Appendix for TensorRight

A Extended Operator Semantics

We provide the denotational semantics for the following additional operators:

- Pad: The pad operator performs low (S_l), interior (S_i), and high (S_h) padding on the input tensor.

$$\frac{\begin{array}{c}\text{let } S = \mathsf{Shape}(e) \qquad S_i \geq 0 \\ \text{let } I = S + (S-1) \times S_i \qquad \text{let } L = I + S_l \qquad \text{let } S_o = L + S_h \geq 0 \\ \text{let } A' = \lambda A.\ \dfrac{A - S_l}{S_i + 1} \qquad \text{let } \mathit{not\text{-}int\text{-}pad} = \lambda A.\ (A - S_l)\ \%\ (S_i + 1) = 0 \\ \text{let } \mathit{not\text{-}pad\text{-}area} = \lambda A.\ A \geq S_l\ \wedge\ A < L\ \wedge\ \mathit{not\text{-}int\text{-}pad}(A)\end{array}}{\begin{array}{l}[\![\mathsf{pad}(e, v, S_l, S_h, S_i)]\!] = \\ \{A \mapsto \text{if } \mathit{not\text{-}pad\text{-}area}(A) \text{ then } [\![e]\!]\,[A'(A)] \text{ else } v \mid A \in \mathsf{Access}(S_o)\}\end{array}}\ \textsc{Pad}$$

- Dot: The dot operator performs sum-of-products over the specified set of contracting aggregated-axes (X_c). X_b denotes the set of batch aggregated-axes that stay independent across the computation. For this operation to be valid, X_c and X_b must be disjoint, and $X_c \cup X_b$ is the set of exactly those aggregated-axes which are common in the input tensors e_1 and e_2. For verification purposes, we describe the semantics of the dot operator by first expanding the two tensors to the same shape, followed by an element-wise multiplication, and then reducing on the contracting aggregated-axes. The remaining set of aggregated-axes in e_1, i.e., $S_1 \setminus S_1|_{X_c \cup X_b}$, and in e_2, i.e., $S_2 \setminus S_2|_{X_c \cup X_b}$, also called the spatial-axes, need to be disjoint. This condition is implicit in the expand operator below.

$$\frac{\begin{array}{c}\text{let } S_1 = \mathsf{Shape}(e_1) \qquad \text{let } S_2 = \mathsf{Shape}(e_2) \\ X_c \cup X_b = \mathsf{Axes}(e_1) \cap \mathsf{Axes}(e_2) \qquad X_c \cap X_b = \varnothing \\ \text{let } t = [\![\mathsf{reduce}(+, \mathsf{binary}(\times, \mathsf{expand}(e_1, S_2 \setminus S_2|_{X_c \cup X_b}), \mathsf{expand}(e_2, S_1 \setminus S_1|_{X_c \cup X_b})), X_c)]\!]\end{array}}{[\![\mathsf{dot}(e_1, e_2, X_c, X_b)]\!] = t}\ \textsc{Dot}$$

- Conv: We first model a basic convolution operator without padding and dilation. This can be expressed on top of slice (for extracting the window) and dot (dot product of the window and the kernel) operators. Then we express the general convolution operator by using the base operator with the pad operator. In the inference rules, the subscript i relates to the input and the subscript w relates to the weights. X_b, X_o, X_f, and X_{sp} are the sets of batch, output-feature, input-feature, and spatial aggregated-axes, respectively.

$$\frac{\begin{array}{ccc}\text{let } S_i = \mathsf{Shape}(e_i) & \text{let } S_w = \mathsf{Shape}(e_2) & \text{let } X_i = \mathsf{Axes}(e_i) \\ \text{let } X_w = \mathsf{Axes}(e_w) & X_b = X_i \setminus X_w & X_o = X_w \setminus X_i \\ F \subseteq X_i \cap X_w & \text{let } X_{sp} = X_i \setminus (X_b \cup X_f) & \mathsf{dom}(I_p) = X_{sp} \\ \multicolumn{3}{c}{\text{let } S' = S_i|_{X_b} \cup S_w|_{X_o} \cup \left\lfloor \dfrac{S_i|_{X_{sp}} - S_w|_{X_{sp}}}{I_p} + 1 \right\rfloor} \\ \multicolumn{3}{c}{\text{let } \mathit{sub} = \lambda A_{sp}.\ [\![\mathsf{dot}(\mathsf{slice}(e_i, A_{sp} \times I_p, A_{sp} \times I_p + S_w|_{X_{sp}}, 1), e_w, X_f \cup X_{sp}, \varnothing)]\!]}\end{array}}{[\![\mathsf{conv\text{-}base}(e_i, e_w, X_b, X_f, X_o, I_p)]\!] = \{A \mapsto \mathit{sub}(A|_{X_{sp}})[A|_{X_b \cup X_o}] \mid A \in \mathsf{Access}(S')\}}\ \textsc{ConvBase}$$

Author's Contact Information:

The conv operator uses the conv-base operator by padding its operands with the specified arguments. Note that the inputs S_i and S_i' to the conv operator denote *dilations*, which are the actual padding amounts incremented by 1. Therefore, we appropriately subtract 1 from these attributes before passing them to the pad operator.

$$\frac{[\![\text{conv-base}(\text{pad}(e_1, 0, S_l, S_h, S_i - 1), \text{pad}(e_2, 0, 0, 0, S_i' - 1), X_b, X_f, X_o, I_p)]\!] = t}{[\![\text{conv}(e_i, e_w, X_b, X_f, X_o, S_l, S_h, S_i, S_i', I_p)]\!] = t}\ \textsc{Conv}$$

- Reverse: The reverse operator reverses the order of elements in the input tensor along the specified set of aggregated-axes X_r.

$$\frac{\begin{array}{c}\text{let } S = \text{Shape}(e) \qquad X_r \subseteq \text{Axes}(e) \\ \text{let } A' = \lambda A.\ (A \setminus A|_{X_r}) \cup (S|_{X_r} - A|_{X_r} - 1)\end{array}}{[\![\text{reverse}(e, X_r)]\!] = \{A \mapsto [\![e]\!]\ [A'(A)] \mid A \in \text{Access}(S)\}}\ \textsc{Reverse}$$

- Select: The select operator constructs an output tensor from elements of two input tensors e_1 and e_2, based on the values of a predicate tensor e_b. This operator is only valid if e_b contains elements of type Bool and e_1, e_2 contain elements of the same type.

$$\frac{\begin{array}{c}\text{Shape}(e_b) = \text{Shape}(e_1) = \text{Shape}(e_2) \\ \text{DType}(e_b) = \text{Bool} \qquad \text{DType}(e_1) = \text{DType}(e_2)\end{array}}{\begin{array}{l}[\![\text{select}(e_b, e_1, e_2)]\!] = \\ \{A \mapsto \text{if } [\![e_b]\!]\ [A] \text{ then } [\![e_1]\!]\ [A] \text{ else } [\![e_2]\!]\ [A] \mid A \in \text{Access}(e)\}\end{array}}\ \textsc{Select}$$

- Clamp: The clamp operator clamps the input tensor (e) to within the range specified by the minimum (e_{min}) and maximum (e_{max}) tensors.

$$\frac{\begin{array}{c}\text{Shape}(e_{min}) = \text{Shape}(e) = \text{Shape}(e_{max}) \\ [\![\text{binary}(\text{min}, \text{binary}(\text{max}, e, e_{min}), e_{max})]\!] = t\end{array}}{[\![\text{clamp}(e_{min}, e, e_{max})]\!] = t}\ \textsc{Clamp}$$

- ReduceSI: The reduce operator takes a tensor and a set of aggregated-axes X as inputs and returns a tensor which contains uninterpreted reduction elements. The resulting tensor has the shape $S \setminus S|_X$, essentially removing all the aggregated-axes in X. This results in the following semantics:

$$\frac{\begin{array}{c}\text{let } S = \text{Shape}(e) \qquad \text{let } \{x_0 \cdots x_k\} = X \subseteq \text{Axes}(e) \\ \text{let } acc = \lambda A.\text{Red}^{\oplus}_{I_0,\cdots,I_k}\ [\![e]\!]\ [\{x_0 \mapsto I_0, \cdots, x_k \mapsto I_k\} \cup A]\end{array}}{[\![\text{reduce}(\oplus, e, X)]\!] = \{A \mapsto acc(A) \mid A \in \text{Access}(S \setminus S|_X)\}}\ \textsc{ReduceOrig}$$

The indices $I_0, \cdots, I_k$ in the function *acc* are *symbolic reduction* indices, hence making the reduction element uninterpreted. The term $[\![e]\!]\ [\{x_0 \mapsto I_0, \cdots, x_k \mapsto I_k\} \cup A]$ represents a (multi-)set containing elements obtained by instantiating the indices $I_0, \cdots, I_k$ with concrete-maps. As the sizes of reduced axes are unbounded, we cannot compute this set of elements and hence cannot expand the reduction to sum all the elements being reduced. Thus, we leave the sum uninterpreted and provide special treatment for such elements during verification. Verifying rules with reductions would involve proving equivalence of reduction elements. Proving the equivalence of two reduction elements $\text{Red}_X f(X)$ and $\text{Red}_Y g(Y)$ can *often* be done by showing that the LHS and RHS are sums of the same values, i.e., $f(X)$ and $g(Y)$ represent the same (multi-)set. One way to prove set equivalence is to establish a bijection between X and Y and show each pair of values in $f(X)$ and $g(Y)$ are equal, regardless of tensor instantiation and operator attributes.

In TensorRight, the user provides a relation between X and Y as a hint. As we see in ReduceOrig, X and Y contain symbolic indices. The current semantics suggest that we have to generate fresh symbolic variables for each invocation of reduce. However, this precludes the user from defining hints as they cannot refer to these indices. To solve this problem, we provide the symbolic reduction indices as input to the operator semantics. One can think of these extra arguments as symbolic *names*, based on which we can define relations between them. The modified semantics are shown below.

$$\frac{\begin{array}{c}\text{let } S = \mathsf{Shape}(e) \qquad \text{let } \{x_0 \mapsto SI_0, \cdots, x_k \mapsto SI_k\} = SI \\ \text{let } X = \mathsf{dom}(I) \subseteq \mathsf{Axes}(e) \\ \text{let } acc = \lambda A.\mathsf{Red}^{\oplus}_{SI_0,\cdots,SI_k} [\![e]\!]\ [SI \cup A]\end{array}}{[\![\mathsf{reduce}(\oplus, e, SI)]\!] = \{A \mapsto acc(A) \mid A \in \mathsf{Access}(S \setminus S|_X)\}}\ \textsc{ReduceSI}$$

The reduce operator now takes an aggregated-map I as input, instead of the set of aggregated-axes to reduce. I contains the symbolic reduction indices $SI_0, \cdots, SI_k$, which are used the construct the uninterpreted reduction element. For any rewrite rule with reductions, the user can now use these supplied reduction indices from LHS and RHS to define hints, also called *si-relations* (symbolic-index relations).

B Validity of Rewrite Rules

Given the semantics of TensorRight DSL, we now define what it means for a rewrite rule to be valid. Fig. 1 lists all the tensor operators supported in TensorRight DSL. As a part of our verification methodology, we symbolically execute all operator semantics and generate symbolic representations of the LHS and RHS expressions. The validity condition of a rewrite rule is simply the assertion of equality between the LHS and RHS expressions, under the specified precondition. We now formalize this notion.

Definition 1 (Valid Rewrite Rule)**.** A rewrite rule $\mathsf{LHS} \Rightarrow_C \mathsf{RHS}$ is valid if and only if

$$\forall v \in vars,\ C \wedge \text{valid-expr}(\mathsf{LHS}) \rightarrow [\![\mathsf{LHS}]\!] = [\![\mathsf{RHS}]\!] \tag{1}$$

where *vars* is the set of all variables appearing in the rewrite rule, such as tensor variables and operator attributes. The predicate valid-expr takes a tensor expression and returns a condition under which the expression is valid. It is defined inductively on the structure of tensor expressions, collecting all assertions from the operator semantics, as shown in Fig. 2.

Definition 1 states that a rewrite rule must be valid for *all* valuations of all the variables appearing in a rewrite rule. Note that we only consider valuations where the term prior to rewriting is valid, i.e., when LHS is valid. The inner equality $[\![\mathsf{LHS}]\!] = [\![\mathsf{RHS}]\!]$ asserts that, once all the variables are instantiated, the resulting tensors on both sides are equal. §G extends this definition to account for validity of the RHS expression.

Based on the denotational semantics of operators in TensorRight DSL, we can symbolically compute the shape of any tensor expression as a function of the shapes of input tensors and operator attributes. We only consider rewrite rules where under the precondition, the LHS and RHS expressions have the same shape, since the rule would be invalid if the shapes do not match. Under this assumption, semantic validity reduces to a pointwise equality of tensor values:

$$\forall v \in vars, C \wedge \text{valid-expr}(\mathsf{LHS}) \rightarrow \forall A \in \mathsf{Access}(\mathsf{LHS}),\ [\![\mathsf{LHS}]\!]\ [A] = [\![\mathsf{RHS}]\!]\ [A] \tag{2}$$

Definition 2 (Invalid Rewrite Rule)**.** A rewrite rule $\mathsf{LHS} \Rightarrow_C \mathsf{RHS}$ is invalid if and only if

$$\exists v \in vars, C \wedge \text{valid-expr}(\mathsf{LHS}) \wedge \exists A \in \mathsf{Access}(\mathsf{LHS}),\ [\![\mathsf{LHS}]\!]\ [A] \neq [\![\mathsf{RHS}]\!]\ [A]$$

where *vars* is the set of variables appearing in the rewrite rule.

τ	$:=$	Int | Bool | Real	Type
a	$\in$	$\mathcal{A}$	Named-axes
x	$\in$	$\mathcal{X} = \mathcal{P}(\mathcal{A})$	Aggregated-axes
f	$\in$	list[Int] $\rightarrow$ Int	Map function
m	$:=$	$\mathcal{M}$ | fmap$(f, m+)$	Maps
X	$\in$	$\mathcal{P}(\mathcal{X})$	Set of aggregated-axes
S, I	$\in$	$m^{\mathcal{X}}$	Shapes and indices
R	$\in$	$\mathcal{X}^{\mathcal{X}}$	Relabel maps
v	$:=$	i : Int | b : Bool | r : Real	Scalar literal
e	$:=$	$\mathcal{T}$ (Literal)	Tensor expression
	|	$\mathcal{V}$ (Variable)	
	|	const(v, S)	
	|	iota(S, x)	
	|	expand(e, S)	
	|	binary$(\oplus, e_l, e_r)$	
	|	pad(e, v, S_l, S_h, S_i)	
	|	slice(e, I_s, I_e, I_p)	
	|	dy-slice(e, I, S)	
	|	dyup-slice(e, e_u, I)	
	|	reduce$(\oplus, e, I)$	
	|	relabel(e, R)	
	|	concat(e_l, e_h, x)	
	|	conv-base$(e_i, e_w, X_b, X_f, X_o, I_p)$	
	|	conv$(e_i, e_w, X_b, X_f, X_o, S_l, S_h, S_i, S_i', I_p)$	
	|	dot(e_1, e_2, X_c, X_b)	
	|	clamp(e_{min}, e, e_{max})	
	|	select(e_b, e_1, e_2)	
	|	reverse(e, X_r)	
g	$\in$	list[Int] $\rightarrow$ Bool	Predicate function
P	$:=$	fold$(g, m+)$	Precondition
$Rule$	$:=$	$e_{lhs} \Rightarrow_{P*} e_{rhs}$	Rewrite rule

Fig. 1. Core rewrite rule representation with extended operators.

