The Formal Core of OCANNL Shape and Projection Inference

Lukasz Stafiniak, Claude Fable 5, GPT 5.5

This report is a standalone companion to the OCANNL workshop article "Neural Networks Without Shape Boilerplate: An OCaml DSL Case Study" (`docs/ocannl_workshop_article_human.md` in the OCANNL repository). It consolidates the working notes in `docs/blog/ocannl-formal-core.md` and `docs/formal-core-appendix.md` (also in the repository) into one coherent account. The scope is the core shape language used by the current proof effort: dimensions, row variables, broadcasting, flat row equality, solving, closing, and core projection inference. The staged extensions for affine indexing, convolution padding, concatenation, and total-element constraints are discussed only to delimit what is already part of the core proof and what is outside that core.

The main result is deliberately not overstated. The core semantic rules are proved sound; finite solving terminates with the persistent rank-cycle guard; successful closing produces a real solution; and projection inference over closed core shapes is canonical and address-safe.

What is proved here:

- The rank-cycle guard is sound and exact for the facts it records.
- The Detection Lemma is proved: every divergent rank-unsatisfiable run eventually records a positive cycle.
- Projection inference is fully proved for the core projection language.

The remaining caveats are separate from those proved core results:

- The deterministic solver commits policy choices for marker placement and for deferred word equations. These choices are sound but incomplete for the abstract $\approx$ semantics. The conjecture that surface OCANNL programs do not observe the placement incompleteness remains open.
- The affine, convolution, and concatenation strata used by the implementation need a separate staged soundness proof.

The implementation was checked against the formal rules while preparing this report, especially `tensor/row.ml` and `tensor/shape.ml`. The current code already implements the key corrections discovered by the formalization: flat closed-row equality, deferred shifted self equations/row-subtyping constraints,

explicit closed-closed row-subtyping checks across structural flanks, and the persistent rank graph. No code change was required for this report.

1. Dimensions

Fix a set B of basis tags with a distinguished ordinary basis default. There is also a special claim-free unit written $\mathbf{1}_\emptyset$; in the implementation it is represented by a dimension of size 1 with basis bcast_if_1. The mathematical presentation keeps it separate from ordinary atoms, including the ordinary atom 1_{default}.

Definition 1.1 (Dimensions). The set of dimensions is

$$D = \{\mathbf{1}_\emptyset\} \cup \{\, n_b \mid n \geq 1 \text{ and } b \in B \,\}.$$

Elements n_b are atoms. Adjoin a fresh bottom element $\bot$ and write $D_\bot = D \cup \{\bot\}$.

Definition 1.2 (Broadcast order). On D,

$$d_1 \sqsubseteq d_2 \quad \iff \quad d_2 = \mathbf{1}_\emptyset \text{ or } d_1 = d_2.$$

Extend this to $D_\bot$ by $\bot \sqsubseteq d$ for every d.

Thus $\mathbf{1}_\emptyset$ is the greatest element, atoms are pairwise incomparable, and $\bot$ is only a proof device for meets.

Proposition 1.3 (Dimension lattice). $D_\bot$ is a bounded lattice of height two. Its top is $\mathbf{1}_\emptyset$, its bottom is $\bot$, and the atoms form an antichain between them. Meets and joins are:

$$\begin{aligned} &d \wedge d = d, \quad d \wedge \mathbf{1}_\emptyset = d, \quad d_1 \wedge d_2 = \bot \quad \text{for distinct atoms } d_1, d_2, \\ &d \vee d = d, \quad d \vee \bot = d, \quad d_1 \vee d_2 = \mathbf{1}_\emptyset \quad \text{for distinct atoms } d_1, d_2. \end{aligned}$$

Proof. Reflexivity follows from equality in Definition 1.2. For antisymmetry, if $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_1$ and $d_1 \neq d_2$, then the first inequality forces $d_2 = \mathbf{1}_\emptyset$, while the second forces $d_1 = \mathbf{1}_\emptyset$, a contradiction. For transitivity, if the rightmost element is $\mathbf{1}_\emptyset$ the claim is immediate; otherwise the second inequality is equality, and the first inequality gives the result.

The displayed meet and join equations exhaust all cases: equal arguments, a top or bottom argument, and two distinct atoms. In each case the candidate is checked directly from the order. A distinct pair of atoms has no common lower bound above $\bot$ and no common upper bound below $\mathbf{1}_\emptyset$. Therefore the displayed operations are the greatest lower bounds and least upper bounds. □

Proposition 1.4 (Non-distributivity). If there are at least three atoms, $D_\bot$ is not distributive. It is, however, modular; the earlier working claim that the lattice is not modular was false.

Proof. Let a, b, and c be distinct atoms. The sublattice $\{\bot, a, b, c, \mathbf{1}_\emptyset\}$ is the diamond lattice M_3. It violates distributivity:

$$a \wedge (b \vee c) = a \wedge \mathbf{1}_\emptyset = a, \qquad (a \wedge b) \vee (a \wedge c) = \bot \vee \bot = \bot.$$

So any lattice containing it as a sublattice is not distributive. The same argument cannot show non-modularity because M_3 is modular. In fact the height-two atom lattice is modular: if $x \sqsubseteq z$, then the only non-trivial case is $x = \bot$ or $z = \mathbf{1}_\emptyset$, and the modular law reduces to the defining equations for meet and join. □

2. Rows

A row is a sequence of axes with a distinguished insertion point. The marker is not syntax trivia: it is where broadcasting inserts implicit axes and where a row variable absorbs axes during inference.

