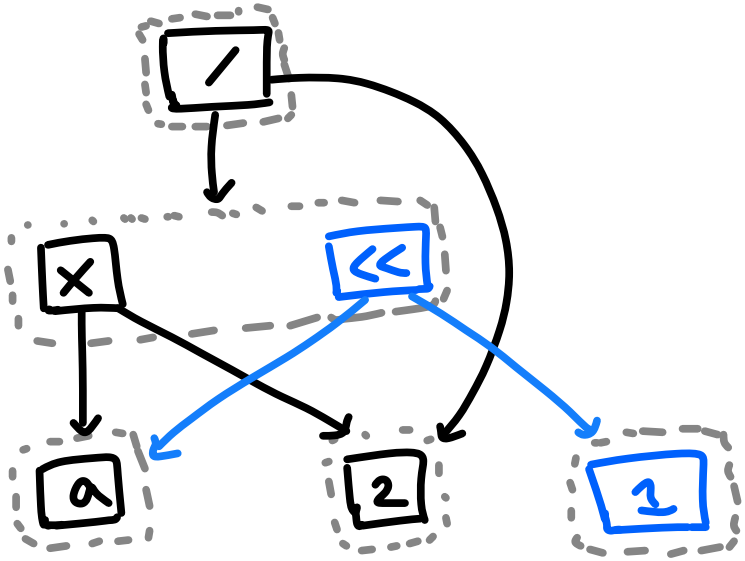
Substitutions:

$x \times 2 \rightarrow x \ll 1$

$(x \times y)/z \rightarrow x \times (y/z)$

$x/x \rightarrow 1$

$x \times 1 \rightarrow x$



Apply first rewrite rule

Example - Exploration

Input: $(a \times 2)/2 = a \times (2/2) = a \times 1 = a$

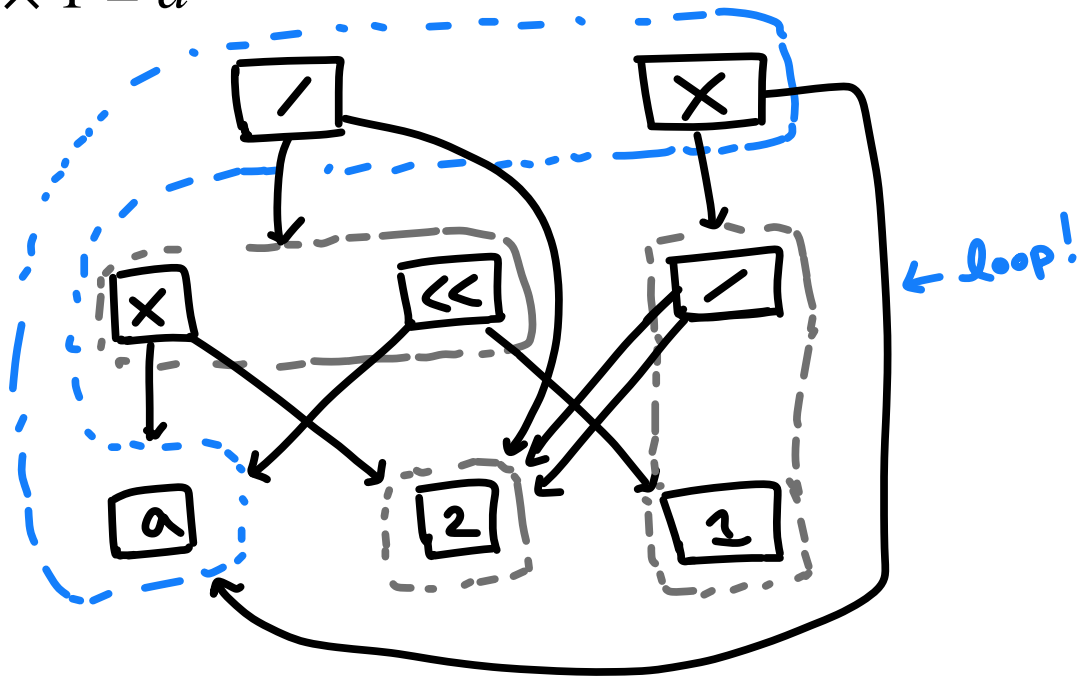
Substitutions:

$x \times 2 \rightarrow x \ll 1$

$(x \times y)/z \rightarrow x \times (y/z)$

$x/x \rightarrow 1$

$x \times 1 \rightarrow x$



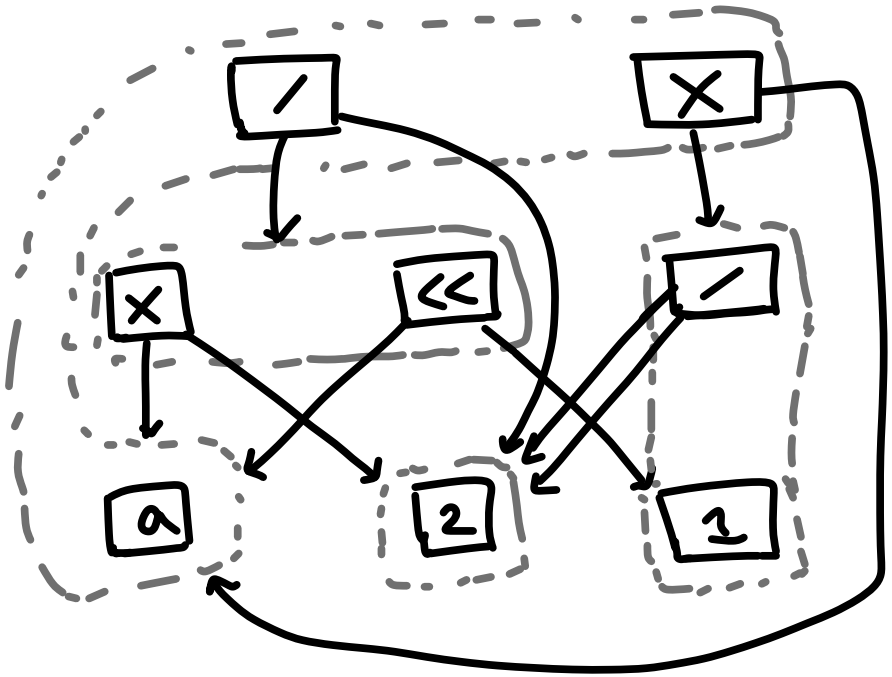
Applied all four rewrite rules

Explore until saturated or user-specified number of iterations

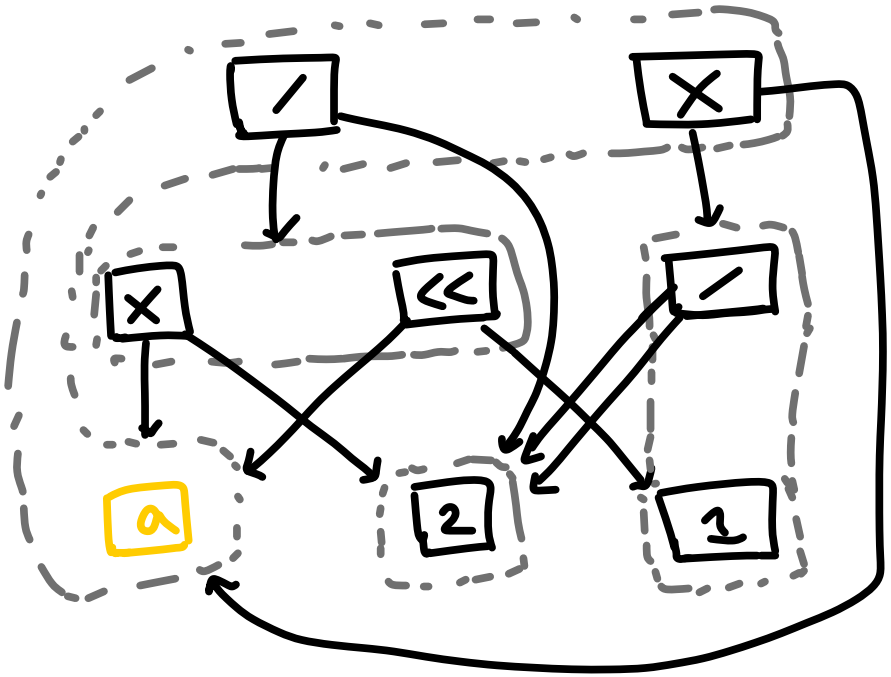
Equality Saturation

2 phases to find optimal equivalent expression (e.g. computation graph)