Definition 2 states that if a rewrite rule is invalid, then there exists some valuation of all variables and an access A, such that (1) the precondition is true, (2) LHS is valid, and (3) LHS and RHS do not match at the access A. Such a valuation of variables and the access A together comprise a *counterexample* for the rule.

C Preliminaries

In §B, we defined what it means for a rewrite rule to be valid (Equation 1). A key challenge is that rewrite rules must apply to tensors with arbitrary ranks and arbitrary sizes along each axis. Thus, we must verify rewrite rules in the *unbounded setting*, where both the number of axes (rank) and the size along each axis are unbounded. To handle arbitrary number of axes, we partition the named-axes of a tensor into finite groups or classes, called aggregated-axes, where each aggregated-axis can be instantiated to any rank. Once the aggregated-axes in a rewrite rule are instantiated, we obtain a concrete-rank instantiation of the rule, though the sizes along each axis remain unbounded.

Basis:

$$\begin{aligned}
&\text{valid-expr}(\underline{t}) \coloneqq \text{true} && \underline{t} \in \mathcal{T}\\
&\text{valid-expr}(var) \coloneqq \text{Shape}(var) \geq 0 && var \in \mathcal{V}\\
&\text{valid-expr}(\text{const}(v, S)) \coloneqq S \geq 0\\
&\text{valid-expr}(\text{iota}(S, x)) \coloneqq S \geq 0 \wedge x \in S
\end{aligned}$$

Induction Step:

$$\begin{aligned}
&\text{valid-expr}(\text{expand}(e, S)) \coloneqq \text{valid-expr}(e) \wedge S \geq 0 \wedge \text{dom}(S) \cap \text{Axes}(e) = \varnothing\\
&\text{valid-expr}(\text{binary}(\oplus, e_l, e_r)) \coloneqq \text{valid-expr}(e_l) \wedge \text{valid-expr}(e_r) \wedge \text{Shape}(e_l) = \text{Shape}(e_r)\\
&\text{valid-expr}(\text{slice}(e, I_s, I_e, I_p)) \coloneqq \text{valid-expr}(e) \wedge 0 \leq I_s \leq I_e \leq \text{Shape}(e) \wedge I_p > 0\\
&\text{valid-expr}(\text{dy-slice}(e, I, S)) \coloneqq \text{valid-expr}(e) \wedge I + S \leq \text{Shape}(e) \wedge S > 0 \wedge I \geq 0\\
&\qquad\qquad\vdots
\end{aligned}$$

Fig. 2. The predicate valid-expr is defined inductively over the structure of tensor expressions. Other cases follow similarly, derived from the operator semantics.

Verifying the rewrite rule in the unbounded setting therefore reduces to verifying the rule for all valid instantiations of the aggregated-axes. In this section, we present our key observation (§C.2), and in §D we show how TensorRight verifies any rewrite rule for a possibly infinite number of instantiations.

C.1 Assumptions

To simplify the discussion, we make the following assumptions, which will be relaxed later. Let LHS $\Rightarrow_C$ RHS be a rewrite rule such that:

- The rewrite rule contains only one input tensor, denoted X
- X contains exactly one aggregated-axis, say x, which has the RClass c, i.e. $x : c$.
- The rewrite rule does not contain any reduce, relabel, expand, dot, conv-base, and conv operators: these operators can add, remove or rename aggregated-axes. For now, we keep the discussion restricted to a single aggregated-axis.

C.2 Observation

We can symbolically execute any rewrite rule LHS $\Rightarrow_C$ RHS under a general access A using the operator semantics. We observe that the equality $[\![\text{LHS}]\!]\,[A] = [\![\text{RHS}]\!]\,[A]$ (the inner equality in Equation 2) can always be expressed in the following canonical form:

$$\begin{aligned}
&\text{scalarf}(\mathsf{X}[x \mapsto \text{fmap}(e_1, \mathcal{M}_1)],\ \cdots,\ \mathsf{X}[x \mapsto \text{fmap}(e_n, \mathcal{M}_n)],\\
&\qquad \text{fold}(g_1, \mathcal{N}_1),\ \cdots,\ \text{fold}(g_m, \mathcal{N}_m))
\end{aligned} \tag{3}$$

We now illustrate, through examples, how symbolic evaluation of rewrite rules results in expressions of this form.

C.3 Symbolic Evaluation Examples

C.3.1 Pad-Low.

Setting. Let $x : c$ be an aggregated-axis with RClass c. It can be instantiated to an arbitrary number of named-axes. Let X be a tensor with shape $\{x \mapsto s\}$, where s contains sizes of named-axes in x. Let the rewrite rule in consideration be,

$$\text{pad-low}(\mathsf{X}, 0, S_l) \Rightarrow \text{pad-low}(\mathsf{X}, 1, S_l)$$

where $S_l = \{x \mapsto s_l\}$ and s_l contains padding sizes of named-axes in x. The difference between LHS and RHS is that the former is padded with a value of 0, while the latter is padded with a value of 1. Therefore, this rewrite rule is invalid.

Shape of the Output Tensors. We compute the shape of the LHS (same as RHS) as follows:

$$\begin{aligned}\mathsf{Shape}(\text{pad-low}(\mathsf{X}, 0, S_l)) &= \mathsf{Shape}(\mathsf{X}) + S_l \\ &= \{x \mapsto s\} + \{x \mapsto s_l\} \\ &= \{x \mapsto s + s_l\}\end{aligned}$$

Therefore, an arbitrary, valid access A on the output tensors has the form $A = \{x \mapsto a\}$, where a contains the index-values for the named-axes in x. We symbolically evaluate the expression $[\![\mathsf{LHS}]\!]\,[A] = [\![\mathsf{RHS}]\!]\,[A]$ under the access A:

$$\begin{aligned}&[\![\mathsf{LHS}]\!]\,[A] = [\![\mathsf{RHS}]\!]\,[A] \\ \Leftrightarrow\ &(\text{if } \textit{not-pad}(A) \text{ then } \mathsf{X}[x \mapsto (a - s_l)] \text{ else } 0) = \\ &\qquad (\text{if } \textit{not-pad}(A) \text{ then } \mathsf{X}[x \mapsto (a - s_l)] \text{ else } 1)\end{aligned}$$

We express the above in the canonical form given by Equation 3, as follows:

$$\begin{aligned}e(v, l) &\stackrel{\text{df}}{=} v - l \\ y &\stackrel{\text{df}}{=} \mathsf{X}[x \mapsto \mathsf{fmap}(e, [a, s_l])] = \mathsf{X}[x \mapsto a - s_l] \\ g(v, l) &\stackrel{\text{df}}{=} v \geq l \\ b &\stackrel{\text{df}}{=} \textit{not-pad}(A) = A \geq S_l \\ &= a \geq s_l = \mathsf{fold}(g, [a, s_l]) \\ \mathsf{scalarf}(y, b) &\stackrel{\text{df}}{=} (\text{if } b \text{ then } y \text{ else } 0) = (\text{if } b \text{ then } y \text{ else } 1) \\ [\![\mathsf{LHS}]\!]\,[A] = [\![\mathsf{RHS}]\!]\,[A] &\stackrel{\text{df}}{=} \mathsf{scalarf}(\mathsf{X}[x \mapsto \mathsf{fmap}(e, [a, s_l])], \mathsf{fold}(g, [a, s_l]))\end{aligned}$$

C.3.2 Slice.

Setting. Let $x : c$ be an aggregated-axis with RClass c. It can be instantiated to an arbitrary number of named-axes. Let X be a tensor with shape $\{x \mapsto s\}$, where s contains sizes of named-axes in x. Let the rewrite rule in consideration be,

$$\text{slice}(\text{pad-low}(\mathsf{X}, 0, S_l), S_l, S + S_l, S_p) \Rightarrow \mathsf{X}$$

where $S = \mathsf{Shape}(\mathsf{X})$, $S_l = \{x \mapsto s_l\}$, and $S_p = \{x \mapsto 1\}$.

The LHS first pads the input tensor by 0 with padding sizes s_l. It then slices the resulting tensor by removing s_l elements from the beginning along every named-axis, while skipping no elements till the end. It is easy to see that the LHS is a no-op since it adds padding values and then removes them. The RHS is simply the input tensor. Therefore, this rewrite rule is valid.

Shape of the output tensors. We compute the shape of the LHS (same as RHS) as follows:

$$\begin{aligned}\mathsf{Shape}(\mathsf{slice}(\mathsf{pad\text{-}low}(\mathsf{X}, 0, S_l), S_l, S + S_l, S_p) &= S + S_l - S_l \\ &= S \\ &= \{x \mapsto s\}\end{aligned}$$

Therefore, an arbitrary, valid access A on the output tensors has the form $A = \{x \mapsto a\}$, where a contains the index-values for the named-axes in x. We symbolically evaluate the expression $[\![\mathsf{LHS}]\!]\,[A] = [\![\mathsf{RHS}]\!]\,[A]$ under the access A:

$$\begin{aligned}&[\![\mathsf{LHS}]\!]\,[A] = [\![\mathsf{RHS}]\!]\,[A] \\ \Leftrightarrow\ &[\![\mathsf{pad}(\mathsf{X}, 0, S_l)]\!]\,[S_l + A] = \mathsf{X}[A] \\ \Leftrightarrow\ &(\text{if } \mathit{not\text{-}pad}(S_l + A) \text{ then } \mathsf{X}[(A + S_l) - S_l] \text{ else } 0) = \mathsf{X}[A] \\ \Leftrightarrow\ &(\text{if } \mathit{not\text{-}pad}(S_l + A) \text{ then } \mathsf{X}[A] \text{ else } 0) = \mathsf{X}[A]\end{aligned}$$

We express the above in the canonical form given by Equation 3, as follows:

$$\begin{aligned}e(v) &\stackrel{\text{df}}{=} v \\ y &\stackrel{\text{df}}{=} \mathsf{X}[x \mapsto \mathsf{fmap}(e, [a])] = \mathsf{X}[x \mapsto a] \\ g(v, l) &\stackrel{\text{df}}{=} v + l \geq l \\ b &\stackrel{\text{df}}{=} \mathit{not\text{-}pad}(A + S_l) = A + S_l \geq S_l \\ &= a + s_l \geq s_l = \mathsf{fold}(g, [a, s_l]) \\ \mathsf{scalarf}(y, b) &\stackrel{\text{df}}{=} (\text{if } b \text{ then } y \text{ else } 0) = y \\ [\![\mathsf{LHS}]\!]\,[A] = [\![\mathsf{RHS}]\!]\,[A] &\stackrel{\text{df}}{=} \mathsf{scalarf}(\mathsf{X}[x \mapsto \mathsf{fmap}(e, [a])], \mathsf{fold}(g, [a, s_l]))\end{aligned}$$

C.3.3 Multiple RClasses.

Setting. Let $x_1 : c_1$ and $x_2 : c_2$ be aggregated-axes with RClasses c_1 and c_2 respectively. They can be instantiated to an arbitrary number of named-axes. Let X be a tensor with shape $\{x_1 \mapsto s_1\}$, where s_1 contains sizes of named-axes in x_1.

Let the rewrite rule rule in consideration be,

$$\mathsf{pad\text{-}low}(\mathsf{expand}(\mathsf{X}, \{x_2 \mapsto s_2\}), 0, S_{l_1}) \Rightarrow \mathsf{expand}(\mathsf{pad\text{-}low}(\mathsf{X}, 0, S_{l_2}), \{x_2 \mapsto s_2\})$$

where $S_{l_1} = \{x_1 \mapsto s_l, x_2 \mapsto 0\}$, $S_{l_2} = \{x_1 \mapsto s_l\}$, and s_l contains padding sizes for named-axes in x_1. The LHS first expands the tensor X with a new aggregated-axis x_2 and then low-pads x_1 with s_l and leaves x_2 unchanged. The RHS first low-pads x_1 with s_l and then expands with a new aggregated-axis x_2. This is a valid rewrite rule since LHS does not pad x_2 after the expansion, while RHS simply replicates the data across x_2.

Shape of the output tensors. We compute the shape of the LHS (same as RHS) as follows:

$$\begin{aligned}\mathsf{Shape}(\mathsf{pad\text{-}low}(\mathsf{expand}(\mathsf{X}, \{x_2 \mapsto s_2\}), 0, S_{l_1})) &= \mathsf{Shape}(\mathsf{expand}(\mathsf{X}, \{x_2 \mapsto s_2\})) + S_{l_1} \\ &= \mathsf{Shape}(\mathsf{X}) \cup \{x_2 \mapsto s_2\} + S_{l_1} \\ &= \{x_1 \mapsto s_1, x_2 \mapsto s_2\} + \{x_1 \mapsto s_l, x_2 \mapsto 0\} \\ &= \{x_1 \mapsto s_1 + s_l, x_2 \mapsto s_2\}\end{aligned}$$

Therefore, an arbitrary, valid access A on the output tensors has the form $A = \{x_1 \mapsto a_1, x_2 \mapsto a_!\}$, where a_1 and a_2 contain the index-values for the named-axes in x_1 and x_2 respectively. We symbolically evaluate the expression $[\![\mathsf{LHS}]\!]\ [A] = [\![\mathsf{RHS}]\!]\ [A]$ under the access A:

$$
\begin{aligned}
&[\![\mathsf{LHS}]\!]\ [A] = [\![\mathsf{RHS}]\!]\ [A] \\
\Leftrightarrow\ &(\text{if } \mathit{not\text{-}pad}(A) \text{ then } [\![\mathsf{expand}(\mathsf{X}, \{x_2 \mapsto s_2\})]\!]\ [A - S_{l_1}] \text{ else } 0) = \\
&\qquad [\![\mathsf{pad\text{-}low}(\mathsf{X}, 0, S_{l_2})]\!]\ [A|_{x_1}] \\
\Leftrightarrow\ &(\text{if } \mathit{not\text{-}pad}(A) \text{ then } \mathsf{X}[(A - S_{l_1})|_{x_1}] \text{ else } 0) = \\
&\qquad (\text{if } \mathit{not\text{-}pad}(A|_{x_1}) \text{ then } \mathsf{X}[A|_{x_1} - S_{l_2}] \text{ else } 0) \\
\Leftrightarrow\ &(\text{if } \mathit{not\text{-}pad}(A) \text{ then } \mathsf{X}[x_1 \mapsto a_1 - s_l] \text{ else } 0) = \\
&\qquad (\text{if } \mathit{not\text{-}pad}(A|_{x_1}) \text{ then } \mathsf{X}[x_1 \mapsto a_1 - s_l] \text{ else } 0)
\end{aligned}
$$

We express the above in the canonical form given by Equation 3, as follows:

$$
\begin{aligned}
e(v, l) &\stackrel{\text{df}}{=} v - l \\
y &\stackrel{\text{df}}{=} \mathsf{X}[x_1 \mapsto \mathsf{fmap}(e, [a_1, s_l])] = \mathsf{X}[x_1 \mapsto a_1 - s_l] \\
g_1(v, l) &\stackrel{\text{df}}{=} v \geq l \\
g_2(v, l) &\stackrel{\text{df}}{=} v \geq 0 \\
b_1 &\stackrel{\text{df}}{=} \mathit{not\text{-}pad}(A) = A \geq S_{l_1} \\
&\quad = a_1 \geq s_l \wedge a_2 \geq 0 = \mathsf{fold}(g_1, [a_1, s_l]) \wedge \mathsf{fold}(g_2, [a_2]) \\
g_3 &\stackrel{\text{df}}{=} \mathsf{true} \\
b_2 &\stackrel{\text{df}}{=} \mathit{not\text{-}pad}(A|_{x_1}) = a_1 \geq s_l \\
&\quad = a_1 \geq s_l \wedge \mathsf{true} = \mathsf{fold}(g_1, [a_1, s_l]) \wedge \mathsf{fold}(g_3, [\,]) \\
\mathsf{scalarf}(y, b_1, b_2) &\stackrel{\text{df}}{=} (\text{if } b_1 \text{ then } y \text{ else } 0) = (\text{if } b_2 \text{ then } y \text{ else } 0) \\
[\![\mathsf{LHS}]\!]\ [A] = [\![\mathsf{RHS}]\!]\ [A] &\stackrel{\text{df}}{=} \mathsf{scalarf}(\mathsf{fmap}(e, [a_1, s_l]), \mathsf{fold}(g_1, [a_1, s_l]) \wedge \mathsf{fold}(g_2, [a_2]), \\
&\qquad\qquad \mathsf{fold}(g_1, [a_1, s_l]) \wedge \mathsf{fold}(g_3, [\,]))
\end{aligned}
$$

Here, the condition b_2 has been "padded" with a *trivial* fold over the aggregated-axis x_2, using a constant predicate function $g_3 = \mathsf{true}$. This notational convention ensures that each condition contains exactly one fold per aggregated-axis, some of which may be trivial. Such trivial folds are introduced for uniformity and can be safely removed during postprocessing.

