

Recovering TASO Substitutions Under Bounded Rewrite Closure

Pietro Marsella

Bodo Wirth

marsella@stanford.edu

bodow@stanford.edu

Abstract

Machine learning compilers have emerged in recent years as a major tool for automatically optimizing the performance of deep learning models. A major component of this optimization pipeline is **algebraic rewriting**, where a compiler repeatedly applies algebraic graph substitutions to transform a model into a semantically equivalent but more efficient form.

In several modern compilers, these substitutions are handwritten. Recent works such as TASO, however, have shown that substitutions can instead be discovered automatically by enumerating candidate graph pairs and then formally verifying the potentially valid ones using a smaller set of base operator axioms. In this paper, we study the following question: are these substitutions, given a powerful enough rewrite system, derivable as explicit compositions of the base TASO axioms? If so, the compiler pipeline could be simplified significantly by removing the need for a separate substitution discovery stage and instead relying on an expressive axiom set.

To answer this question, we design a powerful rewrite system over a symbolic graph IR and use it together with bounded search to find explicit axiomatic derivations for the substitutions discovered by TASO. Evaluated on 444 substitutions extracted from the TASO implementation, our system recovers over 97% of them, with most derivations requiring only a small number of axiom applications. These results suggest that given a powerful enough rewrite system, most substitutions discovered by TASO are derivable from the base axioms.

The implementation is available at <https://github.com/Marsella8/taso-sparse>.

1 Introduction

Machine learning compilers have become a central part of the deep learning software stack. One of the initial steps of compilation, generally done on a high-level graph IR, is algebraic rewriting: from a set of predetermined substitutions, the compiler matches a substitution to a model and applies it, which effectively means taking a subgraph and replacing it with a semantically equivalent one. This yields models that are semantically equivalent but better according to some cost such as runtime, memory traffic, or latency.

A traditional way to build such substitution sets is to manually design them. This requires substantial manual effort, scales poorly as the operator set grows, and increases the likelihood of bugs as each substitution is written independently. TASO addressed part of this problem by showing that useful substitutions can instead be discovered automatically and then formally verified from a smaller set of base operator axioms, before being used in optimization [1]. This naturally leads to a further question: once TASO has discovered such substitutions, how many of them are already obtainable by repeatedly composing the original axioms inside an explicit rewrite system?

This paper studies that question. Rather than using SMT-based validation to prove that candidate substitutions are correct, we ask whether the substitutions discovered by TASO can be reconstructed as concrete derivations from TASO’s base axioms. If most of TASO’s substitutions are recoverable in this sense, then a large part of the optimizer’s substitution knowledge can be understood as derived from a compact trusted core, rather than as a long list of independent primitive substitutions. This simplifies the compiler pipeline significantly, as it suggests that expanding the axiom set is an alternative and effective way of increasing the expressiveness of the compiler.

To investigate this, we implement a powerful graph rewrite engine over a symbolic graph intermediate representation. Our system supports substitution matching and application on arbitrary subgraphs, structural canonicalization for efficient deduplication, and bounded search for explicit derivations with derivation-trace reconstruction. We evaluate it on 444 non-isomorphic substitutions extracted from the TASO implementation, and under our search configuration recover 432 of them. The recovered derivations are typically short, with median length 2 and maximum length 6. These results provide direct evidence that the large majority of TASO’s discovered substitutions are already derivable from a comparatively small axiom basis.

2 Motivation and Related Work

TASO is the most direct starting point for our work. As stated in the introduction, it discovers substitutions automatically from a smaller semantic basis [1]. It begins from a fixed set of supported operators together with a collection of base axioms describing their equivalences. It then enumerates

candidate graphs up to a chosen size bound, groups together graphs that appear equivalent under randomized testing, and uses formal verification to prove which pairs in the same candidate class are actually sound substitutions. The result is a collection of verified graph substitutions that can then be handed to an optimizer. TASO thus factors the optimization pipeline into two stages: an offline discovery-and-verification stage that produces substitutions, and a later substitution-application stage that applies them to optimize a given model.