Definition 2.1 (Rows). A ground row is a triple $l \cdot \diamond \cdot r$ with $l, r \in D^*$. The lists are the leading and trailing flanks. Its rank is $|l| + |r|$. Row identity is marker-sensitive:

$$[3] \cdot \diamond \cdot [4] \neq [3, 4] \cdot \diamond \cdot [].$$

Definition 2.2 (Expansion). For $R = l \cdot \diamond \cdot r$ and $n \geq \mathrm{rank}(R)$,

$$R \uparrow n = l \cdot \mathbf{1}_\emptyset^{\,n-\mathrm{rank}(R)} \cdot r.$$

The expansion inserts claim-free broadcast axes at the marker.

Definition 2.3 (Row order). Write $R_2 = l_2 \cdot \diamond \cdot r_2$ and $R_1 = l_1 \cdot \diamond \cdot r_1$. Then $R_2 \sqsubseteq R_1$ iff:

1. $|l_1| \leq |l_2|$ and $|r_1| \leq |r_2|$;
2. for every position $i < \mathrm{rank}(R_2)$, $(l_2 \cdot r_2)[i] \sqsubseteq (R_1 \uparrow \mathrm{rank}(R_2))[i]$.

The left row is the more material row. A shorter row can sit above a longer one because the shorter row has more broadcast credit at its marker.

Lemma 2.4 (Expansion monotonicity). If $R_2 \sqsubseteq R_1$, then for every $n \geq \mathrm{rank}(R_2)$ and every position $i < n$,

$$(R_2 \uparrow n)[i] \sqsubseteq (R_1 \uparrow n)[i].$$

Proof. Write $R_j = l_j \cdot \diamond \cdot r_j$. There are three regions.

If $i < |l_2|$, the left side is $l_2[i]$. When $i < |l_1|$, the defining pointwise condition gives $l_2[i] \sqsubseteq l_1[i]$. When $i \geq |l_1|$, the position lies in the middle of $R_1 \uparrow n$ unless

it is already in the trailing region; but $i < |l_2| \le n - |r_2| \le n - |r_1|$, so it is not trailing. Hence the right side is $\mathbf{1}_\emptyset$, and the inequality is automatic.

If $|l_2| \le i < n - |r_2|$, the left side is $\mathbf{1}_\emptyset$. By the flank-fit conditions, i also lies in the middle of $R_1 \uparrow n$, so the right side is $\mathbf{1}_\emptyset$.

If $i \ge n - |r_2|$, the proof is symmetric to the leading-flank case, using the outer-right alignment of trailing flanks. □

Proposition 2.5 (Partial order). The row relation $\sqsubseteq$ is a partial order.

Proof. Reflexivity is immediate. For antisymmetry, mutual flank-fit gives equal leading and trailing flank lengths. The expansions at that common rank are the rows' flat contents, so pointwise antisymmetry in D gives equal flanks.

For transitivity, suppose $R_3 \sqsubseteq R_2 \sqsubseteq R_1$ and let $n = \mathrm{rank}(R_3)$. Flank-fit conditions compose. The pointwise condition follows from $R_3 \uparrow n \sqsubseteq R_2 \uparrow n$ by the first inequality and $R_2 \uparrow n \sqsubseteq R_1 \uparrow n$ by Lemma 2.4 applied to $R_2 \sqsubseteq R_1$. Transitivity in D completes the proof. □

Proposition 2.6 (Top row). The empty marked row $[] \cdot \diamond \cdot []$ is the greatest row, and it is the unique greatest row.

Proof. Its flank lengths are zero and its expansion to any rank is all $\mathbf{1}_\emptyset$, so every row refines it. If some row is above the empty row, the flank-fit inequalities force both of its flanks to have length zero. □

Proposition 2.7 (Row joins). Rows form a join-semilattice with top. The least upper bound of $R_1 = l_1 \cdot \diamond \cdot r_1$ and $R_2 = l_2 \cdot \diamond \cdot r_2$ is $R_j = l_j \cdot \diamond \cdot r_j$, where:

$$|l_j| = \min(|l_1|, |l_2|), \qquad l_j[i] = l_1[i] \vee l_2[i].$$

and similarly for the trailing flank, aligned from the right.

Proof. R_j is an upper bound because its flanks are no longer than either input flank. On positions retained in R_j, the dimension join is an upper bound; positions outside the retained flanks expand to $\mathbf{1}_\emptyset$.

Now let U be any upper bound. Since $R_1 \sqsubseteq U$ and $R_2 \sqsubseteq U$, the flanks of U are no longer than the corresponding flanks of both inputs, hence no longer than the flanks of R_j. Thus the flank-fit condition for $R_j \sqsubseteq U$ holds. At each retained leading position of U, both $l_1[i]$ and $l_2[i]$ are below U's dimension there, so their join is below it. The trailing side is symmetric. Positions in the middle of U are $\mathbf{1}_\emptyset$. Therefore $R_j \sqsubseteq U$. □

Definition 2.8 (Flat equivalence). The flat content of $l \cdot \diamond \cdot r$ is $l \cdot r$. Define $R \approx R'$ iff their flat contents are equal. Equivalently, they have equal rank and equal expansion at that rank.

$\approx$ is not the identity of row elements. It is the equality relation used for row equality constraints, including einsum template matching.