C.3.4 Key Takeaway. The observation that rewrite rules in TensorRight DSL can be symbolically evaluated and expressed in the form given by Equation 3, extends to *all* rewrite rules in the language. This generalization can be formally proven by induction on the structure of tensor expressions. We omit the proof, as the structure of the argument follows directly from the operator semantics.

D Scalar Function Explained

In §C.3, we illustrated through examples how any rewrite rule LHS $\Rightarrow_C$ RHS in TensorRight DSL can be symbolically evaluated under a general access A using the operator semantics. The assumptions from §C.1 still apply here and will be incrementally relaxed in later sections. The

equality $[\![\text{LHS}]\!]\,[A] = [\![\text{RHS}]\!]\,[A]$ can be expressed in the following form:

$$\text{scalarf}(\mathsf{X}[x \mapsto \text{fmap}(e_1, \mathcal{M}_1)],\ \cdots,\ \mathsf{X}[x \mapsto \text{fmap}(e_n, \mathcal{M}_n)],\\ \text{fold}(g_1, \mathcal{N}_1),\ \cdots,\ \text{fold}(g_m, \mathcal{N}_m))$$

We explain each component below:

- $\mathsf{X}[\_]$: this represents an access to the tensor X.
- $x \mapsto \text{fmap}(e_i, \mathcal{M}_i)$: this represents the access expression for the i^{th} access to X. Since X has only one aggregated-axis x, the access expression contains the same aggregated-axis, with the access map $\text{fmap}(e_i, \mathcal{M}_i)$.
  The fmap construct takes a function and applies it pointwise to a list of maps with the same domain. It as defined a follows:
  $$\text{fmap}(f, [m_1, m_2, \cdots]) = \{i \mapsto f(m_1(i), m_2(i), \cdots) \mid i \in \text{dom}(m_1)\}$$
  For example, if $\mathcal{M} = [\{i \mapsto v_i, j \mapsto v_j\}, \{i \mapsto l_i, j \mapsto l_j\}]$ and $f = \lambda v, l.(v - l)$, then $\text{fmap}(f, \mathcal{M}) = \{i \mapsto v_i - l_i, j \mapsto v_j - l_j\}$.
  Here, $\mathcal{M}_i$ is the list of maps used in the i^{th} access, which include access maps, attribute maps, etc. Each function e_i is independent of the rank of x, and only the maps in $\mathcal{M}_i$ vary with the rank of x. Thus, we are able to capture all the *rank-independent* information in the function e_i. We refer to e_i as an *index transformer* because it transforms output index-values to input index-values. For the fmap construct to be well-formed, the arity of e_i must match the number of maps in $\mathcal{M}_i$.
  When X has multiple aggregated-axes, each access must have separate access maps for each aggregated-axis. For example, if $\text{Axes}(\mathsf{X}) = \{x_1, \cdots, x_p\}$, then the i^{th} access to X has the following form:
  $$\mathsf{X}[x_1 \mapsto \text{fmap}({}^1e_i, {}^1\mathcal{M}_i),\ \cdots, x_p \mapsto \text{fmap}({}^pe_i, {}^p\mathcal{M}_i)]$$
  where ${}^je_i$ denotes the index transformer for x_j in the i^{th} access, and ${}^j\mathcal{M}_i$ is the corresponding list of maps. Each ${}^je_i$ may vary across different aggregated-axes. However, it is applied to all named-axes uniformly in a given aggregated-axis.
- $\text{fold}(g_j, \mathcal{N}_j)$: these represent boolean values, referred to as *conditions*, that appear within if-then-else blocks. These conditions capture the dependency of output tensor values on the input tensor values and originate directly from the operator semantics. For instance, *not-pad* in the pad semantics (Pad) takes an access A and determines if it lies in the padded area or not.
  The fold construct takes a predicate and applies it pointwise to a list of maps with the same domains. It returns true if all predicate values are true, and false otherwise. It is defined as:
  $$\text{fold}(g, [m_1, m_2, \cdots]) = \bigwedge_{i \in \text{dom}(m_1)} g(m_1(i), m_2(i), \cdots)$$
  For example, if $\mathcal{N} = [\{i \mapsto v_i, j \mapsto v_j\}, \{i \mapsto l_i, j \mapsto l_j\}]$ and $g = \lambda v, l.(v \geq l)$, then $\text{fold}(g, \mathcal{N}) = v_i \geq l_i \wedge v_j \geq l_j$.
  Here, $\mathcal{N}_j$ is the list of maps used in the j^{th} condition, which include access maps, attribute maps, etc. Each function g_j is independent of the rank of the aggregated-axis x, and only the maps in $\mathcal{N}_j$ vary with the rank of x. Thus, we are able to capture all the *rank-independent* information in the function g_j. For the fold construct to be well-formed, the arity of g_j must match the number of maps in $\mathcal{N}_j$.
  In case where the rewrite rule has multiple aggregated-axes, we observe that all such conditions can be expressed as a conjunction of folds, one for each aggregated-axis. More

specifically, if $x_1 \cdots x_p$ are the aggregated-axes appearing in a rule, then the j^{th} condition has the form:

$$\bigwedge_{i=1\cdots p} \text{fold}({}^{i}g_j, {}^{i}\mathcal{N}_j)$$

where ${}^{i}g_j$ is the j^{th} condition for the aggregated-axis x_i, and ${}^{i}\mathcal{N}_j$ is the corresponding list of maps. Some important observations:

- Some of the ${}^{i}g'_j s$ may be *trivial*, i.e., they return true for all inputs. In that case, the corresponding fold always evaluates to true and does not contribute to the actual condition.
- Introducing trivial folds serves a notational purpose: every condition is uniformly represented as a conjunction of one fold per aggregated-axis, regardless of whether that axis contributes meaningful conditions. This convention simplifies the representation and such trivial folds can be removed once they're no longer needed.
  §C.3.3 illustrates an example where a condition only involving x_1 was extended with a trivial fold over x_2 to maintain uniformity.
- Each ${}^{i}g_j$ may vary across different aggregated-axes. However, it is applied to all named-axes uniformly in a given aggregated-axis.

- scalarf: this is a scalar function which returns a boolean value, indicating whether the values of LHS and RHS tensors match at a given access A. We refer to it as scalarf since it captures the core, *scalar computation* in the expression, typically consisting of arithmetic and conditionals over scalars.
  Based on the arguments to scalarf, we say: scalarf comprises *n accesses to X* and *m conditions*. Just like index transformers, scalarf is independent of the rank of x.

We now rewrite Equation 2 in terms of the scalarf function, accesses, and conditions, to get the validity condition of a rewrite rule:

$$\begin{aligned}\forall v \in vars, C \wedge \text{valid-expr}(\text{LHS}) \rightarrow \forall A \in \text{Access}(\text{LHS}),\\ \text{scalarf}(\text{X}[x \mapsto \text{fmap}(e_1, \mathcal{M}_1)],\ \cdots,\ \text{X}[x \mapsto \text{fmap}(e_n, \mathcal{M}_n)],\\ \text{fold}(g_1, \mathcal{N}_1),\ \cdots,\ \text{fold}(g_m, \mathcal{N}_m))\end{aligned} \tag{4}$$

D.1 Verification Methodology

Definition 3 (RClass-rank Map)**.** Let R be a rewrite rule with RClasses $c_1, \cdots, c_p$. An *RClass-rank map* is a mapping $\{c_1 \mapsto r_1, \cdots, c_p \mapsto r_p\}$, that assigns a concrete rank r_i to each RClass c_i, representing a specific rank instantiation of the rule R.

Definition 4 (Validity for Concrete RClass Ranks)**.** Let R be a rewrite rule and m be an RClass-rank map for R. Then the predicate $\text{valid}(R, m)$ holds if and only if:

$$\text{valid}(R, m) \Leftrightarrow R \text{ is valid for the number of named-axes as specified by } m$$

This corresponds to a concrete, bounded verification instance: the ranks of all RClasses (and thus of all aggregated-axes) are fixed according to m, while the sizes of the named-axes remain arbitrary.

Let $R = \text{LHS} \Rightarrow_C \text{RHS}$ be the rewrite rule that we wish to verify in the unbounded setting. For simplicity, the assumptions from §C.1 still apply. We identify a *sufficient* rank k such that:

$$\forall\ i \geq k,\ \text{valid}(R, \{c \mapsto i\}) \rightarrow \text{valid}(R, \{c \mapsto i{+}1\}) \tag{5}$$

This means that for all $i \geq k$, if the rule is valid when the RClass c has rank i, then it is also valid when c has rank $i{+}1$. Given such a sufficient rank k, we can verify the rule in the unbounded setting using induction on tensor ranks:

- Basis: Use bounded verification to prove that the rule is valid for all ranks up to k:

$$\bigwedge_{i=1\cdots k} \text{valid}(\{c \mapsto i\})$$

- Induction Step: Use induction on the rank of c with Equation 5 as induction hypothesis:

$$\text{valid}(\{c \mapsto k\}) \wedge [\forall i \geq k,\ \text{valid}(\{c \mapsto i\}) \rightarrow \text{valid}(\{c \mapsto i{+}1\})] \implies \bigwedge_{i \geq k} \text{valid}(\{c \mapsto i\})$$

In other words, the rule is valid for any number of named-axes in the RClass c. What remains is to determine a sufficient rank k for each RClass in a given rule.

We begin by analyzing the validity condition of the rule when the aggregated-axis x is instantiated with a concrete rank i, i.e., when $\text{valid}(\{c \mapsto i\})$ holds. We first instantiate x to x^i, a concrete aggregated-axis containing i distinct named-axes, namely $\{\underline{1}, \cdots, \underline{i}\}$. We similarly instantiate the precondition C, input tensor X, list of maps $\mathcal{M}_j$ and $\mathcal{N}_j$, operator attributes, and tensor expressions. Thus, we can expand the validity $\text{valid}(\{c \mapsto i\})$ by rewriting Equation 4 as:

$$\begin{aligned}\forall v^i \in \mathit{vars}^i, C^i \wedge \text{valid-expr}(\mathsf{LHS}^i) \rightarrow \forall A^i \in \text{Access}(\mathsf{LHS}^i),\\ \text{scalarf}(\mathsf{X}^i[x^i \mapsto \text{fmap}(e_1, \mathcal{M}_1^i)],\ \cdots,\ \mathsf{X}^i[x^i \mapsto \text{fmap}(e_n, \mathcal{M}_n^i)],\\ \text{fold}(g_1, \mathcal{N}_1^i),\ \cdots,\ \text{fold}(g_m, \mathcal{N}_m^i))\end{aligned} \tag{6}$$

As noted previously, the scalar function scalarf, the index transformers e_j, and the condition predicates g_j are independent of the rank of x. The only components that vary with the rank i are the variables vars^i, access A^i, and list of maps M_j^i and $\mathcal{N}_j^i$.

To compute a sufficient rank k, we start from the inductive goal $[\text{valid}(\{c \mapsto k\}) \rightarrow \text{valid}(\{c \mapsto k{+}1\})]$. We instead work with its contrapositive,

$$\text{valid}(\{c \mapsto k\}) \rightarrow \text{valid}(\{c \mapsto k{+}1\}) \Leftrightarrow \neg\text{valid}(\{c \mapsto k{+}1\}) \rightarrow \neg\text{valid}(\{c \mapsto k\})$$

Intuitively, $\neg\text{valid}(\{c \mapsto i\})$ holds if there exists a counterexample to the rewrite rule at rank i. Our goal is to find a sufficient rank k such that any counterexample at rank $k{+}1$ (or higher) can be *lowered* to a counterexample at rank k.

On expanding the validity condition using Equation 6, the contrapositive becomes:

$$\begin{aligned}&\exists\, v^{k+1} \in \mathit{vars}^{k+1}, C^{k+1} \wedge \text{valid-expr}(\mathsf{LHS}^{k+1}) \wedge \exists A^{k+1} \in \text{Access}(\mathsf{LHS}^{k+1}),\\ &\quad \neg\text{scalarf}(\mathsf{X}^{k+1}[x^{k+1} \mapsto \text{fmap}(e_1, \mathcal{M}_1^{k+1})],\ \cdots,\ \mathsf{X}^{k+1}[x^{k+1} \mapsto \text{fmap}(e_n, \mathcal{M}_n^{k+1})],\\ &\qquad \text{fold}(g_1, \mathcal{N}_1^{k+1}),\ \cdots,\ \text{fold}(g_m, \mathcal{N}_m^{k+1}))\\ &\qquad\qquad \downarrow\\ &\exists\, v^k \in \mathit{vars}^k, C^k \wedge \text{valid-expr}(\mathsf{LHS}^k) \wedge \exists A^k \in \text{Access}(\mathsf{LHS}^k),\\ &\quad \neg\text{scalarf}(\mathsf{X}^k[x^k \mapsto \text{fmap}(e_1, \mathcal{M}_1^k)],\ \cdots,\ \mathsf{X}^k[x^k \mapsto \text{fmap}(e_n, \mathcal{M}_n^k)],\\ &\qquad \text{fold}(g_1, \mathcal{N}_1^k),\ \cdots,\ \text{fold}(g_m, \mathcal{N}_m^k))\end{aligned}$$

This means that we are given a counterexample at rank $k{+}1$, which comprises:

- A tensor X^{k+1} with $k{+}1$ named-axes in x^{k+1}, having the shape $\{x^{k+1} \mapsto m\}$, where $m = \{\underline{1} \mapsto n_1,\ \cdots,\ \underline{k{+}1} \mapsto n_{k+1}\}$,
- A valuation of variables vars^{k+1} such that the precondition C^{k+1} is satisfied and the expression LHS^{k+1} is valid, and
- An access $A^{k+1} \in \text{Access}(\mathsf{LHS}^{k+1})$, where the output tensors do not match.