Subsequent work has mostly focused on the second of these stages, namely how to search more effectively through the space induced by a set of known substitutions. MetaFlow observes that restricting attention to immediately profitable substitutions can be too limiting, and therefore allows relaxed substitutions that may take locally worse intermediate steps in order to reach better final graphs [2]. TENSAT instead uses equality saturation to represent many equivalent graphs simultaneously inside an e-graph, thereby reducing the phase-ordering problems of sequential rewriting [4, 5]. Unity extends this viewpoint to distributed training, where both algebraic substitutions and parallelization choices are represented in a unified optimization space over parallel computation graphs [3]. These systems differ in representation and search strategy, but they share the same general perspective: the substitution set is taken as given, and the main question is how best to utilize it.

Our motivation is to look one level earlier. If the substitutions discovered by TASO are already recoverable directly as compositions of the base axioms, then the compiler pipeline can in principle be simplified. Rather than having one subsystem whose job is to discover a large set of derived substitutions and another whose job is to apply them, one could instead work directly with the rewrite system induced by the original axioms. This is attractive not only because the system becomes conceptually simpler, but also because soundness comes for free from the axioms themselves: any substitution obtained by composing sound substitutions is sound, so there is no need to separately verify each recovered substitution.

This is the perspective we take in this paper. We treat TASO’s discovered substitutions as an empirical target and ask how much of that target can already be generated by the axiom basis alone.

As shown later, this requires a more powerful rewrite system than the one employed by TASO, including bidirectional rewrite rules, occurrence-based matching, and input-output aliasing.

3 Approach and Implementation

3.1 Graph IR and Substitutions

We represent computation graphs using a symbolic graph intermediate representation. A graph is a set of bindings from tensor variables to expressions, with one defining expression

per tensor. Expressions range over the TASO operators, such as `MatMul`, `Conv2D`, `Transpose`, `Concat`, and `Split`, together with non-tensor parameters such as scalars, axes, kernels, strides, padding modes, and activation modes. Inputs are represented by a distinguished Input expression, and outputs are made explicit using Output nodes.

A substitution consists of a source graph, a destination graph, and explicit correspondences between the two sides: a bijection between source and destination input tensors and source and destination output tensors, as well as a bijection between symbolic non-tensor variables. We encode TASO’s base axioms in exactly this format. When an axiom is invertible, the reverse orientation is also available during search. Figure 1 shows one example substitution in our representation.

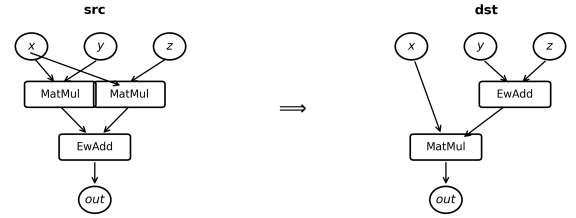


Figure 1. Example substitution represented as a source graph and a destination graph.

3.2 Matching and Application

Given a substitution and a target graph (the graph to which we want to apply the substitution), we first enumerate all matches of the source graph inside the target. Matching starts from a source output tensor and a candidate target tensor, and then recursively compares the two expressions, progressively building a partial match as it proceeds. Source inputs may match arbitrary target tensors, but internal source tensors must match target tensors produced by the same operator with compatible arguments. Symbolic parameters are matched in a sort-correct way, and the matching from source to target must be “decreasing” in terms of concretization (i.e. a variable in the source can match with either a variable or a literal in the target, but a literal in the source can only match with the same literal in the target). A successful match, which happens when the recursion terminates without conflict, records both a tensor map and a symbolic term map.

The tensor map is not required to be injective. This lets a source pattern such as `matmul(x, y)` match a target such as `matmul(x, x)`. This is important in practice, since several TASO substitutions rely on such surjective matches to be discovered.

Once a match has been found, we instantiate the destination side using the symbolic bindings from the match and connect its inputs to the corresponding tensors in the target graph. If the destination output is already one of those inputs, the rewrite is just a redirection. Otherwise, we insert a fresh copy of the destination fragment into the target graph and redirect the consumers of the matched source output to the new result.

A key point is that rewriting is occurrence-local. If a tensor is consumed by multiple expressions, our rewrite system must be able to rewrite only one occurrence. For example, starting from $\text{matmul}(x, x)$, the system can produce $\text{matmul}(x, \text{transpose}(\text{transpose}(x)))$ by rewriting only one use of x . After each rewrite, we remove any now-unused bindings.

3.3 Search

For a target TASO substitution, our goal is to find an explicit directed derivation from one side to the other. A naive forward breadth-first search for either orientation quickly becomes infeasible, since the number of reachable graphs grows exponentially with depth.