Exploration

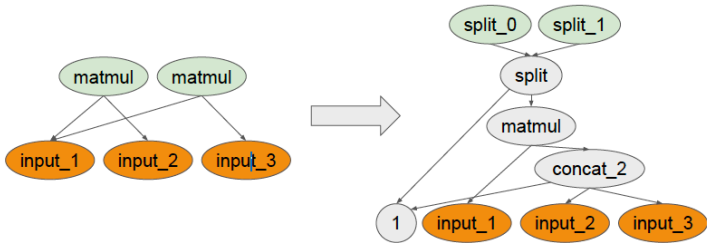


Extraction

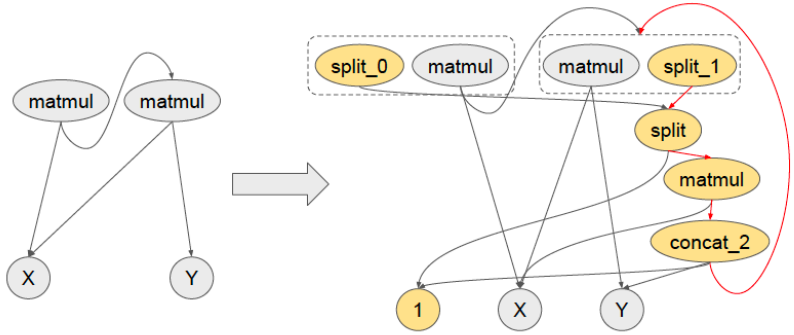


Repurpose for tensor graph superoptimization

- Extensions to standard equality saturation
  - Multiple output subgraphs [exploration] - note: makes e-graph large!



  - No cycles for tensor computation graphs



Tensat: Extraction

- Cost model:
  - Each operation (1-1 with e-node) associated with a cost
  - Total cost = sum of costs of all operations in resulting graph
- Greedy extraction vs **ILP extraction**
- ILP struggles with cycle constraints

Minimize: $f(x) = \sum_i c_i x_i$

Subject to:

$$x_i \in \{0, 1\}, \quad (1)$$

$$\sum_{i \in e_0} x_i = 1, \quad (2)$$

$$\forall i, \forall m \in h_i, x_i \leq \sum_{j \in e_m} x_j, \quad (3)$$

$$\forall i, \forall m \in h_i, t_{g(i)} - t_m - \epsilon + A(1 - x_i) \geq 0, \quad (4)$$

$$\forall m, 0 \leq t_m \leq 1, \quad (5)$$

Tensat: Cycle Filtering

Revisiting the exploration step

- Idea: filter for cycles in exploration phase
- Naive approach checking for each substitution
- Efficient implementation: for each iteration
  - **Pre-filtering**: create a list of descendant e-classes for each node
  - For each substitution, check with descendants list to see whether it causes cycle
  - **Post-processing**: find all cycles created in iteration via DFS, remove last node that causes the cycle

Experiments

Comparison with TASO

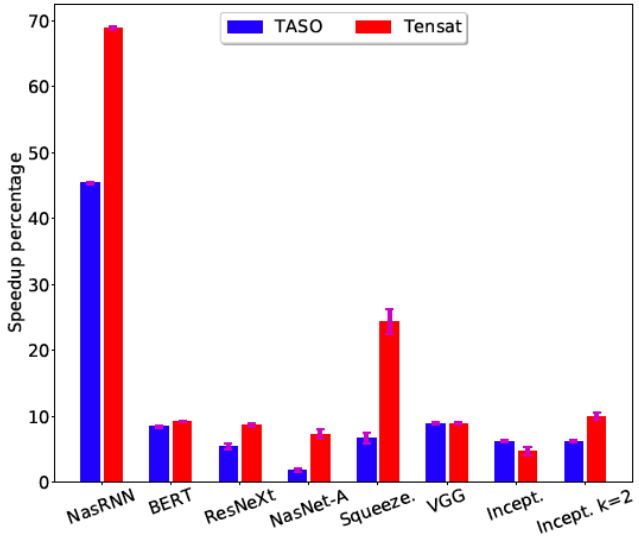


Figure 4. Speedup percentage of the optimized graph with respect to the original graph: TASO v.s. TENSAT. Each setting (optimizer × benchmark) is run for five times, and we plot the mean and standard error for the measurements.