To lower this counterexample from rank k+1 to rank k, we must construct:

- A tensor X^k with k named-axes in x^k,
- A valuation of variables $vars^k$ such that the precondition C^k is satisfied and the expression LHS^k is valid, and
- An access $A^k \in \mathsf{Access}(\mathsf{LHS}^k)$, where the output tensors do not match.

Our goal is to derive lower bounds on k while ensuring that any $(k{+}1)$-ranked counterexample can be lowered to a k-ranked counterexample. This lower bound serves as a sufficient rank for the inductive argument.

D.1.1 Counterexample Construction. Our counterexample construction algorithm involves *projecting* the $(k{+}1)$-ranked aggregated-axis x^{k+1} to a k-ranked aggregated-axis x^k by choosing k named-axes from $\{\underline{1}, \cdots, \underline{k{+}1}\}$. There are k+1 such projections in total, but not all of them necessarily yield a valid counterexample at rank k. We express this construction using a set of equations and use them to derive *constraints* on valid projections.

Projection Notation. Let $\Gamma \subset x$ be any projection of the aggregated-axis x. We define projection operations on different constructs as follows:

- Map Projection: Given a map m defined on an aggregated-axis x (i.e., $\mathsf{dom}(m) = x$), its projection to Γ is defined as:

$$m|_\Gamma^x \stackrel{\text{df}}{=} \{\underline{j} \mapsto m(\underline{j}) \mid \underline{j} \in \Gamma\}$$

  If m is defined on a different aggregated-axis $x' \neq x$, then $m|_\Gamma^x \stackrel{\text{df}}{=} m$.
- Aggregated-Map Projection: Given an aggregated-map M that contains the aggregated-axis x, its projection to Γ is defined as:

$$M|_\Gamma^x \stackrel{\text{df}}{=} (M \setminus \{x \mapsto M(x)\}) \cup \{\Gamma \mapsto M(x)|_\Gamma^x\}$$

  If M does not contain x, then $M|_\Gamma^x \stackrel{\text{df}}{=} M$.
- Tensor Projection: Given a tensor X, its projection is a tensor with shape $\mathsf{Shape}(\mathsf{X})|_\Gamma^x$. The projected tensor $\mathsf{X}|_\Gamma^x$ has unspecified values (unless determined by other constraints).
- Valuation Projection: Given a set of variable valuations $vars$, its projection is defined pointwise as:

$$vars|_\Gamma^x \stackrel{\text{df}}{=} \{v|_\Gamma^x \mid v \in vars\}$$

- Map List projection: Given a list of maps $\mathcal{M} = [m_1, \cdots, m_r]$, its projection is defined elementwise as:

$$\mathcal{M}|_\Gamma^x \stackrel{\text{df}}{=} [m_1|_\Gamma^x, \cdots, m_r|_\Gamma^x]$$

- Bijection Projection: Given a bijection $\mu : x \leftrightarrow y$, its projection is defined as:

$$\mu|_\Gamma^x \stackrel{\text{df}}{=} \{\Gamma(i) \in y \mid i \in x\}$$

  The resulting function $\mu|_\Gamma^x : \Gamma \leftrightarrow \Gamma(x)$ is also a bijection.
- Successive Projections: For any construct α which can be projected (map, aggregated map, tensor, valuation, bijection, etc.), successive projections are defined as:

$$\alpha|_{\Gamma_1 \cdots \Gamma_p}^{x_1 \cdots x_p} \stackrel{\text{df}}{=} \alpha|_{\Gamma_1}^{x_1} \cdots |_{\Gamma_p}^{x_p}$$

Map-List Notation. Let $\mathcal{M} = [m_1, m_2, \cdots]$ be a list of maps sharing a common domain, and let $i \in \mathsf{dom}(m_1)$ be a named-axis. We write $\mathcal{M}(i)$ for the list obtained by applying each map to i:

$$\mathcal{M}(i) \stackrel{\text{df}}{=} [m_1(i), m_2(i), \cdots]$$

Let $\Gamma \subset \{\underline{1}, \cdots, \underline{k{+}1}\}$ be a projection of size k, representing a selection of k named-axes from x^{k+1}. Note that Γ is same as x^k, and we use them interchangeably. At this point, the named-axes in Γ are unknown. As part of constructing a k-ranked counterexample from a rank $k{+}1$ counterexample, we define the k-ranked components by projecting their $(k{+}1)$-ranked counterparts using Γ:

- Tensors: $\mathsf{X}^k = \mathsf{X}^{k+1}|^x_\Gamma$,
- Variable Valuations: $vars^k = vars^{k+1}|^x_\Gamma$,
- Output Counterexample Access: $A^k = A^{k+1}|^x_\Gamma$,
- Access Map Lists: for all $i \in \{1 \cdots n\}$, $\mathcal{M}_i^k = \mathcal{M}_i^{k+1}|^x_\Gamma$, and
- Condition Map Lists: for all $j \in \{1 \cdots m\}$, $\mathcal{N}_j^k = \mathcal{N}_j^{k+1}|^x_\Gamma$

In all of the projections listed above, we omit the superscript from the aggregated-axis for brevity, i.e., $\alpha|^x_\Gamma$ as shorthand for $\alpha|^{x^{k+1}}_\Gamma$.

Note that each entry in the map lists $\mathcal{M}_i$ and $\mathcal{N}_j$ also appears within the variable valuations $vars$, so projecting them separately is consistent with projecting $vars$. At this point, the k-ranked components are also unknown.

To complete the construction of a valid k-ranked counterexample, we require that the arguments passed to scalarf in the k-ranked setting match those in the $(k{+}1)$-ranked setting. This imposes a set of *constraints* on the projection Γ. These constraints play a key role in computing a lower bound on the rank k for which counterexample lowering is valid, as described in §E.

D.1.2 Handling Generalizations. All the assumptions from §C.1 can be relaxed and the verification methodology can be extended accordingly. To do so, we simply assume a more general symbolic form of the rule, i.e., the general form of scalarf, accesses, and conditions. More concretely, suppose the rule has RClasses $c_1, \cdots, c_p$:

- Consider any RClass-rank map $m = \{c_1 \mapsto r_1, \cdots, c_p \mapsto r_p\}$ for the rewrite rule.
- For each RClass c_l, compute a rank k_l such that

  $$\forall i \geq k_l,\ \mathsf{valid}(R, m[c_l \mapsto i]) \rightarrow \mathsf{valid}(R, m[c_l \mapsto i{+}1]).$$

  Similar to the single-aggregated-axis case, we reason using the contrapositive:

  $$\neg\mathsf{valid}(R, m[c_l \mapsto k_l{+}1]) \rightarrow \neg\mathsf{valid}(R, m[c_l \mapsto k_l])$$

  This states that, given a counterexample at rank $k_l{+}1$ for all aggregated-axes of RClass c_l, we must construct a counterexample at rank k_l for all those aggregated-axes. Importantly, all aggregated-axes with the same RClass must have the same rank in any valid instantiation, so we project all such aggregated-axes together. The ranks of all other RClasses remain fixed throughout the lowering.

  Expanding the validity condition using Equation 6, we obtain:

$$\exists\, v^{k_l+1} \in vars^{k_l+1}, C^{k_l+1} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k_l+1}) \wedge \exists A^{k_l+1} \in \mathsf{Access}(\mathsf{LHS}^{k_l+1}), \neg\, \mathsf{scalarf}(\ \cdots\ )$$

$$\downarrow$$

$$\exists\, v^{k_l} \in vars^{k_l}, C^{k_l} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k_l}) \wedge \exists A^{k_l} \in \mathsf{Access}(\mathsf{LHS}^{k_l}), \neg\, \mathsf{scalarf}(\ \cdots\ ) \tag{7}$$

  We observe that the computed rank k_l is independent of the ranks of other RClasses in m.
- After computing this sufficient rank for each RClass, we obtain the RClass-rank map $\{c_1 \mapsto k_1, \cdots, c_p \mapsto k_p\}$.

- Finally, we verify the rewrite rule for all concrete instantiations within these bounds:

$$\bigwedge_{1\le r_1\le k_1} \cdots \bigwedge_{1\le r_p\le k_p} \mathsf{valid}(R, \{c_1 \mapsto r_1, \cdots, c_p \mapsto r_p\})$$

If this conjunction is a tautology, then the rewrite rule R is verified in the unbounded setting.

E Proof and Helper Lemmas

In this section, we compute a sufficient rank for each RClass in any rewrite rule. We begin under the assumptions listed in §C.1 and incrementally relax them to handle more general settings. Our approach is to reason from Equation 7: we begin with a counterexample at rank $k{+}1$ and construct a corresponding counterexample at rank k. In each case, we assume a particular form for the scalar function scalarf, the accesses, and the conditions, and then derive a sufficient rank for that form.

E.1 Sufficient Rank for Arbitrary Number of Conditions

Lemma 1. Let $R = \mathsf{LHS} \Rightarrow_C \mathsf{RHS}$ be a rewrite rule such that:

(1) R contains at most one input tensor, denoted X,
(2) X has exactly one aggregated-axis $x : c$,
(3) R does not contain any of the following operators: reduce, expand, dot, conv, conv-base, relabel, and
(4) The scalar function scalarf comprises one access to X and m conditions.

Then, the following holds:

$$\max(m, 1) \le k \implies \mathsf{valid}(R, \{c \mapsto k\}) \rightarrow \mathsf{valid}(R, \{c \mapsto k{+}1\})$$

Proof. The scalarf of R has the following canonical form, comprising one access and m conditions:

$$\mathsf{scalarf}(\mathsf{X}[x \mapsto \mathsf{fmap}(e, \mathcal{M})],\ \mathsf{fold}(g_1, \mathcal{N}_1), \cdots, \mathsf{fold}(g_m, \mathcal{N}_m))$$

where e is an index transformer and $g_1, \cdots, g_m$ are condition predicates.

Our goal is to find a sufficient rank k such that any counterexample at rank $k{+}1$ can be lowered to a counterexample at rank k. Let $x^{k+1} = \{\underline{1}, \cdots, \underline{k{+}1}\}$ denote a $(k{+}1)$-ranked instantiation of x. Let $\Gamma \subset x^{k+1}$ be a projection of size k.

Starting from Equation 7, we have:

$$\exists\, v^{k+1} \in vars^{k+1}, C^{k+1} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k+1}) \wedge \exists A^{k+1} \in \mathsf{Access}(\mathsf{LHS}^{k+1}),$$
$$\neg\mathsf{scalarf}(\mathsf{X}^{k+1}[x^{k+1} \mapsto \mathsf{fmap}(e, \mathcal{M}^{k+1})],\ \mathsf{fold}(g_1, \mathcal{N}_1^{k+1}), \cdots, \mathsf{fold}(g_m, \mathcal{N}_m^{k+1}))$$
$$\downarrow$$
$$\exists\, v^{k} \in vars^{k}, C^{k} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k}) \wedge \exists A^{k} \in \mathsf{Access}(\mathsf{LHS}^{k}),$$
$$\neg\mathsf{scalarf}(\mathsf{X}^{k}[x^{k} \mapsto \mathsf{fmap}(e, \mathcal{M}^{k})],\ \mathsf{fold}(g_1, \mathcal{N}_1^{k}), \cdots, \mathsf{fold}(g_m, \mathcal{N}_m^{k}))$$

We define the k-ranked components via projection with Γ:

- $\mathsf{X}^k = \mathsf{X}^{k+1}|^x_\Gamma$,
- $vars^k = vars^{k+1}|^x_\Gamma$,
- $A^k = A^{k+1}|^x_\Gamma$,
- $\mathcal{M}^k = \mathcal{M}^{k+1}|^x_\Gamma$, and
- for all $j \in \{1 \cdots m\}$, $\mathcal{N}_j^k = \mathcal{N}_j^{k+1}|^x_\Gamma$

Since $C^{k+1} \Rightarrow C^k$ and valid-expr(LHS^{k+1}) $\Rightarrow$ valid-expr(LHS^k), the precondition remains satisfied and the LHS expression remains valid in the k-ranked counterexample (Theorem 7). What remains is ensuring equivalence of scalarf arguments in both counterexamples and deriving constraints on Γ:

- Conditions: For all $l \in \{1 \cdots m\}$, we require that $\mathsf{fold}(g_l, \mathcal{N}_l^k)$ and $\mathsf{fold}(g_l, \mathcal{N}_l^{k+1})$ must have the same truth value. Let $\mathcal{N}_l^{k+1} = [m_1^{k+1}, m_2^{k+1}, \cdots]$. Then:

$$\mathcal{N}_l^k = \mathcal{N}_l^{k+1}|_\Gamma^x = [m_1^{k+1}|_\Gamma^x, m_2^{k+1}|_\Gamma^x, \cdots] = [m_1^k, m_2^k, \cdots]$$

  Let C_l be the set of constraints we get from this equation. We expand the definition of fold:

$$\mathsf{fold}(g_l, \mathcal{N}_l^{k+1}) = \mathsf{fold}(g_l, [m_1^{k+1}, m_2^{k+1}, \cdots]) = \bigwedge_{i=1}^{k+1} g_l(m_1^{k+1}(\underline{i}), m_2^{k+1}(\underline{i}), \cdots)$$

$$\mathsf{fold}(g_l, \mathcal{N}_l^{k}) = \mathsf{fold}(g_k, [m_1^{k}, m_2^{k}, \cdots]) = \bigwedge_{\underline{j} \in \Gamma} g_l(m_1^{k}(\underline{j}), m_2^{k}(\underline{j}), \cdots)$$

  Let $b = \mathsf{fold}(g_l, \mathcal{N}_l^{k+1})$, which contains $k{+}1$ clauses, and $c = \mathsf{fold}(g_l, \mathcal{N}_l^{k})$, which contains k clauses. The value of b is known since it depends entirely on the $(k{+}1)$-ranked counterexample. Let r be the number of clauses in b which evaluate to true. The remaining $(k{+}1) - r$ clauses evaluate to false. We perform a case analysis on r:
  - $r < k$: Then b = false. We require c = false. We can see that for any size-k projection Γ, at least one false clause will be included. Hence, c = false always. There are no constraints on Γ, i.e., $C_l = \emptyset$.
  - $r = k$: Then b = false. We require c = false. There exists exactly one $\underline{j} \in x^{k+1}$, for which the corresponding clause is false. To ensure c = false, Γ must include the named-axis $\underline{j}$. Leaving out this named-axis will lead to c being true. Therefore, $C_l = \{\underline{j}\}$.
  - $r = k{+}1$: Then b = true. We require c = true. We can see that for any size-k projection Γ, c will be true. There are no constraints on Γ, i.e., $C_l = \emptyset$.