Proposition 2.9 ($\approx$ versus $\sqsubseteq$).

1. The equivalence induced by mutual $\sqsubseteq$ is identity.
2. Distinct rows related by $\approx$ are incomparable under $\sqsubseteq$.
3. $\approx$ is not a congruence for the marked row order.

Proof. Item 1 is antisymmetry. For item 2, two flat-equal but distinct rows must place the marker at different positions. If one has a longer leading flank, it has a shorter trailing flank because total rank is equal. The leading flank-fit condition fails in one direction and the trailing flank-fit condition fails in the other.

For item 3, take

$$R = [3] \cdot \diamond \cdot [2, 4], \qquad S = [3] \cdot \diamond \cdot [4], \qquad S' = [3, 4] \cdot \diamond \cdot [].$$

Then $S \approx S'$, and $R \sqsubseteq S$: the expansion of S to rank three is $[3, \mathbf{1}_\emptyset, 4]$. But $R \sqsubseteq S'$ fails because the upper row would need a leading flank of length at most one, while S' has leading length two. Thus replacing an $\approx$-equivalent row inside the marked order can change truth. $\square$

This is the first non-principality point. Flat equality is needed for useful spec matching, but marker placement remains a real policy choice whenever an equality binds a row variable.

2.1 Rigid Rows and Row Subtyping

It is useful to name the marker-free interpretation of a flat row.

Definition 2.10 (Rigid rows). A rigid row is $F^\bullet$ for $F \in D^n$. Its expansion is defined only at rank n and equals F. Extend the order by:

- marked-marked: Definition 2.3;
- rigid-rigid: equal rank and pointwise refinement;
- rigid below marked: $F^\bullet \sqsubseteq R$ iff $\mathrm{rank}(F) \geq \mathrm{rank}(R)$ and $F[i] \sqsubseteq (R \uparrow \mathrm{rank}(F))[i]$ pointwise;
- marked below rigid: never.

Proposition 2.11 (Two-sorted order). The extended relation is a partial order. The empty marked row remains the unique top. Rigid row equality is flat content equality.

Proof. Reflexivity and antisymmetry hold within each sort as before; a cross-sort mutual comparison is impossible because marked rows are never below rigid rows. For transitivity, the all-marked case is Proposition 2.5, the all-rigid case is pointwise transitivity, and the mixed cases are:

- rigid $\sqsubseteq$ rigid $\sqsubseteq$ marked, where the equal rigid ranks and pointwise inequalities compose;

- rigid $\sqsubseteq$ marked $\sqsubseteq$ marked, where Lemma 2.4 transports the second marked inequality to the rigid rank.

No other mixed chain is possible because marked-below-rigid comparisons are forbidden. Topness of the empty marked row follows as in Proposition 2.6. $\square$

The forbidden marked-below-rigid clause is forced. If even rank-equal marked rows were allowed below their rigidifications, then

$$[5] \cdot \diamond \cdot [3] \sqsubseteq [] \cdot \diamond \cdot [3] \sqsubseteq [3]^{\bullet}.$$

would require $[5] \cdot \diamond \cdot [3] \sqsubseteq [3]^{\bullet}$, impossible by rank.

Definition 2.12 (Row subtype/refinement). Write $R \preceq S$ when $\mathrm{flat}(R)^{\bullet} \sqsubseteq S$ in the two-sorted order. This is the marker-erasing-on-the-left, marker-sensitive-on-the-right relation used for semantic row-subtyping constraints. The left row is viewed as a rigid flat result; the right row remains a marked broadcastable operand whose marker determines where implicit $\mathbf{1}_{\emptyset}$ axes are inserted. We call this a row subtype/refinement relation to emphasize its role as the surface shape relation, while reserving the marked order $\sqsubseteq$ for the internal structural order of rows.

3. Terms, Substitutions, and Constraints

Definition 3.1 (Terms). Dimension terms are:

$$t ::= \alpha \mid n_b \mid \mathbf{1}_{\emptyset}.$$

Row terms are:

$$R ::= l \cdot \diamond \cdot r \mid l \cdot \langle \rho \rangle \cdot r.$$

where l and r are sequences of dimension terms. A term is ground if it is variable-free. Each row term contains at most one row variable, at the marker.

Definition 3.2 (Substitution). A substitution is a finite sort-respecting map from variables to terms. It is kept idempotent:

$$\mathrm{dom}(\sigma) \cap \mathrm{vars}(\mathrm{range}(\sigma)) = \emptyset.$$

Application to row variables splices at the marker. If $\sigma(\rho) = l' \cdot \langle \rho' \rangle \cdot r'$, then

$$\sigma(l \cdot \langle \rho \rangle \cdot r) = \sigma(l) \cdot l' \cdot \langle \rho' \rangle \cdot r' \cdot \sigma(r).$$

If $\sigma(\rho) = l' \cdot \diamond \cdot r'$, the result is the closed row $\sigma(l) \cdot l' \cdot \diamond \cdot r' \cdot \sigma(r)$.

Lemma 3.3 (Composition). With this splice definition, $(\sigma \circ \tau)(X) = \sigma(\tau(X))$ for all terms X.