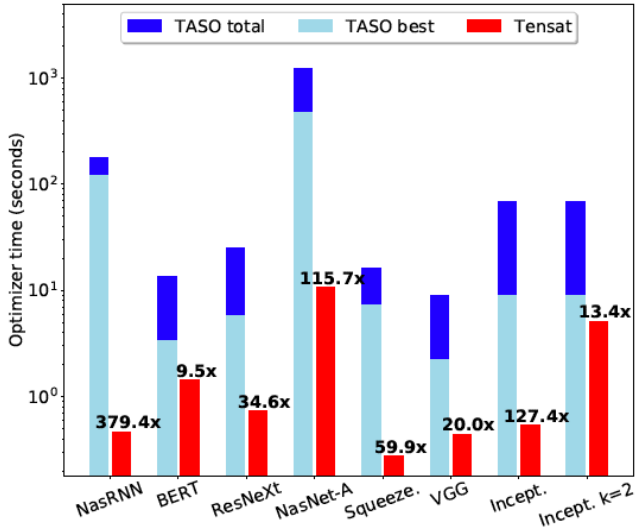


Figure 5. Optimization time (log scale): TASO v.s. TENSAT. "TASO total" is the total time of TASO search. "TASO best" indicates when TASO found its best result; achieving this time would require an oracle telling it when to stop.

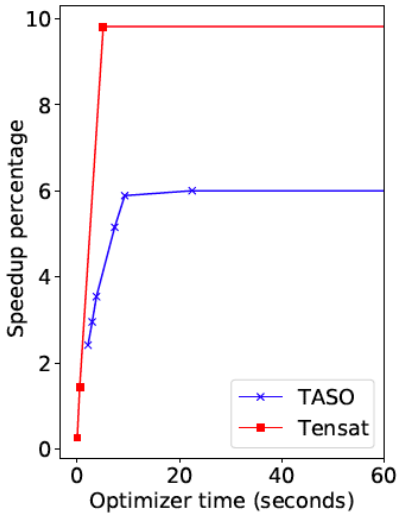


Figure 6. Speedup over optimization time for TASO and TENSAT, on Inception-v3. We use a timeout of 60 seconds.