  We get at most one constraint for each g_l, i.e., $|C_l| \leq 1$. The set of all constraints from the conditions is given by:

$$C = \bigcup_{l=1}^{m} C_l$$

- Accesses: We require the following,

$$\mathsf{X}^k[x^k \mapsto \mathsf{fmap}(e, \mathcal{M}^k)] = \mathsf{X}^{k+1}[x^{k+1} \mapsto \mathsf{fmap}(e, \mathcal{M}^{k+1})]$$

  This can always be satisfied since we can just assign the value of X^{k+1} at the access $\{x^{k+1} \mapsto \mathsf{fmap}(e, \mathcal{M}^{k+1})\}$ to $\mathsf{X}^k[x^k \mapsto \mathsf{fmap}(e, \mathcal{M}^k)]$, irrespective of the projection. Therefore, no constraints on Γ are introduced by this equation.

The final set of constraints is C. To construct a valid projection of size k, we require $|C| \leq k$. We know that $|C| \leq m$, as each condition results in at most one constraint. We also require $1 \leq k$ to avoid empty aggregated-axes. Therefore, $\max(m, 1) \leq k$ is a sufficient condition for a valid counterexample lowering.

Finally, we fully construct the k-ranked counterexample as follows:

- Projection: The named-axes in C must be a part of the projection Γ, but the other axes are unspecified. To get the final projection Γ, extend C by any $(k - |C|)$ named-axes from $\{\underline{1}, \cdots, \underline{k{+}1}\} \setminus C$.

- Construct all k-ranked components as described above using Γ. Performing a projection ensures that the tensor shapes, access maps, and attributes are valid and the precondition is satisfied in the k-ranked counterexample (Theorem 7).
- Tensor values: Let $v = \mathsf{X}^{k+1}[x^{k+1} \mapsto \mathsf{fmap}(e, \mathcal{M}^{k+1})]$. We only require X^k to have the value v at the access $\{x^k \mapsto \mathsf{fmap}(e, \mathcal{M}^k)\}$, and the values at other accesses are unspecified.

Thus, $k = \max(m, 1)$ is a sufficient rank for verifying the rewrite rule R. □

E.2 Sufficient Rank for Arbitrary Number of Accesses

Lemma 2. Let $R = \mathsf{LHS} \Rightarrow_C \mathsf{RHS}$ be a rewrite rule such that:

(1) R contains at most one input tensor, denoted X,
(2) X has exactly one aggregated-axis $x : c$,
(3) R does not contain any of the following operators: reduce, expand, dot, conv, conv-base, relabel, and
(4) The scalar function scalarf comprises n accesses to X and m conditions.

Then, the following holds:

$$\max(m + \binom{n}{2}, 1) \leq k \Rightarrow \mathsf{valid}(R, \{c \mapsto k\}) \rightarrow \mathsf{valid}(R, \{c \mapsto k{+}1\})$$

Proof. The scalarf of R has the following canonical form, comprising n accesses and m conditions:

$$\mathsf{scalarf}(\mathsf{X}[x \mapsto \mathsf{fmap}(e_1, \mathcal{M}_1)], \cdots, \mathsf{X}[x \mapsto \mathsf{fmap}(e_n, \mathcal{M}_n)], \\ \mathsf{fold}(g_1, \mathcal{N}_1), \cdots, \mathsf{fold}(g_m, \mathcal{N}_m))$$

where $e_1, \cdots, e_n$ index transformers and $g_1, \cdots, g_m$ are condition predicates.

Our goal is to find a sufficient rank k such that any counterexample at rank $k{+}1$ can be lowered to a counterexample at rank k. Let $x^{k+1} = \{\underline{1}, \cdots, \underline{k{+}1}\}$ denote a $(k{+}1)$-ranked instantiation of x. Let $\Gamma \subset x^{k+1}$ be a projection of size k.

Starting from Equation 7, we have:

$$\exists\, v^{k+1} \in vars^{k+1}, C^{k+1} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k+1}) \wedge \exists A^{k+1} \in \mathsf{Access}(\mathsf{LHS}^{k+1}), \\ \neg\mathsf{scalarf}(\mathsf{X}^{k+1}[x^{k+1} \mapsto \mathsf{fmap}(e_1, \mathcal{M}_1^{k+1})], \cdots, \mathsf{X}^{k+1}[x^{k+1} \mapsto \mathsf{fmap}(e_n, \mathcal{M}_n^{k+1})], \\ \mathsf{fold}(g_1, \mathcal{N}_1^{k+1}), \cdots, \mathsf{fold}(g_m, \mathcal{N}_m^{k+1})) \\ \downarrow \\ \exists\, v^{k} \in vars^{k}, C^{k} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k}) \wedge \exists A^{k} \in \mathsf{Access}(\mathsf{LHS}^{k}), \\ \neg\mathsf{scalarf}(\mathsf{X}^{k}[x^{k} \mapsto \mathsf{fmap}(e_1, \mathcal{M}_1^{k})], \cdots, \mathsf{X}^{k}[x^{k} \mapsto \mathsf{fmap}(e_n, \mathcal{M}_n^{k})], \\ \mathsf{fold}(g_1, \mathcal{N}_1^{k}), \cdots, \mathsf{fold}(g_m, \mathcal{N}_m^{k}))$$

We define the k-ranked components via projection with Γ:

- $\mathsf{X}^k = \mathsf{X}^{k+1}|^x_\Gamma$,
- $vars^k = vars^{k+1}|^x_\Gamma$,
- $A^k = A^{k+1}|^x_\Gamma$,
- for all $i \in \{1 \cdots n\}$, $\mathcal{M}_i^k = \mathcal{M}_i^{k+1}|^x_\Gamma$, and
- for all $j \in \{1 \cdots m\}$, $\mathcal{N}_j^k = \mathcal{N}_j^{k+1}|^x_\Gamma$

Since $C^{k+1} \Rightarrow C^k$ and $\mathsf{valid\text{-}expr}(\mathsf{LHS}^{k+1}) \Rightarrow \mathsf{valid\text{-}expr}(\mathsf{LHS}^k)$, the precondition remains satisfied and the LHS expression remains valid in the k-ranked counterexample (Theorem 7). What

remains is ensuring equivalence of scalarf arguments in both counterexamples and deriving constraints on Γ:

- Conditions: For all $l \in \{1 \cdots m\}$, we require that $\mathsf{fold}(g_l, \mathcal{N}_l^k)$ and $\mathsf{fold}(g_l, \mathcal{N}_l^{k+1})$ must have the same truth value. We assume the conditions to be independent of the accesses in the worst case. Therefore, the constraints as given by $C_1 = \bigcup_{l=1}^{m} C_l$ still hold (Lemma 1).
- Accesses: For all $l \in \{1 \cdots n\}$, we require:

$$\mathsf{X}^k[x^k \mapsto \mathsf{fmap}(e_l, \mathcal{M}_l^k)] = \mathsf{X}^{k+1}[x^{k+1} \mapsto \mathsf{fmap}(e_l, \mathcal{M}_l^{k+1})]$$

  Let v_l be the value of the tensor X^{k+1} at the access $\{x^{k+1} \mapsto \mathsf{fmap}(e_l, \mathcal{M}_l^{k+1})\}$. The projection Γ should not lead to any inconsistency with respect to the elements of X^k.
  For example, if $v_1 \neq v_2$, then these values correspond to different accesses in the tensor X^{k+1}, i.e., $\mathsf{fmap}(e_1, \mathcal{M}_1^{k+1}) \neq \mathsf{fmap}(e_2, \mathcal{M}_2^{k+1})$. Therefore, there exists some $j \in \{1 \cdots (k{+}1)\}$ such that $e_1(\mathcal{M}_1^{k+1}(\underline{j})) \neq e_2(\mathcal{M}_2^{k+1}(\underline{j}))$, i.e., the access indices are unequal.
  We require that $\mathsf{fmap}(e_1, \mathcal{M}_1^k)$ and $\mathsf{fmap}(e_2, \mathcal{M}_2^k)$ are different accesses in X^k, otherwise the same access in X^k would be assigned two different values v_1 and v_2 in the counterexample lowering. Selecting $\underline{j}$ as one of the named-axes in the projection will make sure that elements of X^k are consistent.
  Consider any arbitrary pair of values v_r and v_s, where $r, s \in \{1 \cdots n\}$ and $r \neq s$. These correspond to two accesses in X^{k+1}. There are $\binom{n}{2}$ such access pairs. Let $C_{r,s}$ be the set of constraints we get from this pair. The following cases arise:
  - $v_r = v_s$: Then any projection will result in a valid counterexample since the lowering will not lead to any inconsistency. For this case, $C_{r,s} = \emptyset$
  - $v_r \neq v_s$: Then these values correspond to different accesses in X^{k+1}, i.e.,

$$\mathsf{fmap}(e_r, \mathcal{M}_r^{k+1}) \neq \mathsf{fmap}(e_s, \mathcal{M}_s^{k+1})$$

    We require that they also correspond to different accesses in X^k, i.e.,

$$\mathsf{fmap}(e_r, \mathcal{M}_r^k) \neq \mathsf{fmap}(e_r, \mathcal{M}_s^k) \tag{8}$$

    Let $E = \{\underline{l} \in x^{k+1} \mid e_r(\mathcal{M}_r^{k+1}(\underline{l})) = e_s(\mathcal{M}_s^{k+1}(\underline{l}))\}$ denote the set of named-axes having the same index value in accesses r and s. The following cases arise:
    * $|E| < k$: Then any size-k projection will ensure that Equation 8 holds, since there exists at least one named-axis with different index values. Therefore, $C_{r,s} = \emptyset$.
    * $|E| = k$: Then there exists exactly one $\underline{l} \in x^{k+1}$ for which $e_r(\mathcal{M}_r^{k+1}(\underline{l})) \neq e_s(\mathcal{M}_s^{k+1}(\underline{l}))$. To make sure that Equation 8 holds, $\underline{l}$ needs to be in the projection. If we leave this named-axis, then the accesses r and s will be the same in the tensor X^k, but would be assigned two different values v_r and v_s. Therefore, $C_{r,s} = \{\underline{l}\}$.
    * $|E| = k{+}1$: This means that the accesses r and s are the same in X^{k+1}. This implies that $v_r = v_s$, which is a contradiction. Therefore, such a case cannot arise.

The set of all constraints from accesses is given by:

$$C_2 = \bigcup_{(r,s)} C_{r,s}$$

We know that $|C_2| \leq \binom{n}{2}$, as each access pair results in at most one constraint. The final set of constraints is $C = C_1 \cup C_2$. The set of constraints C is bounded by:

$$|C| = |C_1 \cup C_2| \leq |C_1| + |C_2| \leq m + \binom{n}{2}$$

To construct a valid projection of size k, we require $|C| \leq k$. We also require $1 \leq k$ to avoid empty aggregated-axes. Therefore, $\max(m + \binom{n}{2}, 1) \leq k$ is a sufficient condition for a valid counterexample lowering.

Finally, we fully construct the k-ranked counterexample as follows:

- Projection: The named-axes in C must be a part of the projection Γ, but the other axes are unspecified. To get the final projection Γ, extend C by any $(k - |C|)$ named-axes from $\{\underline{1}, \cdots, \underline{k+1}\} \setminus C$.
- Construct all k-ranked components as described above using Γ. Performing a projection ensures that the tensor shapes, access maps, and attributes are valid and the precondition is satisfied in the k-ranked counterexample (Theorem 7).
- Tensor values: Let $v_l = \mathsf{X}^{k+1}[x^{k+1} \mapsto \mathsf{fmap}(e_l, \mathcal{M}_l^{k+1})]$ be the value at the l^{th} access to X^{k+1}. We only require X^k to have the value v_l at the access $\{x^k \mapsto \mathsf{fmap}(e_l, \mathcal{M}_l^k)\}$, and the values at other accesses are unspecified. This should hold for all $l \in \{1 \cdots n\}$.

Thus, $k = \max(m + \binom{n}{2}, 1)$ is a sufficient rank for verifying the rewrite rule R. □

E.3 Sufficient Rank for Arbitrary Number of Aggregated-Axes

Lemma 3. Let $R = \mathsf{LHS} \Rightarrow_C \mathsf{RHS}$ be a rewrite rule such that:

(1) R contains at most one input tensor, denoted X,
(2) All aggregated-axes appearing in the rule have the same RClass, denoted c,
(3) R does not contain any of the following operators: reduce, dot, conv, conv-base, and
(4) The scalar function scalarf comprises n accesses to X and m conditions.

Then, the following holds:

$$\max(m + \binom{n}{2}, 1) \leq k \Rightarrow \mathsf{valid}(R, \{c \mapsto k\}) \rightarrow \mathsf{valid}(R, \{c \mapsto k{+}1\})$$

Proof. Let $x_1, \cdots, x_p$ denote all the aggregated-axes that appear in the rewrite rule R, each belonging to the same RClass c by assumption. Among these, let $x_1, \cdots, x_h$ be the aggregated-axes that index the input tensor X. The remaining aggregated-axes $x_{h+1}, \cdots, x_p$ occur as arguments to operators such as expand but do not index X directly. Since all aggregated-axes belong to the same class c, they are instantiated to the same rank in any valid instantiation of the rule.

The scalarf of R has the following canonical form, comprising n accesses and m conditions:

$$\begin{aligned}\mathsf{scalarf}(\mathsf{X}[x_1 \mapsto \mathsf{fmap}({}^1e_1, {}^1\mathcal{M}_1), \cdots, x_h \mapsto \mathsf{fmap}({}^he_1, {}^h\mathcal{M}_1)], \cdots,\\ \mathsf{X}[x_1 \mapsto \mathsf{fmap}({}^1e_n, {}^1\mathcal{M}_n), \cdots, x_h \mapsto \mathsf{fmap}({}^he_n, {}^h\mathcal{M}_n)],\\ \bigwedge_{j=1}^{p} \mathsf{fold}({}^jg_1, {}^j\mathcal{N}_1), \cdots, \bigwedge_{j=1}^{p} \mathsf{fold}({}^jg_m, {}^j\mathcal{N}_m))\end{aligned}$$

where each ${}^je_i$ is an index transformer and each ${}^jg_i$ is a condition predicate.