Proof. Dimension terms are immediate. For rows, the only non-structural case is a row variable at the marker. Applying τ splices $\tau(\rho)$ into the outer row; applying σ then splices any row variable in $\tau(\rho)$ at the same marker and maps the flank dimensions. This is exactly the row-term component of the composite substitution. □

Definition 3.4 (Constraint semantics). Atomic constraints are:

$$t_1 = t_2, \qquad t_1 \sqsubseteq t_2, \qquad R_1 \approx R_2, \qquad R_1 \preceq R_2.$$

A ground substitution γ satisfies dimension equality by identity, dimension inequality by Definition 1.2, row equality by flat equivalence $\approx$, and row subtyping by Definition 2.12. Write $\mathrm{Sol}(\Phi)$ for the set of ground substitutions satisfying every constraint in Φ.

The row equality choice is the important one: equality constraints are marker-blind, while row-subtyping constraints are marker-blind on the lower/result side and marker-sensitive on the upper/operand side.

Definition 3.5 (Substitution models and order). A substitution σ models Φ, written $\sigma \vDash \Phi$, iff every ground substitution γ makes $\gamma \circ \sigma$ a solution. The substitution preorder is $\sigma_1 \leq \sigma_2$ iff there exists u with $u \circ \sigma_1 = \sigma_2$.

Example 3.6 (No principal model). For $\Phi = \{3_b \sqsubseteq \alpha\}$, the identity substitution is not a model because α could be grounded to a different atom. The substitutions $[\alpha \mapsto 3_b]$ and $[\alpha \mapsto \mathbf{1}_\emptyset]$ are both models, and neither factors through the other: ground dimensions cannot be rewritten by a later substitution. Thus solvable constraint sets need not have a least substitution model. The solver must return a substitution plus a residual bound store.

3.1 Constraint Generation

The core operation rules generate row-subtyping constraints for ordinary broadcasting and row equalities for spec-based operations. Let C be the result shape; A, B, and D operands; and S_k the row of kind $k \in \{\mathsf{Batch}, \mathsf{Input}, \mathsf{Output}\}$.

Operation	Generated row constraints
transpose	$C_B \preceq A_B$, $C_I \preceq A_O$, $C_O \preceq A_I$
pointwise unary	$C_k \preceq A_k$ for every kind k
pointwise binary	$C_k \preceq A_k$, $C_k \preceq B_k$ for every kind k
pointwise ternary	$C_k \preceq A_k$, $C_k \preceq B_k$, $C_k \preceq D_k$ for every kind k
compose	$A_I \preceq B_O$, $C_B \preceq A_B$, $C_B \preceq B_B$, $C_I \preceq B_I$, $C_O \preceq A_O$
compose accumulate	compose constraints plus $C_k \preceq D_k$
batch slice	$s \cdot C_B \approx A_B$, $C_I \approx A_I$, $C_O \approx A_O$

Operation	Generated row constraints
permute	$C_k \approx T_{lhs,k}$, $T_{rhs,k} \approx A_k$
einsum	$C_k \approx T_{lhs,k}$, $T_{rhs,i,k} \approx A_{i,k}$

This table matches the implementation in `tensor/shape.ml`: ordinary pointwise and compose rules emit `Row_ineq`, while batch slice, permute, and einsum emit `Row_eq`. Batch slice is the canonical source of a non-empty leading flank; it prepends the slice axis to the result batch row.

4. The Solver

Definition 4.1 (Configurations). A configuration is

$$\langle \Phi; \sigma; B \rangle.$$

where Φ is the pending constraint list, σ is the idempotent substitution, and B is a bound store. The store records, for dimension variables, atom lower bounds and variable adjacencies; for row variables, row caps and row adjacencies. A failure configuration is written fail.

Binding a variable moves the equality into σ, substitutes through Φ and B, and re-emits all stored bounds for the bound variable, instantiated at its new value. This single re-emission discipline is the reason stored facts can be interpreted as constraints in the metatheory.

4.1 Dimension Rules

- **DE-refl:** $t = t$; discard.
- **DE-clash:** distinct ground dimensions; fail.
- **DE-bind:** $\alpha = t$, $t \neq \alpha$; bind $\alpha \mapsto t$.
- **DI-top:** $t \sqsubseteq \mathbf{1}_\emptyset$; discard.
- **DI-refl:** $t \sqsubseteq t$; discard.
- **DI-ground:** ground $d \sqsubseteq d'$; check Definition 1.2, else fail.
- **DI-pin:** $\alpha \sqsubseteq d$, with d an atom; replace by $\alpha = d$.
- **DI-pin-top:** $\mathbf{1}_\emptyset \sqsubseteq \alpha$; replace by $\alpha = \mathbf{1}_\emptyset$.
- **DI-cap:** $d \sqsubseteq \alpha$, with d an atom; record the lower bound. If a distinct atom is already recorded, bind $\alpha \mapsto \mathbf{1}_\emptyset$.
- **DI-adj:** $\alpha \sqsubseteq \beta$, $\alpha \neq \beta$; record the adjacency and forward existing caps.

Lemma 4.2 (Dimension rules preserve solutions). Each non-failing dimension rule preserves the denoted solution set. Each failing dimension rule fires only on an unsatisfiable denotation.

Proof. The reflexive and top rules remove valid constraints. DE-clash and failed DI-ground are unsatisfiable by Definition 1.2. DE-bind is the standard substitution-preserving move from a pending equality into σ.