Experiments

Multiple output subgraphs

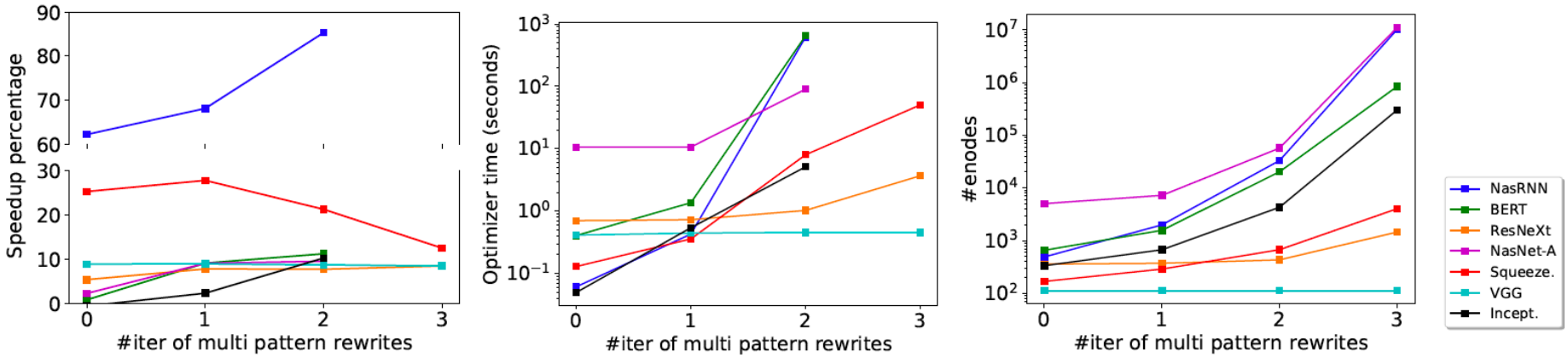


Figure 7. Effect of varying the number of iterations of multi-pattern rewrites k_{multi}. For BERT, NasNet-A, NasRNN, Inception-v3, the ILP solver times out at one hour for $k_{\text{multi}} = 3$. Left: speedup of the optimized graphs (the y-axis is split for clarity). Middle: time taken by TENSAT. Right: final e-graph size (number of e-nodes). The middle and right figures are in log scale.

In summary

Tensat

- Framework for tensor graph super-optimization via graph substitutions
  - search algorithm using **equality saturation**
  - extensions for tensor computation graphs:
    - multiple output subgraphs
    - cycle filtering
- run-time and optimization time speed ups compared to TASO

Discussion + Questions

- Changing cost model to incorporate other hardware configurations (e.g. parallel execution of operations)

- Is the solution a **global** optimum?
  - cycle filtering
  - user decides number of multiple output subgraphs
  - [added after Q&A] how big does the e-graph really grow? Implementation limits number of nodes. Hard to grow until saturation
  - [added after Q&A] local cost model (they currently use operation run-time vs global one)
- Reasons for extraction algorithms explored? Any others?
- Heat-map for which graph substitutions were made for each architecture (like in TASO paper) would be interesting

Thank you!

Questions?

References

Yichen Yang, Phitchaya Mango Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, and Jacques Pienaar. 2021. Equality Saturation for Tensor Graph Superoptimization. In Proceedings of the 4th MLSys Conference, San Jose, CA, USA (2021). https://doi.org/10.48550/arXiv.2101.01332

Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19). Association for Computing Machinery, New York, NY, USA, 47–62. https://doi.org/10.1145/3341301.3359630

https://egraphs-good.github.io/

https://slideslive.com/38952736/oral-equality-saturation-for-tensor-graph-superoptimization?ref=recommended

More experiments

Extraction time (s)	k_{multi}	With cycle real	With cycle int	Without cycle
BERT	1	0.96	0.98	**0.16**
	2	>3600	>3600	**510.3**
NasRNN	1	1116	1137	**0.32**
	2	>3600	>3600	**356.7**
NasNet-A	1	424	438	**1.81**
	2	>3600	>3600	**75.1**

Table 5. Effect of whether or not to include cycle constraints in ILP on extraction time (in seconds), on BERT, NasRNN, and NasNet-A. For the cycle constraints, we compare both using real variables and using integer variables for the topological order variables t_m.

Exploration time (s)	k_{multi}	Vanilla	Efficient
BERT	1	0.18	**0.17**
	2	32.9	**0.89**
NasRNN	1	1.30	**0.08**
	2	2932	**1.47**
NasNet-A	1	3.76	**1.27**
	2	>3600	**8.62**

Table 6. Comparison between vanilla cycle filtering and efficient cycle filtering, on the exploration phase time (in seconds) for BERT, NasRNN, and NasNet-A.

More experiments

Graph Runtime (ms)	Original	Greedy	ILP
BERT	1.88	1.88	**1.73**
NasRNN	1.85	1.15	**1.10**
NasNet-A	17.8	22.5	**16.6**

Table 4. Comparison between greedy extraction and ILP extraction, on BERT, NasRNN, and NasNet-A. This table shows the runtime of the original graphs and the optimized graphs by greedy extraction and ILP extraction. The exploration phase is run with $k_{\text{multi}} = 1$.