Our goal is to find a sufficient rank k for the RClass c, such that any counterexample at rank $k{+}1$ can be lowered to a counterexample at rank k. Let $x_i^{k+1} = \{\underline{1}^i, \cdots, \underline{k+1}^i\}$ denote a $(k{+}1)$-ranked instantiation of each aggregated-axis x_i. Since all aggregated-axes belong to the same RClass c, for

all $i, j \in \{1 \cdots p\}$, there exists a canonical bijection:

$$\mu_{i,j}^{k+1} = \mathsf{MapAxes}(c, x_i^{k+1}, x_j^{k+1}) : x_i^{k+1} \leftrightarrow x_j^{k+1}$$

To construct a valid counterexample, we project each aggregated-axis x_i since they all have the same RClass and are instantiated to the same rank. Let $\Gamma_i \subset x_i^{k+1}$ be a projection of size k for x_i. These projections must be pairwise disjoint and must respect the canonical bijections:

$$\forall i, j \in \{1 \cdots p\}, \quad \mu_{i,j}^{k+1}(\Gamma_i) = \Gamma_j \tag{9}$$

Starting from Equation 7, we have:

$$\begin{aligned}
&\exists\, v^{k+1} \in vars^{k+1}, C^{k+1} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k+1}) \wedge \exists A^{k+1} \in \mathsf{Access}(\mathsf{LHS}^{k+1}),\\
&\qquad \neg\mathsf{scalarf}(v_1, \cdots, v_n,\ b_1, \cdots, b_m)\\
&\qquad\qquad \downarrow\\
&\exists\, v^{k} \in vars^{k}, C^{k} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k}) \wedge \exists A^{k} \in \mathsf{Access}(\mathsf{LHS}^{k}),\\
&\qquad \neg\mathsf{scalarf}(w_1, \cdots, w_n,\ c_1, \cdots, c_m)
\end{aligned}$$

where,

- For each $l \in \{1 \cdots n\}$, v_l denotes the l^{th} access to X^{k+1} in the $(k{+}1)$-ranked counterexample:
$$v_l = \mathsf{X}^{k+1}[x_1^{k+1} \mapsto \mathsf{fmap}({}^1e_l, {}^1\mathcal{M}_l^{k+1}), \cdots, x_h^{k+1} \mapsto \mathsf{fmap}({}^he_l, {}^h\mathcal{M}_l^{k+1})]$$
- For each $l \in \{1 \cdots n\}$, w_l denotes the l^{th} access to X^{k} in the k-ranked counterexample:
$$w_l = \mathsf{X}^{k}[x_1^{k} \mapsto \mathsf{fmap}({}^1e_l, {}^1\mathcal{M}_l^{k}), \cdots, x_h^{k} \mapsto \mathsf{fmap}({}^he_l, {}^h\mathcal{M}_l^{k})]$$
- For each $l \in \{1 \cdots m\}$, b_l denotes the l^{th} condition in the $(k{+}1)$-ranked counterexample:
$$b_l = \bigwedge_{j=1}^{p} \mathsf{fold}({}^jg_l, {}^j\mathcal{N}_l^{k+1})$$
- For each $l \in \{1 \cdots m\}$, c_l denotes the l^{th} condition in the k-ranked counterexample:
$$c_l = \bigwedge_{j=1}^{p} \mathsf{fold}({}^jg_l, {}^j\mathcal{N}_l^{k})$$

We define the k-ranked components via projections with $\Gamma_1, \cdots, \Gamma_p$:

- $\mathsf{X}^k = \mathsf{X}^{k+1}|^{x_1,\cdots,x_p}_{\Gamma_1,\cdots,\Gamma_p}$,
- $vars^k = vars^{k+1}|^{x_1,\cdots,x_p}_{\Gamma_1,\cdots,x_p}$,
- $A^k = A^{k+1}|^{x_1,\cdots,x_p}_{\Gamma_1,\cdots,\Gamma_p}$,
- for all $l \in \{1 \cdots n\}, i \in \{1 \cdots h\}$, ${}^i\mathcal{M}_l^k = {}^i\mathcal{M}_l^{k+1}|^{x_i}_{\Gamma_i}$,
- for all $l \in \{1 \cdots m\}, j \in \{1 \cdots p\}$, ${}^j\mathcal{N}_l^k = {}^j\mathcal{N}_l^{k+1}|^{x_j}_{\Gamma_j}$, and
- for all $i, j \in \{1 \cdots p\}$, the canonical bijection $\mu_{i,j}^k = \mathsf{MapAxes}(c, \Gamma_i, \Gamma_j)$ is simply $\mu_{i,j}^{k+1}|^{x_i}_{\Gamma_i}$. The resulting canonical bijections must satisfy:
  - $\mu_{i,j}^k \circ \mu_{j,i}^k = \mathsf{id}$, and
  - $\mu_{j,l}^k \circ \mu_{i,j}^k = \mu_{i,l}^k$.

  To ensure that these properties hold, we must ensure that Equation 9 is satisfied.

Since $C^{k+1} \Rightarrow C^k$ and valid-expr(LHS^{k+1}) $\Rightarrow$ valid-expr(LHS^k), the precondition remains satisfied and the LHS expression remains valid in the k-ranked counterexample (Theorem 7). What remains is ensuring equivalence of scalarf arguments in both counterexamples and deriving constraints on $\Gamma_1, \cdots, \Gamma_p$:

- Conditions: For all $l \in \{1 \cdots m\}$, $c_l = \bigwedge_{j=1}^{p} \mathsf{fold}({}^j g_l, {}^j \mathcal{N}_l^k)$ and $b_l = \bigwedge_{j=1}^{p} \mathsf{fold}({}^j g_l, {}^j \mathcal{N}_l^{k+1})$ must have the same truth value. Let ${}^1C_l, \cdots, {}^pC_l$ be the set of constraints we get on $\Gamma_1, \cdots, \Gamma_p$ from this equation, respectively.
  The value of b_l is known since it depends entirely on the $(k+1)$-ranked counterexample. Both b_l and c_l contain p folds. Let r be the number of folds in b_l that evaluate to true. The remaining $p - r$ folds evaluate to false. We perform a case analysis on r:
  - $r = p$: Then b_l = true. We require c_l = true. We can see that for any size-k projections $\Gamma_1, \cdots, \Gamma_p$, all folds in c_l will be true. Therefore, ${}^1C_l = \cdots = {}^pC_l = \emptyset$.
  - $r < p$: Then b_l = false. We require c_l = false. This implies that there exists some aggregated-axis, say x_i^{k+1}, whose fold evaluates to false in b_l, i.e., $\mathsf{fold}({}^i g_l, {}^i \mathcal{N}_l^{k+1})$ = false. We ensure that the fold of x_i^k, i.e., $\mathsf{fold}({}^i g_l, {}^i \mathcal{N}_l^k)$ also evaluates to false, resulting in c_l = false. This is similar to Lemma 1, and we get at most one constraint on Γ_i in ${}^iC_l$ and no constraints on other projections. Therefore, $|{}^iC_l| \leq 1$ and ${}^jC_l = \emptyset$ for all $j \neq i$.

  Each condition contributes at most one constraint to exactly one projection. The set of all constraints through the conditions is given by, where ${}^iC$ denotes the constraints on Γ_i:

$$ {}^1C = \bigcup_{l=1}^{m} {}^1C_l, \ \cdots, \ {}^pC = \bigcup_{l=1}^{m} {}^pC_l $$

  Since we get at most m constraints partitioned into these p sets, we conclude that the total number of constraints is bounded by:

$$ |{}^1C| + \cdots + |{}^pC| \leq m $$

- Accesses: For all $l \in \{1 \cdots n\}$, w_l must be the same as v_l. The projections $\Gamma_1, \cdots, \Gamma_p$ should not lead to inconsistencies with respect to the elements of X^k.
  Consider any arbitrary pair of values v_r and v_s, where $r, s \in \{1 \cdots n\}$ and $r \neq s$. These correspond to two accesses to X^{k+1}. There are $\binom{n}{2}$ such access pairs. Let ${}^1D_{r,s}, \cdots, {}^hD_{r,s}$ be the constraints we get on $\Gamma_1, \cdots, \Gamma_h$ from this equation, respectively. We exclude ${}^{h+1}D_{r,s}, \cdots, {}^pD_{r,s}$ from the analysis because the corresponding aggregated-axes do not index the tensor X, and hence do not introduce any inconsistencies. Thus, their constraints are equal to $\emptyset$. The following cases arise:
  - $v_r = v_s$: Then any projection will result in a valid counterexample since the lowering will not lead to any inconsistency. For this case, ${}^iD_{r,s} = \emptyset$ for all i.
  - $v_r \neq v_s$: Then these values correspond to different accesses in X^{k+1}, i.e.,

$$ \{x_1^{k+1} \mapsto \mathsf{fmap}({}^1e_r, {}^1\mathcal{M}_r^{k+1}), \cdots, x_h^{k+1} \mapsto \mathsf{fmap}({}^he_r, {}^h\mathcal{M}_r^{k+1})\} \neq \{x_1^{k+1} \mapsto \mathsf{fmap}({}^1e_s, {}^1\mathcal{M}_s^{k+1}), \cdots, x_h^{k+1} \mapsto \mathsf{fmap}({}^he_s, {}^h\mathcal{M}_s^{k+1})\} $$

    We require that they also correspond to different accesses in X^k, i.e.,

$$ \{x_1^{k} \mapsto \mathsf{fmap}({}^1e_r, {}^1\mathcal{M}_r^{k}), \cdots, x_h^{k} \mapsto \mathsf{fmap}({}^he_r, {}^h\mathcal{M}_r^{k})\} \neq \{x_1^{k} \mapsto \mathsf{fmap}({}^1e_s, {}^1\mathcal{M}_s^{k}), \cdots, x_h^{k} \mapsto \mathsf{fmap}({}^he_s, {}^h\mathcal{M}_s^{k})\} \quad (10) $$

Since the accesses do not match in X^{k+1}, there exists some aggregated-axis, say x_i^{k+1}, for $i \in \{1 \cdots h\}$, such that its corresponding access maps in the two accesses are unequal:

$$\mathsf{fmap}({}^{i}e_r, {}^{i}\mathcal{M}_r^{k+1}) \neq \mathsf{fmap}({}^{i}e_s, {}^{i}\mathcal{M}_s^{k+1})$$

We ensure that the access maps of x_i^k are unequal in the two accesses, to ensure that Equation 10 holds:

$$\mathsf{fmap}({}^{i}e_r, {}^{i}\mathcal{M}_r^{k}) \neq \mathsf{fmap}({}^{i}e_s, {}^{i}\mathcal{M}_s^{k})$$

This is similar to Lemma 2, and we get at most one constraint on Γ_i in ${}^{i}D_{r,s}$ and no constraints on other projections. Therefore, $|{}^{i}D_{r,s}| \leq 1$ and ${}^{j}D_{r,s} = \emptyset$ for all $j \neq i$.

Each access pair contributes at most one constraint to exactly one projection. The set of all constraints through the accesses is given by, where ${}^{i}D$ denotes the constraints on Γ_i:

$${}^{1}D = \bigcup_{(r,s)} {}^{1}D_{r,s}, \ \cdots, {}^{p}D = \bigcup_{(r,s)} {}^{p}D_{r,s}$$

Since we get at most $\binom{n}{2}$ constraints partitioned into these p sets, we conclude that the total number of constraints is bounded by:

$$|{}^{1}D| + \ \cdots \ + |{}^{p}D| \leq \binom{n}{2}$$

We now define the combined set of constraints (from both conditions and accesses) on each projection Γ_i as $F_i = {}^{i}C \cup {}^{i}D$. The total number of constraints across all projections is bounded by:

$$\begin{aligned} |F_1| + \ \cdots \ + |F_p| &= |{}^{1}C \cup {}^{1}D| + \ \cdots \ + |{}^{p}C \cup {}^{p}D| \\ &\leq |{}^{1}C| + |{}^{1}D| + \ \cdots \ + |{}^{p}C| + |{}^{p}D| \\ &= |{}^{1}C| + \ \cdots \ + |{}^{p}C| + |{}^{1}D| + \ \cdots \ + |{}^{p}D| \\ &\leq m + \binom{n}{2} \end{aligned}$$

Thus, the total number of constraints is at most $m + \binom{n}{2}$, partitioned across the p constraint sets. This implies that $\max_i |F_i| \leq m + \binom{n}{2}$ (the maximum possible constraints on any projection). For all the projections to be valid, we require $|F_1| \leq k, \cdots, |F_p| \leq k$, or equivalently, $\max_i |F_i| \leq k$. We also need $1 \leq k$ to avoid empty aggregated-axes. Therefore, $\max(m + \binom{n}{2}, 1) \leq k$ is a sufficient condition for a valid counterexample lowering.

Finally, we fully construct the k-ranked counterexample as follows:

- Projection: For all $j \in \{1 \cdots p\}$, the named-axes in F_j must be a part of the projection Γ_j. Additionally, the projections should respect the canonical bijections of the $(k+1)$-ranked counterexample. We compute the final projections using the ComputeProjections routine described in Algorithm 1. It takes the final constraints $F_1, \cdots, F_p$ as input and returns the projections $\Gamma_1, \cdots, \Gamma_p$. In lines 4-9, we expand each F_j using the canonical bijection images of other constraint sets.
  After line 9, the sets $F'_1, \cdots, F'_p$ are of the same size. In fact, there exists a bijection between all the sets: the bijection between F'_i and F'_j is simply $\mu_{i,j}^{k+1}|_{F'_i}^{x_i}$. This ensures that all F'_i contain the exact same number of elements, since their elements are simply relabelings of each other via the bijections.

Algorithm 1: ComputeProjections

Input: Constraint sets $F_1, \cdots, F_p$
Output: Final projections $\Gamma_1, \cdots, \Gamma_p$

1 **for** $i \in \{1 \cdots p\}$ **do**
2 $\quad F'_i \leftarrow F_i$;
3 **end**
4 **for** $j \in \{1 \cdots p\}$ **do**
5 $\quad$ **for** $i \in \{1 \cdots p\}, i \neq j$ **do**
6 $\quad\quad \Delta_{i,j} \leftarrow \mu_{i,j}^{k+1}(F_i)$;
7 $\quad\quad F'_j \leftarrow F'_j \cup \Delta_{i,j}$;
8 $\quad$ **end**
9 **end**
10 $R \leftarrow$ Choice$(x_1^{k+1} \setminus F'_1,\ k - |F'_1|)$;
11 **for** $i \in \{1 \cdots p\}$ **do**
12 $\quad \Gamma_i \leftarrow F'_i \cup \mu_{1,i}^{k+1}(R)$;
13 **end**
14 **return** $\Gamma_1, \cdots, \Gamma_p$

We also prove that the size of the sets $F'_1, \cdots, F'_p$ is bounded by k, the projection size. For all i, j, $|\Delta_{i,j}| = |F_i|$ since $\mu_{i,k}^{k+1}$ is a bijection. For all $j \in \{1 \cdots p\}$, the set F'_j is $F_j \cup \bigcup_{i=1, i\neq j}^{p} \Delta_{i,j}$, whose size is bounded by:

$$
\begin{aligned}
|F'_j| &= \left| F_j \cup \bigcup_{i=1, i\neq j}^{p} \Delta_{i,j} \right| \\
&\leq |F_j| + \sum_{i=1, i\neq j}^{p} |\Delta_{i,j}| \\
&= |F_j| + \sum_{i=1, i\neq j}^{p} |F_i| = \sum_{i=1}^{p} |F_i| \leq m + \binom{n}{2} \leq k
\end{aligned}
$$

Therefore, the number of constraints in each set does not exceed the projection size k. On line 10, we add any $(k - |F'_1|)$ named-axes from $x_1^{k+1} \setminus F'_1$ to complete the projection Γ_1. In lines 11-13, we add their bijection images to complete the other projections.

- Construct all k-ranked components as described above using Γ. Performing a projection ensures that the tensor shapes, access maps, and attributes are valid and the precondition is satisfied in the k-ranked counterexample (Theorem 7).
- Tensor values: For all $i \in \{1 \cdots n\}$, we require w_i, the value of X^k at the i^{th} access, to be same as v_i, the value of X^{k+1} at the i^{th} access. The values at other accesses in X^k are unspecified.