For DI-pin, the only dimension below an atom is the atom itself. For DI-pin-top, the only dimension above the top is the top. For DI-cap, storing a first lower bound only moves the constraint into B. If a distinct atom is already stored, any common upper bound of the two atoms is $\mathbf{1}_\emptyset$, so the two caps are equivalent to $\alpha = \mathbf{1}_\emptyset$. DI-adj only records and forwards the transitive consequence $d \sqsubseteq \alpha \sqsubseteq \beta$. □

4.2 Row Rules

Definition 4.3 uses the mnemonic layout from the workshop article. In all row rules, leading flanks are aligned from the outer-left edge and trailing flanks from the outer-right edge. Overlapping aligned dimensions emit the corresponding dimension equalities or inequalities.

Definition 4.3 (Row equality rules).

- **RE-closed:** both sides closed. Fail iff ranks differ; otherwise zip flat rows and emit dimension equalities.
- **RE-open-closed:** one side open with ρ, the other closed. Fail iff the open side's total explicit flank length exceeds the closed rank; otherwise bind ρ to the uncovered middle of the closed flat row, using the inherit-the-split placement policy.
- **RE-nested:** both sides open, distinct variables, and both surpluses are on ρ_2's side. Bind $\rho_1 \mapsto m_l \cdot \langle \rho_2 \rangle \cdot m_r$.
- **RE-cross:** both sides open, distinct variables, with split surpluses. Defer the residual word equation $s \cdot x_1 = x_2 \cdot t$ into closing.
- **RE-same-empty:** same variable and no surplus; discard.
- **RE-same-occurs:** same variable and unequal total flank lengths; fail.
- **RE-same-rot:** same variable, equal total flank length, but shifted split. Defer the rotational word equation $x \cdot t = s \cdot x$ into closing.

Explanation. RE-closed is flat equality, so marker placement is ignored. RE-open-closed forces the flat content of the row variable, but not its marker placement. The implementation chooses the closed side's split when available and the left edge otherwise. That choice is a policy commitment: sound, but not complete for all $\approx$-solutions. RE-nested is the same phenomenon with another row variable in the middle.

RE-cross is a two-variable word equation. Its solutions are the principal family

$$x_2 = s \cdot w, \qquad x_1 = w \cdot t.$$

plus finitely many sporadic crossover solutions where x_2 is a proper prefix of s. Eagerly binding a fresh middle variable would keep only the principal family and would be a rank commitment not entailed by $\approx$. Therefore the correct core rule defers.

RE-same-rot is governed by the classical conjugacy equation $x \cdot t = s \cdot x$. Non-empty periodic solutions exist when s and t are conjugate. The deterministic solver does not represent that family; it defers until either other constraints solve the variable or closing chooses the least-material branch.

Definition 4.4 (Row inequality rules).

- **RI-cap:** operand open and result supplies at least the operand flanks. Record the result's interior residue as a cap on the operand row variable.
- **RI-closed-op:** operand closed and result rank is at least the operand flanks. Compare explicit operand material against the result flat row; inserted operand middle positions are unconstrained.
- **RI-short-closed:** result closed and shorter than the operand's known flanks; fail.
- **RI-deficit:** result open and shorter than operand known flanks by $k > 0$. Grow the result variable by a template with fresh dimensions and a fresh row variable, then reprocess.

RI-closed-op is easy to state incorrectly. The operand's inserted broadcast positions are unconstrained because they are $\mathbf{1}_\emptyset$, but the operand's explicit axes still pin corresponding result positions, even across structural flanks. The current implementation performs this flat comparison.

Lemma 4.5 (Row rule preservation). Every semantic row rule preserves solutions exactly up to extension on fresh variables. Semantic failures occur only when the denotation has no solution. The placement and deferred-equation steps are policy steps: every new solution restricts to an old solution, but some old solutions can be lost.

Proof.

For RE-closed, flat equality of equal-length closed words is equivalent to the pointwise dimension equalities. Unequal ranks are unsatisfiable.

For RE-open-closed, write the open side as $l \cdot \langle \rho \rangle \cdot r$ and the closed flat word as w. The flat equation is $\gamma(l) \cdot x \cdot \gamma(r) = w$, where $x = \mathrm{flat}(\gamma\rho)$. Since the flank lengths are fixed, a solution exists iff $|l| + |r| \leq |w|$; then x is the unique middle segment and the outer segments emit exact dimension equalities. Only the split of x around a marker is policy.

For RE-nested, cancellation of common outer flanks leaves $x_1 = m_l \cdot x_2 \cdot m_r$. The flat content of the binding is entailed. The inherited split is policy, for the same reason as RE-open-closed.

For RE-cross, cancellation leaves $s \cdot x_1 = x_2 \cdot t$ with non-empty split surpluses. If $|x_2| \geq |s|$, then $x_2 = s \cdot w$ and $x_1 = w \cdot t$. If $|x_2| < |s|$, then x_2 is a proper prefix of s; the remaining suffix of s must be a prefix of t, and x_1 is forced to the complementary suffix of t. These cases are exhaustive, so deferral preserves the denotation. Any later closing choice is policy.

RE-same-empty removes a reflexive equation. RE-same-occurs with unequal total flank lengths is unsatisfiable by length. RE-same-rot is a deferred word equation; deferral is exact, while the eventual least-material resolution is policy.