Most of the axioms in our system are invertible, with only a small number of exceptions such as the split rules that delete tensors. We use that invertibility to speed up search. For a fixed orientation, we expand one frontier forward from the start graph and another backward from the target using inverse applications of the invertible substitutions. When the two frontiers meet, we combine the two halves into a single directed derivation. Thus, every meeting point yields a full derivation trace rather than only a reachability result.

In addition to the base axioms, our search uses a small set of derived lemmas. These are not introduced as new semantic primitives, but as proof-shortening rules that make long derivations reachable under a bounded search budget. This is consistent with TASO itself (we employ the same lemmas), which also introduces lemma-style shortcuts during verification. Interestingly, this seems to suggest that the "distance" in rewrite space and the "distance" in SMT proof space are closely related metrics.

3.4 Canonicalization

During search, many generated graphs differ only by tensor naming or by redundant intermediate bindings. To avoid exploring the same state repeatedly, we canonicalize every generated graph before inserting it into the search frontier. Canonicalization performs common subexpression elimination, removes bindings not reachable from the outputs, and then renames the remaining tensors deterministically. This is feasible in our setting because the graph representation has ordered inputs at each operator, so canonicalization does not need to solve a general unlabeled graph isomorphism problem.

Table 1. Distribution of derivation lengths for recovered substitutions.

Steps	Count
0	27
1	67
2	205
3	77
4	41
5	13
6	2

4 Results

We evaluated our system on 444 non-isomorphic substitutions extracted from the TASO implementation. A substitution was counted as recovered if our bounded search found an explicit derivation trace connecting its two sides; bidirectional frontier expansion was used only to accelerate that search. Throughout these experiments, we used a maximum search depth of 6 and a maximum number of different graphs considered of 12000.

Under this search configuration, our system recovered **432** of the **444** substitutions. This suggests that the large majority of TASO’s discovered substitutions are obtainable as explicit compositions of axioms under our rewrite system. In addition, the recovered derivations were typically short. Across all 432 recovered substitutions, the mean derivation length was 2.2 steps, the median was 2, and the maximum was 6.

Table 1 summarizes the distribution of derivation lengths. Most recovered substitutions required only two rewrite steps, and only a small number required more than four. This indicates that, for most of the substitution set, the axioms do not merely imply the desired substitutions in principle, but do so through relatively short derivations.

The longest recovered derivations had length 6. These occurred for substitutions 155 and 308. Even these cases remained well within the search bound, while the typical substitution was recovered after only a very small number of rewrite applications.

Even this headline number should be interpreted conservatively. Recovery in our setting does not merely mean showing that two graphs are semantically equivalent or that they admit a common reduct. Rather, it requires an explicit bounded derivation trace in at least one direction. The fact that 432 of 444 substitutions satisfy this stronger criterion suggests not only that the axioms capture most of the same equalities as TASO’s discovered substitution set, but also that they do so in a form that is operationally usable for directed rewriting.

5 Discussion, Limitations, and Next Steps

Our results support two main conclusions. First, the large majority of TASO’s discovered substitutions are obtainable by explicit composition of axioms under a powerful rewrite system. Second, these derivations are usually short. This matters because it suggests that the base axioms are not merely expressive enough in principle, but already close to the discovered substitution set in rewrite “distance”.

At the same time, our results concern bounded recoverability rather than unrestricted derivability. The search is limited by depth and total expansion budget, and in a few places it relies on a small set of derived lemmas to shorten otherwise long proofs. The right interpretation is therefore not that we have completely characterized the closure of TASO’s axiom basis, but that under a realistic explicit rewrite procedure, most of the discovered substitutions can already be reconstructed from that basis.

5.1 Case Study: $\text{transpose}(I) \rightarrow I$

One instructive example is the substitution $\text{transpose}(I) \rightarrow I$. Although the result is semantically simple, it is surprisingly nontrivial to recover by explicit rewriting. The derivation uses three primitive axioms:

```
axiom26: matmul(x, I) → x
axiom9: transpose(transpose(x)) → x
axiom16: transpose(matmul(x, y)) →
  matmul(transpose(y), transpose(x))
```