Thus, $k = \max(m + \binom{n}{2}, 1)$ is a sufficient rank for verifying the rewrite rule R.

□

E.4 Arbitrary Number of RClasses

Lemma 4. Let $R = \text{LHS} \Rightarrow_C \text{RHS}$ be a rewrite rule such that

(1) R contains at most one input tensor, denoted X,
(2) R does not contain any of the following operators: reduce, dot, conv, and conv-base,
(3) M is an RClass-rank map for R and c is an RClass appearing in the rule,
(4) The scalar function scalarf comprises: (1) n accesses to X and (2) m conditions.

Let $n_{\mathsf{X}} = n$ if an aggregated-axis of X has RClass c, and $n_{\mathsf{X}} = 0$ otherwise. Then the following holds:

$$\max(m + \binom{n_{\mathsf{X}}}{2}, 1) \leq k \Rightarrow \mathsf{valid}(R, M[c \mapsto k]) \rightarrow \mathsf{valid}(R, M[c \mapsto k{+}1])$$

Proof. Let $x_1, \cdots, x_p$ denote all the aggregated-axes of RClass c that appear in the rewrite rule R . Among these, let $x_1, \cdots, x_h$ be the aggregated-axes that index the input tensor X. The aggregated-axes $x_{h+1}, \cdots, x_p$ occur as arguments to operators such as expand but do not index X directly. All other aggregated-axes in R belong to RClasses other than c. The scalarf of R has the following canonical form, comprising n_{X} relevant accesses to X and m conditions:

$$\begin{aligned}\mathsf{scalarf}(\mathsf{X}[x_1 \mapsto \mathsf{fmap}({}^1e_1, {}^1\mathcal{M}_1), \cdots, x_h \mapsto \mathsf{fmap}({}^he_1, {}^h\mathcal{M}_1) \cup Y_1], \cdots,\\ \mathsf{X}[x_1 \mapsto \mathsf{fmap}({}^1e_{n_{\mathsf{X}}}, {}^1\mathcal{M}_{n_{\mathsf{X}}}), \cdots, x_h \mapsto \mathsf{fmap}({}^he_{n_{\mathsf{X}}}, {}^h\mathcal{M}_{n_{\mathsf{X}}}) \cup Y_{n_{\mathsf{X}}}],\\ \phi_1 \wedge \bigwedge_{j=1}^{p} \mathsf{fold}({}^jg_1, {}^j\mathcal{N}_1), \cdots, \phi_m \wedge \bigwedge_{j=1}^{p} \mathsf{fold}({}^jg_m, {}^j\mathcal{N}_m))\end{aligned}$$

where each ${}^je_i$ is an index transformer and each ${}^jg_i$ is a condition predicate. The maps Y_i denote the access maps for the remaining aggregated-axes (those whose RClass is not c), while formulas ϕ_i collect the fold-conditions for these remaining aggregated-axes.

Our goal is to find a sufficient rank k for the RClass c, such that any counterexample at rank $k{+}1$ can be lowered to a counterexample at rank k. Let $x_i^{k+1} = \{\underline{1}^i, \cdots, \underline{k{+}1}^i\}$ denote a $(k{+}1)$-ranked instantiation of each aggregated-axis x_i. Since all aggregated-axes $x_1, \cdots, x_p$ belong to the same RClass c, there exists a canonical bijection between any pair:

$$\mu_{i,j}^{k+1} = \mathsf{MapAxes}(c, x_i^{k+1}, x_j^{k+1}) : x_i^{k+1} \leftrightarrow x_j^{k+1}$$

To construct a valid counterexample, we project each aggregated-axis x_i since they all have the same RClass and are instantiated to the same rank. Let $\Gamma_i \subset x_i^{k+1}$ be a projection of size k for x_i. These projections must be pairwise disjoint and must respect the canonical bijections:

$$\forall i, j \in \{1 \cdots p\}, \quad \mu_{i,j}^{k+1}(\Gamma_i) = \Gamma_j \tag{11}$$

Starting from Equation 7, we have:

$$\begin{aligned}&\exists\, v^{k+1} \in \mathit{vars}^{k+1}, C^{k+1} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k+1}) \wedge \exists A^{k+1} \in \mathsf{Access}(\mathsf{LHS}^{k+1}),\\ &\qquad \neg\mathsf{scalarf}(v_1, \cdots, v_{n_{\mathsf{X}}}, b_1, \cdots, b_m)\\ &\qquad\qquad \downarrow\\ &\exists\, v^{k} \in \mathit{vars}^{k}, C^{k} \wedge \mathsf{valid\text{-}expr}(\mathsf{LHS}^{k}) \wedge \exists A^{k} \in \mathsf{Access}(\mathsf{LHS}^{k}),\\ &\qquad \neg\mathsf{scalarf}(w_1, \cdots, w_{n_{\mathsf{X}}}, c_1, \cdots, c_m)\end{aligned}$$

where,

- For each $l \in \{1 \cdots n_{\mathsf{X}}\}$, v_l denotes the l^{th} access to X^{k+1} in the $(k{+}1)$-ranked counterexample:
$$v_l = \mathsf{X}^{k+1}[x_1^{k+1} \mapsto \mathsf{fmap}({}^1e_l, {}^1\mathcal{M}_l^{k+1}), \cdots, x_h^{k+1} \mapsto \mathsf{fmap}({}^he_l, {}^h\mathcal{M}_l^{k+1}) \cup Y_l^{k+1}]$$
- For all $l \in \{1 \cdots n_{\mathsf{X}}\}$, w_l denotes the l^{th} access to X^k in the k-ranked counterexample:
$$w_l = \mathsf{X}^k[x_1^k \mapsto \mathsf{fmap}({}^1e_l, {}^1\mathcal{M}_l^k), \cdots, x_h^k \mapsto \mathsf{fmap}({}^he_l, {}^h\mathcal{M}_l^k) \cup Y_l^k]$$

- For each $l \in \{1 \cdots m\}$, b_l denotes the l^{th} condition in the $(k+1)$-ranked counterexample:

$$b_l = \phi_l^{k+1} \wedge \bigwedge_{j=1}^{p} \mathsf{fold}({}^j g_l, {}^j \mathcal{N}_l^{k+1})$$

- For all $l \in \{1 \cdots m\}$, c_l denotes the l^{th} condition in the k-ranked counterexample:

$$c_l = \phi_l^{k} \wedge \bigwedge_{j=1}^{p} \mathsf{fold}({}^j g_l, {}^j \mathcal{N}_l^{k})$$

We define the k-ranked components via projections with $\Gamma_1, \cdots, \Gamma_p$:

- $\mathsf{X}^k = \mathsf{X}^{k+1}|^{x_1,\cdots,x_p}_{\Gamma_1,\cdots,\Gamma_p}$,
- $vars^k = vars^{k+1}|^{x_1,\cdots,x_p}_{\Gamma_1,\cdots,x_p}$,
- $A^k = A^{k+1}|^{x_1,\cdots,x_p}_{\Gamma_1,\cdots,\Gamma_p}$,
- for all $l \in \{1 \cdots n_{\mathsf{X}}\}, i \in \{1 \cdots h\}$, ${}^i\mathcal{M}_l^k = {}^i\mathcal{M}_l^{k+1}|^{x_i}_{\Gamma_i}$,
- for all $l \in \{1 \cdots n_{\mathsf{X}}\}$, $Y_l^k = Y_l^{k+1}|^{x_1,\cdots,x_p}_{\Gamma_1,\cdots,\Gamma_p} = Y_l^{k+1}$, since Y_l is independent of the RClass c,
- for all $l \in \{1 \cdots m\}, j \in \{1 \cdots p\}$, ${}^j\mathcal{N}_l^k = {}^j\mathcal{N}_l^{k+1}|^{x_j}_{\Gamma_j}$,
- for all $l \in \{1 \cdots m\}$, $\phi_l^k = \phi_l^{k+1}|^{x_1,\cdots,x_p}_{\Gamma_1,\cdots,\Gamma_p} = \phi_l^{k+1}$, since ϕ_l is independent of the RClass c, and
- for all $i, j \in \{1 \cdots p\}$, the canonical bijection $\mu_{i,j}^k = \mathsf{MapAxes}(c, \Gamma_i, \Gamma_j)$ is simply $\mu_{i,j}^{k+1}|^{x_i}_{\Gamma_i}$. The resulting canonical bijections must satisfy:
  - $\mu_{i,j}^k \circ \mu_{j,i}^k = \mathsf{id}$, and
  - $\mu_{j,l}^k \circ \mu_{i,j}^k = \mu_{i,l}^k$.

  To ensure that these properties hold, we must ensure that Equation 11 is satisfied.

To obtain the k-ranked counterexample, we project each component of the $(k+1)$-ranked counterexample along the projections Γ_i. Components not involving the RClass c are unchanged.

Since $C^{k+1} \Rightarrow C^k$ and valid-expr$(\mathsf{LHS}^{k+1}) \Rightarrow$ valid-expr(LHS^k), the precondition remains satisfied and the LHS expression remains valid in the k-ranked counterexample (Theorem 7). What remains is ensuring equivalence of scalarf arguments in both counterexamples and deriving constraints on $\Gamma_1, \cdots, \Gamma_p$:

- Conditions: For all $l \in \{1 \cdots m\}$, we require $c_l = \phi_l^k \wedge \bigwedge_{j=1}^{p} \mathsf{fold}({}^j g_l, {}^j \mathcal{N}_l^k)$ and $b_l = \phi_l^{k+1} \wedge \bigwedge_{j=1}^{p} \mathsf{fold}({}^j g_l, {}^j \mathcal{N}_l^{k+1})$ must have the same truth value.
  The value of b_l is known since it depends entirely on the $(k+1)$-ranked counterexample. To make sure that c_l and b_l have the same truth value, it is sufficient to make the truth values of $c_l' = \bigwedge_{j=1}^{p} \mathsf{fold}({}^j g_l, {}^j \mathcal{N}_l^k)$ and $b_l' = \bigwedge_{j=1}^{p} \mathsf{fold}({}^j g_l, {}^j \mathcal{N}_l^{k+1})$ the same since $\phi_l^k = \phi_l^{k+1}$. This is similar to Lemma 3, and we get at most one constraint on exactly one projection.
  The set of all constraints through the conditions is given by, where ${}^iC$ denotes the constraints on Γ_i. Since we get at most m constraints partitioned into these p sets, we conclude that the total number of constraints is bounded by:

$$|{}^1C| + \cdots + |{}^pC| \leq m$$

- Accesses: For all $l \in \{1 \cdots n_{\mathsf{X}}\}$, w_l must be the same as v_l. The projections $\Gamma_1, \cdots, \Gamma_p$ should not lead to inconsistencies with respect to the elements of X^k.
  Consider any arbitrary pair of values v_r and v_s, where $r, s \in \{1 \cdots n_{\mathsf{X}}\}$ and $r \neq s$. These correspond to two accesses to X^{k+1}. There are $\binom{n_{\mathsf{X}}}{2}$ such access pairs. Let ${}^1D_{r,s}, \cdots, {}^hD_{r,s}$ be the constraints we get on $\Gamma_1, \cdots, \Gamma_h$ from this equation, respectively. We exclude ${}^{h+1}D_{r,s}, \cdots, {}^pD_{r,s}$

from the analysis because the corresponding aggregated-axes do not index the tensor X, and hence do not introduce any inconsistencies. Thus, their constraints are equal to $\emptyset$. The following cases arise:

- $v_r = v_s$: Then any projection will result in a valid counterexample since the lowering will not lead to any inconsistency. For this case, ${}^iD_{r,s} = \emptyset$ for all i.
- $v_r \neq v_s$: Then these values correspond to different accesses in X^{k+1}, i.e.,

$$\{x_1^{k+1} \mapsto \mathsf{fmap}({}^1e_r, {}^1\mathcal{M}_r^{k+1}), \cdots, x_h^{k+1} \mapsto \mathsf{fmap}({}^he_r, {}^h\mathcal{M}_r^{k+1})\} \cup Y_r^{k+1} \neq \{x_1^{k+1} \mapsto \mathsf{fmap}({}^1e_s, {}^1\mathcal{M}_s^{k+1}), \cdots, x_h^{k+1} \mapsto \mathsf{fmap}({}^he_s, {}^h\mathcal{M}_s^{k+1})\} \cup Y_s^{k+1}$$

We require that they also correspond to different accesses in X^k, i.e.,

$$\{x_1^{k} \mapsto \mathsf{fmap}({}^1e_r, {}^1\mathcal{M}_r^{k}), \cdots, x_h^{k} \mapsto \mathsf{fmap}({}^he_r, {}^h\mathcal{M}_r^{k})\} \cup Y_r^{k} \neq \{x_1^{k} \mapsto \mathsf{fmap}({}^1e_s, {}^1\mathcal{M}_s^{k}), \cdots, x_h^{k} \mapsto \mathsf{fmap}({}^he_s, {}^h\mathcal{M}_s^{k})\} \cup Y_s^{k} \tag{12}$$

Since the accesses do not match in X^{k+1}, two cases arise: (1) $Y_r^{k+1} \neq Y_s^{k+1}$. This implies $Y_r^k \neq Y_s^k$, ensuring that Equation 12 holds. This case results in no constraints on any projection, (2) there exists some aggregated-axis, say x_i^{k+1}, for $i \in \{1 \cdots h\}$, such that its corresponding access maps in the two accesses are unequal:

$$\mathsf{fmap}({}^ie_r, {}^i\mathcal{M}_r^{k+1}) \neq \mathsf{fmap}({}^ie_s, {}^i\mathcal{M}_s^{k+1})$$

We ensure that the access maps of x_i^k are unequal in the two accesses, to ensure that Equation 12 holds:

$$\mathsf{fmap}({}^ie_r, {}^i\mathcal{M}_r^{k}) \neq \mathsf{fmap}({}^ie_s, {}^i\mathcal{M}_s^{k})$$

This is similar to Lemma 2, and we get at most one constraint on Γ_i in ${}^iD_{r,s}$ and no constraints on other projections. Therefore, $|{}^iD_{r,s}| \leq 1$ and ${}^jD_{r,s} = \emptyset$ for all $j \neq i$.