For row-subtyping constraints, the aligned explicit overlaps are exactly the pointwise requirements of Definition 2.12. RI-cap packages the remaining interior requirement as a stored cap. RI-closed-op discards only positions where the operand expansion inserts $\mathbf{1}_\emptyset$, and emits comparisons for explicit operand axes. RI-short-closed violates the rank requirement with a fixed-rank result. RI-deficit is exact because any solution must give the result variable at least the missing k axes; those axes decompose uniquely around the marker according to the template. Fresh dimensions and the fresh row variable are assigned from that decomposition, and conversely every solution after the template binding restricts to a solution before it. □

4.3 Rank-Fact Graph

The row rules without an extra check can diverge on rank-unsatisfiable inputs:

$$\rho_1 \preceq [a] \cdot \langle \rho_2 \rangle, \qquad \rho_2 \preceq [b] \cdot \langle \rho_1 \rangle.$$

The first constraint forces $\mathrm{rank}(\rho_1) \geq \mathrm{rank}(\rho_2) + 1$; the second forces the reverse strict inequality. Operationally, each RI-deficit step grows one row variable and lengthens the other constraint's operand side.

Definition 4.6 (Rank graph). Maintain a persistent directed graph G on row variables. An edge

$$\rho \xrightarrow{k} \rho'.$$

with $k \geq 0$ records the entailed fact

$$\mathrm{rank}(\gamma\rho) \geq \mathrm{rank}(\gamma\rho') + k.$$

Edges are recorded at three points:

1. **RG-bind:** solving $\rho \mapsto l \cdot \langle \rho' \rangle \cdot r$ records $\rho \xrightarrow{|l|+|r|} \rho'$;
2. **RG-equal-flanks:** processing $R_{\mathrm{res}} \preceq R_{\mathrm{op}}$ with both sides open and equal known flank lengths records $\rho_{\mathrm{res}} \xrightarrow{0} \rho_{\mathrm{op}}$;
3. **RG-deficit:** RI-deficit with deficit $k > 0$ and open operand records $\rho_{\mathrm{res}} \xrightarrow{k} \rho_{\mathrm{op}}$ before growing the result.

Every insertion is guarded. If the new edge closes a directed cycle of total positive weight, the solver fails.

No edge is recorded in RI-cap when the result already has surplus known flanks. That situation entails only $\mathrm{rank}(\rho_{\mathrm{res}}) \geq \mathrm{rank}(\rho_{\mathrm{op}}) - s$, a non-positive lower-bound difference. Recording it as a zero or positive edge is unsound and rejects valid programs.

Proposition 4.7 (Exactness for recorded facts). A finite set of recorded rank facts is satisfiable over natural-number ranks iff G has no positive cycle. Therefore the guard fails exactly when the recorded facts are jointly unsatisfiable.

Proof. If there is a positive cycle, summing the inequalities around it gives $\mathrm{rank}(\rho) \geq \mathrm{rank}(\rho) + w$ for $w > 0$, impossible.

Conversely, assume there is no positive cycle. For each vertex ρ, define $r(\rho)$ as the supremum of the total weights of directed walks starting at ρ. Any walk that revisits a vertex contains a cycle of weight zero, since all weights are non-negative and positive cycles are absent. Removing such a cycle does not decrease the walk weight, so the supremum is attained on a simple path and is finite. For every edge $\rho \xrightarrow{k} \rho'$, prefixing that edge to an optimal walk from ρ' shows $r(\rho) \geq r(\rho') + k$. Thus r satisfies every recorded fact. □

This proposition proves the guard sound. It does not prove that the graph records enough facts to catch every possible divergence.

5. Solver Metatheory

Let the denotation of a configuration be

$$\widehat{C} = \Phi \cup \mathrm{eqns}(\sigma) \cup \mathrm{constr}(B).$$

At row sort, $\mathrm{eqns}(\sigma)$ is read using $\approx$.

Lemma 5.1 (Solution preservation with policy coordinate). For every solver step:

- semantic steps preserve $\mathrm{Sol}(\widehat{C})$ exactly, up to extension to fresh variables;
- semantic failures imply $\mathrm{Sol}(\widehat{C}) = \emptyset$;
- policy steps refine the solution set: every new solution is an old solution, but not every old solution necessarily survives.

Proof. Dimension rules are Lemma 4.2 and row rules are Lemma 4.5. Moving constraints between Φ, σ, and B preserves the denotation because the moved equality or bound is retained in $\mathrm{eqns}(\sigma)$ or $\mathrm{constr}(B)$, and re-emitted constraints are the stored constraints instantiated through a binding already present in the denotation.

The only non-semantic components are exactly:

- marker placement in RE-open-closed and RE-nested;

- closing-stage resolution of RE-cross and RE-same-rot.

These choose one representative or branch from a family of $\approx$-solutions. The committed representative satisfies the original flat equality, so the step is sound. It can lose solutions because row-subtyping constraints can distinguish $\approx$-equivalent marker placements when the committed variable appears on the upper/operand side, and because the deferred word equations can have non-selected rank branches. □

Definition 5.2 (Rank abstraction). The rank abstraction of $\widehat{C}$ is the finite set of rank facts obtained from row constraints. For an open-open row constraint comparing

$$l \cdot \langle \rho \rangle \cdot r, \qquad l' \cdot \langle \rho' \rangle \cdot r'.$$

as result/operand of $\preceq$, the fact is

$$\operatorname{rank}(\rho) \geq \operatorname{rank}(\rho') + (|l'| + |r'|) - (|l| + |r|).$$

An equality contributes both directions. Open-closed constraints contribute the corresponding absolute rank bounds. A rank model is a map from row variables to natural numbers satisfying these facts.