Here $\text{inv}(\text{axiom } k)$ denotes applying axiom k in the reverse direction. Under our rewrite system, one derivation is:

```
(0) transpose(I) [start]
(1) matmul(transpose(I), I) [inv(axiom26)]
(2) matmul(transpose(I), transpose(transpose(I)))
  [inv(axiom9)]
(3) transpose(matmul(transpose(I), I)) [inv(axiom16)]
(4) transpose(transpose(I)) [axiom26]
(5) I [axiom9]
```

This case illustrates an important feature of the search space induced by the axioms. Even substitutions that appear immediate at the semantic level may lie several substitution steps away when expressed only through primitive substitutions. In practice, this is one reason that a small set of derived lemmas is useful for bounded search: such lemmas do not change what is semantically derivable, but can greatly shorten the path needed to recover it. Interestingly, this lemma was independently discovered by the TASO team when performing SMT verification for their rewrites. We thus also include their lemmas in addition to the base axioms.

5.2 Case Study: A Valid but Unrecovered Rule

One representative unrecovered substitution is the following multi-output transpose–split axis-swap rule:

```
(split0_0(transpose(concat_1(X,Y))),
  split1_0(transpose(concat_1(X,Y))))
→
(split0_1(transpose(concat_0(X,Y))),
  split1_1(transpose(concat_0(X,Y))))
```

Intuitively, this rule says that transposition swaps the concatenation axis and therefore also swaps the corresponding split axis.

This substitution fails in our system for a structural reason. One way to relate the two sides is to reduce both of them to the same intermediate pair by applying the split-elimination axioms. However, those axioms are information-losing: they collapse a $\text{split}(\text{concat}(\cdot, \cdot))$ pattern to one branch and thereby discard the surrounding concatenation structure. Because of this, the common intermediate object can be reached only by using split elimination on both sides. That is enough for semantic equivalence, but it does not yield a clean directional rewrite from left to right or from right to left. In other words, the rule is recoverable only as a meet-in-the-middle argument, not as a single directed derivation.

This example highlights an important aspect of our task. Our goal is not merely to show that two graphs have a common reduct, but to recover a usable directional rewrite trace. For substitutions involving non-invertible, information-losing rules from both sides, these two notions can come apart.

5.3 Next Steps

A natural next step is to analyze the remaining unrecovered substitutions in more detail. These failures do not all appear to have the same cause. Some, such as the multi-output split case above, reflect limitations of the current bounded search procedure and our approach to the problem. Others seem to depend on operator properties that are not yet represented in our IR or axiom basis, such as grouped-convolution semantics, or additional facts about enlarge , which may have been implicitly baked into the TASO verifier and approach but not mentioned in the paper. A more complete classification of these cases would sharpen the interpretation of the current recovery rate and clarify which extensions are actually necessary.

Another next step is to connect derivability back to optimization performance. If most of TASO’s discovered substitutions can already be generated from the base axioms, then a natural question is whether one can optimize graphs effectively using the rewrite system itself rather than first materializing and verifying the larger substitution library, which would be a significant simplification. A direct comparison between optimization driven by explicit axiom rewriting and optimization driven by TASO’s precomputed substitutions would therefore be a valuable follow-up.

References

- [1] Zhihao Jia, Yuanhao He, Dongdong Li, Ali Shafiee, Siddhartha Mahajan, Ahsen L. Esmail, Ximing Guo, Roy Campbell, Song Han, and Karthik Keutzer. 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)*. ACM, 47–62. doi:10.1145/3341301.3359630
- [2] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Immediate Profitability: Relaxed Graph Substitutions for DNN Computation. In *Workshop on Machine Learning and Systems (MLSys)*.
- [3] Colin Unger, Yichen Yang, Zhihao Jia, Karthik Keutzer, and Alex Aiken. 2022. Unity: Accelerating DNN Training Through Joint Optimization of Algebraic Transformations and Parallelization. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.
- [4] Max Willsey, Yisu Remy Wang, Olli Saarikivi, Vincent Panchekha, Zachary Tatlock, and Dan Grossman. 2021. egg: Fast and Extensible Equality Saturation. In *Proceedings of the ACM on Programming Languages*, Vol. 5. ACM, 1–29. doi:10.1145/3434304
- [5] Yichen Yang, Pengfei Su, Mingliang Liu, Chengfeng Li, Rui Wang, Yizhou Wang, Lei Deng, and Yuan Xie. 2021. Equality Saturation for Tensor Graph Superoptimization. In *Proceedings of Machine Learning and Systems (MLSys)*.