Each access pair contributes at most one constraint to exactly one projection. The set of all constraints through the accesses is given by, where ${}^iD$ denotes the constraints on Γ_i:

$${}^1D = \bigcup_{(r,s)} {}^1D_{r,s}, \ \cdots, {}^pD = \bigcup_{(r,s)} {}^pD_{r,s}$$

Since we get at most $\binom{n_\mathsf{X}}{2}$ constraints partitioned into these p sets, we conclude that the total number of constraints is bounded by:

$$|{}^1D| + \ \cdots \ + |{}^pD| \leq \binom{n_\mathsf{X}}{2}$$

We now define the combined set of constraints (from both conditions and accesses) on each projection Γ_i as $F_i = {}^iC \cup {}^iD$. The total number of constraints across all projections is bounded by:

$$\begin{aligned} |F_1| + \ \cdots \ + |F_p| &= |{}^1C \cup {}^1D| + \ \cdots \ + |{}^pC \cup {}^pD| \\ &\leq |{}^1C| + |{}^1D| + \ \cdots \ + |{}^pC| + |{}^pD| \\ &= |{}^1C| + \ \cdots \ + |{}^pC| + |{}^1D| + \ \cdots \ + |{}^pD| \\ &\leq m + \binom{n_\mathsf{X}}{2} \end{aligned}$$

Similar to Lemma 3, we get $\max(m + \binom{n_\mathsf{X}}{2}, 1) \leq k$ as a sufficient condition for a valid counterexample lowering. Finally, we fully construct the k-ranked counterexample as follows:

- Projection: For all $j \in \{1 \cdots p\}$, the named-axes in F_j must be a part of the projection Γ_j. Additionally, the projections should respect the canonical bijections of the $(k{+}1)$-ranked counterexample. We compute the final projections using the ComputeProjections routine described in Algorithm 1.
- Construct all k-ranked components as described above using Γ. Performing a projection ensures that the tensor shapes, access maps, and attributes are valid and the precondition is satisfied in the k-ranked counterexample (Theorem 7).
- Tensor values: For all $i \in \{1 \cdots n_{\mathsf{X}}\}$, we require w_i, the value of X^k at the i^{th} access, to be same as v_i, the value of X^{k+1} at the i^{th} access. The values at other accesses in X^k are unspecified.

Thus, $k = \max(m + \binom{n_{\mathsf{X}}}{2}, 1)$ is a sufficient rank for the RClass c.

□

E.5 Arbitrary Number of Tensors

Lemma 5. Let $R = \text{LHS} \Rightarrow_C \text{RHS}$ be a rewrite rule such that

(1) R does not contain any of the following operators: reduce, dot, conv, and conv-base,
(2) M is an RClass-rank map for R and c is an RClass appearing in the rule,
(3) R contains q tensors $\mathsf{X}_1, \cdots, \mathsf{X}_q$ that have an aggregated-axis belonging to the RClass c,
(3) The scalar function scalarf comprises: (1) n_{X_i} accesses to X_i and (2) m conditions.

Then the following holds:

$$\max(m + \sum_{i=1}^{q} \binom{n_{\mathsf{X}_i}}{2}, 1) \leq k \Rightarrow \text{valid}(R, M[c \mapsto k]) \rightarrow \text{valid}(R, M[c \mapsto k{+}1])$$

Proof. Only tensors that contain an aggregated-axis belonging to the RClass c can introduce constraints that on the sufficient rank for c. Accesses to any other tensors retain their values during the counterexample lowering, and hence do not contribute to the bound. For each tensor X_i that contains an aggregated-axis with the RClass c, every pair of its accesses can introduce at most one constraint to exactly one projection. Therefore, the total number of constraints through accesses of X_i is bounded by $\binom{n_{\mathsf{X}_i}}{2}$.

The contributions from different tensors are summed independently because accesses across tensors do not lead to any constraints (we assume that distinct tensors do not alias; more §G). Consequently, the total number of constraints is simply bounded by sum of per-tensor contributions, together with the m constraints from the conditions in the worst case. The counterexample construction is similar to Lemma 4. We introduce a routine TensorsWithRClass which takes an RClass c and returns all the input tensors which have an aggregated-axis belonging to c. For each tensor, we use the NumTensorAccess routine to get the number of distinct accesses to that tensor.

Thus, $k = \max(m + \sum_{i=1}^{q} \binom{n_{\mathsf{X}_i}}{2}, 1)$ is a sufficient rank for the RClass c to verify the rewrite rule R, where $\{\mathsf{X}_1, \cdots, \mathsf{X}_q\} = \text{TensorsWithRClass}(c)$ and $n_{\mathsf{X}_i} = \text{NumTensorAccess}(\mathsf{X}_i)$. □

F Preconditions

In this section, we prove that if the precondition holds in the $(k{+}1)$-ranked counterexample, then the precondition holds in the k-ranked counterexample, i.e., $C^{k+1} \Rightarrow C^k$.

F.1 A Language for Preconditions

We first define a simple language for preconditions and a useful substitution lemma.

Definition 5 (Precondition language)**.** Let $\mathcal{A}$ be a set of atomic propositions (atoms). We define a language $\Phi_{\wedge,\vee}$ by the grammar:

$$\phi \coloneqq a \in \mathcal{A} \mid \phi \wedge \phi \mid \phi \vee \phi$$

Similarly, the language $\Phi_{\wedge}$ is defined by the grammar:

$$\phi \coloneqq a \in \mathcal{A} \mid \phi \wedge \phi$$

Note that $\Phi_{\wedge} \subseteq \Phi_{\wedge,\vee}$.

We write $\phi\{b/a\}$ for the formula obtained from ϕ by simultaneously replacing every occurrence of atom a by atom b (also called a *substitution*).

Lemma 6 (Substitution Monotonicity)**.** Let $\phi \in \Phi_{\wedge,\vee}$ and let $a, b \in \mathcal{A}$ be atoms. If the implication $a \Rightarrow b$ holds, then

$$\phi \Rightarrow \phi\{b/a\}$$

The same holds for $\Phi_{\wedge}$.

Proof. By structural induction on ϕ.
Basis. If $\phi = a$, then $\phi\{b/a\} = b$, and the claim follows from the hypothesis $a \Rightarrow b$. If $\phi = c$ for $c \neq a$, then $\phi\{b/a\} = c$ and $c \Rightarrow c$ holds trivially.
Inductive step.

- If $\phi = \phi_1 \wedge \phi_2$, then by the induction hypothesis, $\phi_i \Rightarrow \phi_i\{b/a\}$ for $i = 1, 2$. Since $\phi\{b/a\} = \phi_1\{b/a\} \wedge \phi_2\{b/a\}$ and $\phi_1 \wedge \phi_2 \Rightarrow \phi_1\{b/a\} \wedge \phi_2\{b/a\}$, we obtain the desired implication.
- If $\phi = \phi_1 \vee \phi_2$, similarly the induction hypothesis yields $\phi_1 \vee \phi_2 \Rightarrow \phi_1\{b/a\} \vee \phi_2\{b/a\}$.

This completes the induction proof. □

F.2 Conditions in TensorRight

The preconditions in TensorRight are drawn from the language $\Phi_{\wedge,\vee}$ (Definition 5), whose atoms $\mathcal{A}$ consist precisely of all fold-conditions. The expression validity conditions (such as those returned by valid-expr) are drawn from $\Phi_{\wedge}$.

The fold construct takes a predicate and applies it pointwise to a list of maps with the same domains. It returns true if all predicate values are true, and false otherwise. It is defined as:

$$\mathsf{fold}(g, [m_1, m_2, \cdots]) = \bigwedge_{i \in \mathsf{dom}(m_1)} g(m_1(i), m_2(i), \cdots)$$

Theorem 7. Let $C^{k+1}, C^k \in \Phi_{\wedge,\vee}$ be the preconditions obtained from the same symbolic precondition C by instantiating every fold over a RClass c at rank $k{+}1$ and at rank k, respectively. During the counterexample construction, each $(k{+}1)$-fold is projected to the corresponding k-fold. Then,

$$C^{k+1} \Rightarrow C^k$$

Similarly, for any tensor expression e, $\mathsf{valid\text{-}expr}(e^{k+1}) \Rightarrow \mathsf{valid\text{-}expr}(e^k)$.

Proof. It suffices to show the implication for each atomic fold and then lift it to the whole precondition using Lemma 6.

Let $f^{k+1} = \mathsf{fold}(g, \mathcal{N}^{k+1})$ be one of the atomic folds in C^{k+1}, defined over an aggregated-axis in RClass c. Let $\mathcal{N}_l^{k+1} = [m_1^{k+1}, m_2^{k+1}, \cdots]$. It has the form:

$$\mathsf{fold}(g, \mathcal{N}^{k+1}) = \bigwedge_{i \in \mathsf{dom}(m_1)}^{k+1} g(m_1^{k+1}(i), m_2^{k+1}(i), \cdots)$$

Let Γ be the chosen projection of size k and let $\mathcal{N}^k = \mathcal{N}^{k+1}|^x_\Gamma$ denote the projected maps. Then:

$$\mathcal{N}^k_l = \mathcal{N}^{k+1}_l|^x_\Gamma = [m^{k+1}_1|^x_\Gamma, m^{k+1}_2|^x_\Gamma, \cdots] = [m^k_1, m^k_2, \cdots]$$

The corresponding fold $f^k = \mathsf{fold}(g, \mathcal{N}^k)$ in C^k has the form:

$$\mathsf{fold}(g, \mathcal{N}^k) = \bigwedge_{j \in \Gamma} g(m^k_1(j), m^k_2(j), \cdots)$$

Since f^{k+1} is a conjunction over a superset of indices, if the larger conjunction if true, then sub-conjunction is true, i.e.:

$$f^{k+1} \Rightarrow f^k$$

Thus every $(k{+}1)$-fold atom implies its projected k-fold atom. Applying Lemma 6 repeatedly to substitute each $(k{+}1)$-fold atom by its projected counterpart in the whole precondition implies $C^{k+1} \Rightarrow C^k$, as required. The same holds for validity conditions returned by valid-expr. □

G Discussion

Rewrite Rules with Reductions. So far, we have assumed that rewrite rules do not contain any of the following operators: reduce, dot, conv, and conv-base. These operators share a common characteristic: they perform *reductions* or *accumulations* over one or more axes. Because the axes being reduced may have unbounded sizes, we cannot simply expand the reduction and operate over all elements explicitly. Instead, we treat the reduction sum symbolically and provide special handling for such elements during verification. Moreover, the user supplies a bijection between the LHS and RHS reduction elements, which allows us to reason about them abstractly. Once this is done, the rule can be symbolically evaluated as usual, and all of our previous results regarding sufficient rank and counterexample lowering continue to hold.

Counting Non-Trivial Conditions. We introduced a simplifying assumption (for notational ease) in §D that every condition in a rule is uniformly represented as a conjunction of one fold per aggregated-axis, regardless of whether that aggregated-axis contributes meaningful conditions. In practice, some conditions may be *trivial*: their associated fold is degenerate and therefore impose no constraints. Such conditions should not contribute to the sufficient rank computation.

To account for this, we introduce a routine NumConds which takes an RClass c and returns the number of conditions having an aggregated-axis belonging to the RClass c and whose fold is non-trivial. Only these conditions are *relevant* and contribute to the sufficient rank of c.

Putting it all Together. After relaxing all the assumptions from §C.1, we come up with the following lemma:

Lemma 8. Let R be any rewrite rule written in TensorRight DSL. Let m be an RClass-rank map for R and c be an RClass appearing in R. If $k = \textsc{InferBound}(R, c)$, then

$$\forall i \geq k,\ \mathsf{valid}(m[c \mapsto i]) \rightarrow \mathsf{valid}(m[c \mapsto i{+}1])$$

In other words, for all $i \geq k$ and for *any* ranks of the other RClasses, if the rule is valid when c has rank i, then it implies that the rule is valid when c has rank $i{+}1$. The InferBound routine is described in Algorithm 2.

RHS Expression Validity. Our definition of rewrite rule validity (Definition 1) states that the LHS and RHS expressions should be equal for all variable valuations, given the precondition C holds and the LHS expression is valid, i.e., valid-expr(LHS) holds. In reality, we require a stronger

Algorithm 2: Sufficient Rank for an RClass

Inputs : Rewrite rule R & RClass c
Output: *bound*, i.e., a sufficient rank for c

```
Function InferBound (R, c) :
    bound ← 0;
    tensors ← TensorsWithRClass(R, c);
    for t ∈ tensors do
        n ← NumTensorAccess(R, t);
        if n > 1 then
            bound ← bound + (n choose 2);
        end
    end
    bound ← bound + NumConds(R, c);
    return Max(bound, 1)
End Function
```

postcondition that the RHS expression should be valid, i.e., valid-expr(RHS) holds. On taking contrapositives and considering $(k{+}1)$- and k-ranked instantiations, we get:

$$\exists\, v^{k+1} \in vars^{k+1}, C^{k+1} \wedge \text{valid-expr}(\text{LHS}^{k+1}) \wedge \left(\left[\!\left[\text{LHS}^{k+1} \right]\!\right] \neq \left[\!\left[\text{RHS}^{k+1} \right]\!\right] \vee \neg\text{valid-expr}(\text{RHS}^{k+1}) \right)$$

$$\downarrow$$

$$\exists\, v^{k} \in vars^{k}, C^{k} \wedge \text{valid-expr}(\text{LHS}^{k}) \wedge \left(\left[\!\left[\text{LHS}^{k} \right]\!\right] \neq \left[\!\left[\text{RHS}^{k} \right]\!\right] \vee \neg\text{valid-expr}(\text{RHS}^{k}) \right)$$

Thus there are two distinct kinds of counterexamples to lower from rank $k{+}1$ to k:

- Mismatch counterexamples: If $[\![\text{LHS}]\!] \neq [\![\text{RHS}]\!]$ in rank $k{+}1$, then we lower to a witness of $[\![\text{LHS}]\!] \neq [\![\text{RHS}]\!]$ in rank k. This is handled in §E: the sufficient rank given by Algorithm 2 guarantees a counterexample lowering for this case.
- RHS invalidity counterexamples: If $\neg$valid-expr(RHS) holds in rank $k{+}1$, then we lower to a witness of $\neg$valid-expr(RHS) in rank k. The condition valid-expr(RHS) lies in the fragment $\Phi_\wedge$, i.e., it is a conjunction of fold-atoms. As in the condition-analysis (Lemma 1), any $(k{+}1)$-ranked falsifying fold-conjunction contains at least one fold, say f^{k+1}, whose truth value if false. To ensure that valid-expr(RHS^k) is false, we ensure that f^k is false. This can be done by choosing an appropriate named-axis, giving a sufficient rank of 1.

For worst–case bounds we take the maximum of the two cases. Since each branch is already at least 1, the sufficient rank given by Algorithm 2 remains unchanged.

Input Tensor Aliasing in Tensor Graph Rewrites. Our analysis in Lemma 5 assumes that distinct input tensors do not alias. For example, in the tensor expression $e = \text{binary}(+, \text{X}, \text{Y})$, we assume that X and Y are different tensors. In such a case, during counterexample construction, accesses to different tensors are treated as independent, and no cross-tensor constraints arise. Therefore, the sufficient rank decomposes cleanly into the sum of per-tensor contributions.

However, a compiler matching a symbolic expression e inside a tensor computation graph, may encounter a match such as $\mathsf{binary}(+, t, t)$, where both operands refer to the same concrete tensor. This breaks the independence assumption: accesses that originate syntactically from different tensors may in fact refer to the same underlying source, thereby introducing additional cross-tensor interactions. To account for such cases, (1) one can verify a different rewrite capturing the aliasing relation, or (2) adjust the sufficient rank to account for additional cross-tensor interactions and constraints introduced from them. More specifically, for tensors $\mathsf{X}_1, \cdots, \mathsf{X}_q$ with $n_{\mathsf{X}_1}, \cdots, n_{\mathsf{X}_q}$ number of accesses, respectively, the new constraints are bounded by $\binom{\sum_i n_{\mathsf{X}_i}}{2}$, instead of $\sum_i \binom{n_{\mathsf{X}_i}}{2}$.