Every concrete solution induces a rank model. The converse fails because rank abstraction forgets dimensions and bases.

Lemma 5.3 (Rank conservativity). Every non-failing rule maps a configuration with a rank model to one with a rank model agreeing on the old variables and extended to fresh variables.

Proof. Dimension rules do not change row ranks. RE-closed emits only dimension constraints. RE-open-closed and RE-nested replace a row equality by a binding whose rank equation is the consumed rank equation. RE-cross and RE-same-rot defer, so the denotation is unchanged.

RI-cap stores the same rank condition that the original inequality contributed. RI-closed-op and RI-short-closed are ground-sided; the non-fail case contributes no new open-open rank obligation. RI-deficit with deficit k is triggered only when the consumed rank fact entails $\operatorname{rank}(\rho_{\text{res}}) \geq k$; extend the model by $\operatorname{rank}(\rho_{\text{fresh}}) = \operatorname{rank}(\rho_{\text{res}}) - k$. Binding re-emission substitutes a term whose explicit flank length accounts for exactly the rank removed from the bound variable, so numeric rank facts are preserved. □

Theorem 5.4 (Termination on rank-satisfiable inputs). If finite Φ_0 has a rank model, then every fair run terminates, with or without the rank cycle check. The check never fires on such an input.

Proof. Fix a rank model of the initial problem and extend it along the run using Lemma 5.3. Track each row-inequality lineage through storage, re-emission,

substitution, and reprocessing. If the current result side of a lineage is $l \cdot \langle \rho \rangle \cdot r$, assign the potential $\psi = \mathrm{rank}(\rho)$ from the extended rank model; if the result side is closed, assign $\psi = 0$.

An RI-deficit step on that lineage with deficit $k > 0$ binds the result variable to a template with a fresh row variable whose rank is $\mathrm{rank}(\rho) - k$, so the lineage potential strictly decreases. Other row bindings can only preserve or decrease the potential because their rank facts are entailed by the model. Therefore each lineage fires RI-deficit finitely many times.

No row rule forks a row-sort lineage: a consumed row constraint produces at most one row-sort continuation. Thus the number of lineages is bounded by the initial finite set plus continuations created by finite closing commitments; within the solving phase considered here, it is finite. Since deficit steps are finite, fresh row variables are finite. Each variable binds at most once, so binding events are finite. Between binding events, processing pending constraints strictly decreases the lexicographic measure consisting of the number of pending row constraints and then dimension constraints, because storing a bound moves it out of Φ and non-storing rules consume their trigger. Hence the run terminates.

Every rank-graph edge is an entailed rank fact. The fixed rank model satisfies all entailed facts, so by Proposition 4.7 no positive cycle can be recorded. □

Lemma 5.5 (Detection Lemma). Every infinite run inserts, at some finite stage, an edge that closes a positive cycle in the persistent rank graph.

Proof. We use the rank abstraction of Definition 5.2 and the operational facts already used in Theorem 5.4: row-sort lineages are finite in number and do not fork; every variable binds at most once; and a fair run eventually processes every re-emitted row-subtyping constraint.

First, an infinite run must perform infinitely many RI-deficit steps with open operands. If only finitely many binding events occurred, then after the last one the pending-constraint measure would strictly decrease to solved form or failure. Infinitely many bindings require infinitely many RI-deficit steps, because the equality rules mint only along existing finite lineages and do not fork them. Deficits against closed operands are bounded: a closed operand has fixed rank, so after finitely many result-side growth steps the result is large enough or the constraint fails. Therefore infinitely many deficits have open operands, and each records an RG-deficit edge.

Now consider the signed rank abstraction restricted to the finite set of row-inequality lineages that participate in infinitely many such deficits. If this restricted abstraction had a rank model, the potential proof of Theorem 5.4 would bound all deficits on those lineages, contradiction. Hence it has no rank model. Difference-constraint duality gives a directed cycle of lineages

$$L_1, \ldots, L_n.$$

whose signed weights have positive total $W > 0$. Write the current live variable for the vertex of L_i as x_i, and write $h_i \geq 0$ for the amount of explicit rank already peeled off by substitutions from the cycle's original variable to x_i. The persistent rank graph contains a binding path from the original variable to x_i of weight h_i.

If the original signed weight on edge $L_i : i \to i+1$ is c_i, then after the current substitutions the live weight is

$$c'_i = c_i + h_{i+1} - h_i.$$

This is just substitution algebra: replacing the source by a template with h_i explicit axes subtracts h_i from the edge weight, while replacing the target by a template with h_{i+1} explicit axes adds h_{i+1}. The cycle sum is invariant:

$$\sum_i c'_i = \sum_i c_i = W > 0.$$

Consequently at least one live edge on the cycle is non-negative at every time. If a live edge has weight zero, fair processing records the RG-equal-flanks edge. If it has weight $k > 0$, fair processing records the RG-deficit edge and then binds the source variable to a template with k explicit axes. In terms of the live weights, this sets that edge's weight to zero and adds the same k to the preceding edge's weight. Thus positive surplus is never destroyed; it is transported backwards around the finite cycle. Negative slack on an RI-cap edge is finite, so repeated transport into that edge eventually exhausts the slack and turns the edge into a zero or positive live edge, which is then recorded. This is the negative-edge discharge case: a hidden fact $\mathrm{rank}(x) \geq \mathrm{rank}(y) - s$ becomes recorded once operand-side substitution has inserted at least s axes.