Example

Input: $(a \times 2)/2$

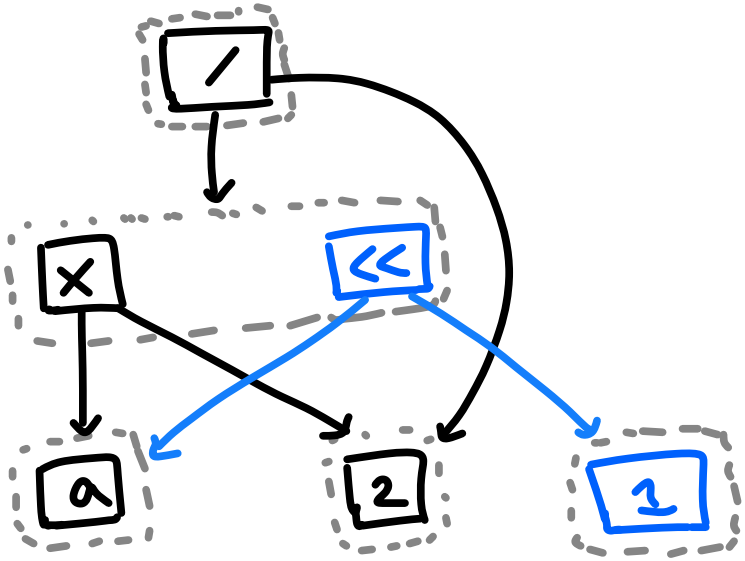
Rewrite rules:

$x \times 2 \rightarrow x \ll 1$

$(x \times y)/z \rightarrow x \times (y/z)$

$x/x \rightarrow 1$

$x \times 1 \rightarrow x$



Apply first rewrite rule

Example

Input: $(a \times 2)/2$

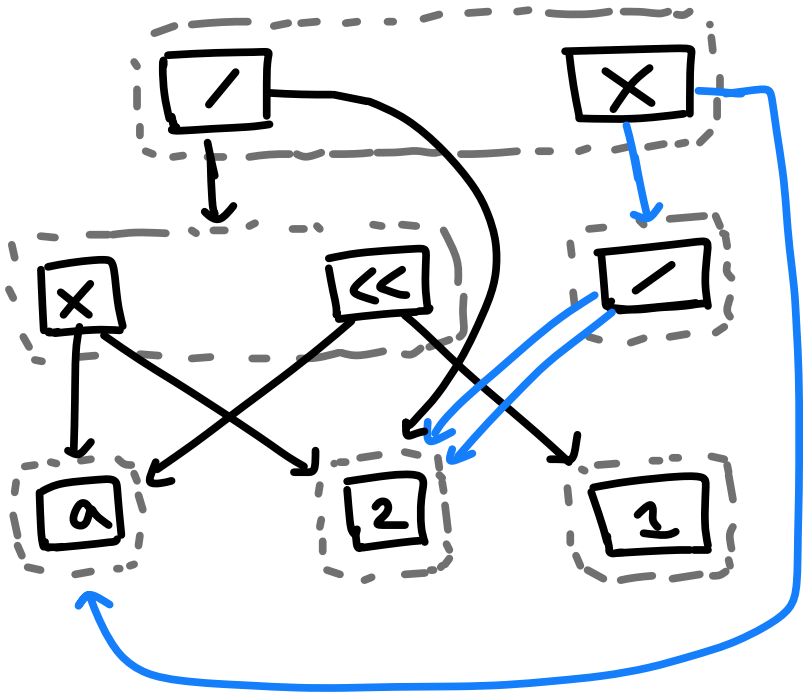
Rewrite rules:

$x \times 2 \rightarrow x \ll 1$

$(x \times y)/z \rightarrow x \times (y/z)$

$x/x \rightarrow 1$

$x \times 1 \rightarrow x$



Apply second rewrite rule

Example

Input: $(a \times 2)/2$

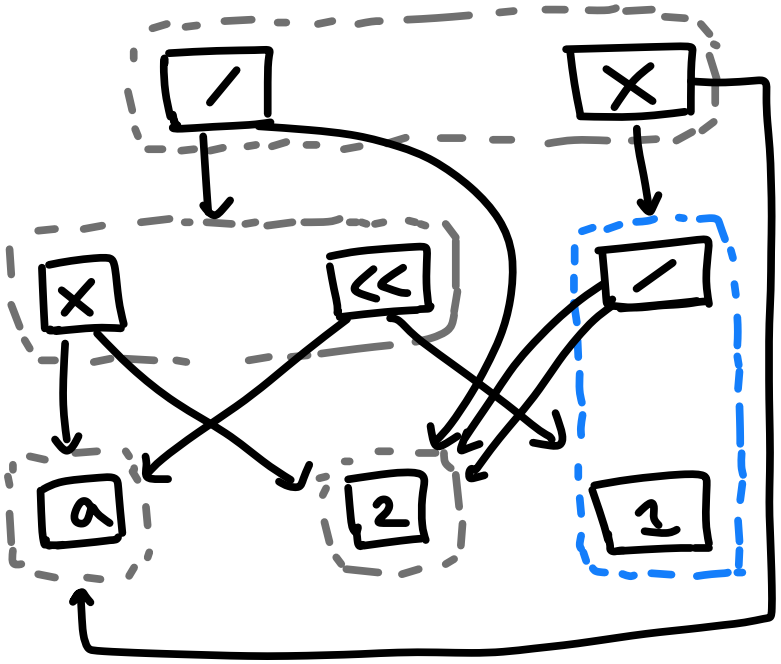
Rewrite rules:

$x \times 2 \rightarrow x \ll 1$

$(x \times y)/z \rightarrow x \times (y/z)$

$x/x \rightarrow 1$

$x \times 1 \rightarrow x$



Apply third rewrite rule

Exampl

Input: $(a \times 2)/2 = a \times (2/2) = a \times 1 = a$

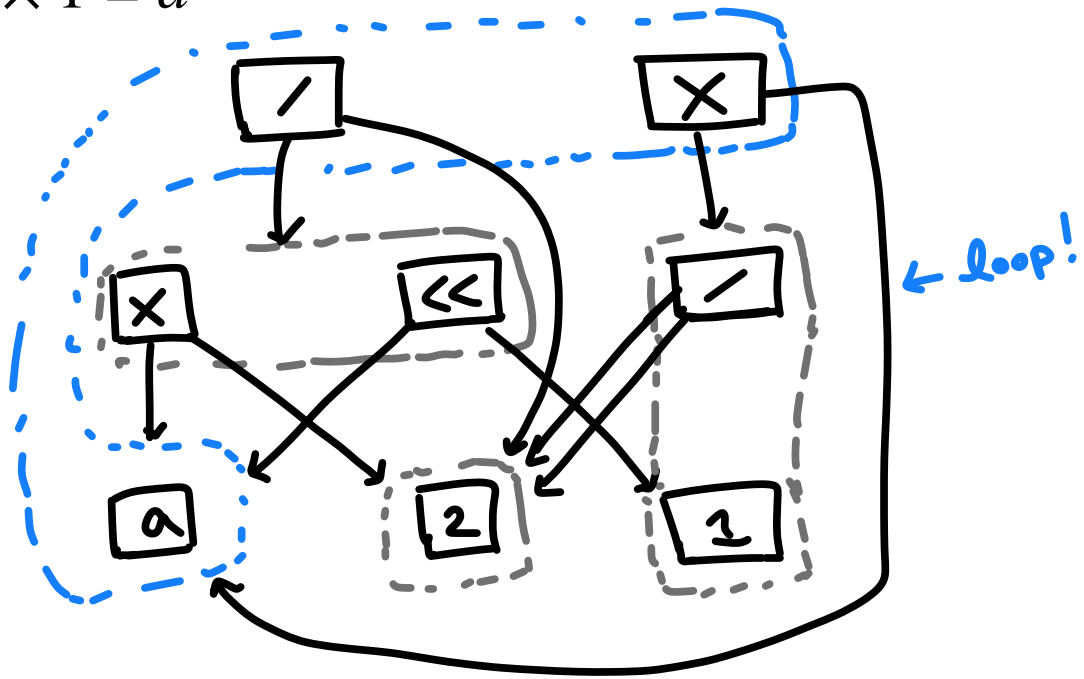
Substitutions:

$x \times 2 \rightarrow x \ll 1$

$(x \times y)/z \rightarrow x \times (y/z)$

$x/x \rightarrow 1$

$x \times 1 \rightarrow x$



Applied all four rewrite rules

Explore until saturated or user-specified number of iterations

Egg implementation

2 core operations (as implemented on egg):

add to add e-nodes and **union** to merge e-classes

Defined so that it keeps e-nodes **unique** (no two e-nodes with the same operators and equivalent children in either the same or different e-classes)