Because the cycle is finite and the run is fair, some full circulation occurs: for each lineage edge on the positive signed cycle, choose a time after the previous chosen edge's target variable has advanced to the source variable used by the next chosen edge, and at which the edge is processed with non-negative live weight. Such times exist by the surplus-transport argument above. At the chosen processing of edge i, the graph records an edge

$$x_i^t \xrightarrow{c_i + h_{i+1}^t - h_i^t} x_{i+1}^t.$$

Between this target x_{i+1}^t and the source chosen for the next edge, the persistent binding graph contains a path of weight $h_{i+1}^{\mathrm{next}} - h_{i+1}^t$, since bindings only move from older variables to descendants and offsets only increase. Composing the recorded lineage edges with these binding paths gives a directed closed walk in G. Its total weight telescopes:

$$\sum_i (c_i + h_{i+1}^{\text{edge } i} - h_i^{\text{edge } i}) + \sum_i (h_i^{\text{edge } i} - h_i^{\text{edge } (i-1)})$$
$$= \sum_i c_i = W > 0.$$

Thus the closed walk contains a positive cycle. The edge that completes this walk is guarded by Definition 4.6, so the solver fails at that finite stage. This contradicts the assumption that the run was infinite without recording a positive cycle. □

Combining Theorem 5.4 with Lemma 5.5 gives total termination of the core solving phase on finite inputs: rank-satisfiable inputs terminate by the potential argument, and rank-unsatisfiable divergence is converted into finite rank-cycle failure by the persistent graph guard. This is a termination result for solving, not a completeness result for the later policy choices: placement commitments, deferred word-equation closure, and leaf closing can still reject satisfiable $\approx$-systems as described below.

Definition 5.6 (Solved form). A final configuration $\langle \emptyset; \sigma_*; B_* \rangle$ is solved when σ_* is idempotent, every variable in B_* is unsolved, each dimension variable has at most one atom cap, and all stored row caps and adjacencies have no pending rule that can fire.

Proposition 5.7 (Residual store has a greatest solution). Define $\gamma_\uparrow$ on unsolved variables by

$$\gamma_\uparrow(\alpha) = \mathbf{1}_\emptyset, \qquad \gamma_\uparrow(\rho) = [] \cdot \diamond \cdot [].$$

and extend through σ_*. Then $\gamma_\uparrow$ satisfies $\text{constr}(B_*) \cup \text{eqns}(\sigma_*)$ and is pointwise greatest for that final configuration: row variables compare by $\preceq$, and dimension variables compare by $\sqsubseteq$.

Proof. Atom caps $d \sqsubseteq \alpha$ hold because $d \sqsubseteq \mathbf{1}_\emptyset$. Dimension adjacencies hold because both sides are $\mathbf{1}_\emptyset$. Row caps and row adjacencies hold because every row is below the empty row by Definition 2.12. Bindings hold by extension through σ_*.

For greatestness, any solution maps unsolved dimension variables below $\mathbf{1}_\emptyset$ and unsolved row variables below the empty row under $\preceq$. For solved variables, use structural induction on the solved term. Ground dimensions and ground flanks compare by equality; variable occurrences use the unsolved case; for an open row term, the lower side is flattened and the upper side under $\gamma_\uparrow$ supplies $\mathbf{1}_\emptyset$ in the middle, so shared flank entries compare by induction and middle positions compare against $\mathbf{1}_\emptyset$. □

The $\preceq$ comparison at row sort is forced, not merely convenient: $\text{eqns}(\sigma_*)$ is read under $\approx$, so a solution may re-place the marker of a solved row variable,

and distinct placements of the same flat content are pairwise $\sqsubseteq$-incomparable (Proposition 2.9). Once a row variable is solved to a row of positive rank, the final configuration has no $\sqsubseteq$-greatest solution at all. When the solver substitutes for a row variable it commits to one marker placement; the commitment is a source of incompleteness (the policy steps of Lemma 5.1), not of order-theoretic strength.

Corollary 5.8 (Decision status). Successful solving implies $\mathrm{Sol}(\Phi_0)$ is non-empty. Semantic failure implies $\mathrm{Sol}(\Phi_0)$ is empty. Policy failure implies only that the policy-strengthened problem is empty. On satisfiable, placement-undiscriminated inputs, every fair run terminates and succeeds. On rank-unsatisfiable inputs, the Detection Lemma converts the only possible divergence mode into finite rank-cycle failure.

Theorem 5.9 (Representation and most generality, policy-qualified). Let a fair run on finite Φ_0 terminate in solved form $\langle \emptyset; \sigma_*; B_* \rangle$. For the fragment of the run consisting only of semantic steps:

1. $\gamma \in \mathrm{Sol}(\Phi_0)$ iff γ extends to a ground $\hat{\gamma}$ over fresh variables such that $\hat{\gamma} = \hat{\gamma} \circ \sigma_*$ and $\hat{\gamma}$ satisfies $\mathrm{constr}(B_*)$;
2. every binding in σ_* is entailed by Φ_0;
3. for every substitution $\sigma \vDash \Phi_0$, there is u with $u \circ \sigma_* = \sigma$ on the original variables.

For full deterministic runs, these statements hold for the policy-strengthened system. Up to flat row content, σ_* is still most general; marker placements are the solver's deterministic policy, not canonical semantics.

Proof. For semantic steps, Lemma 5.1 gives a chain of exact solution-preserving rewrites from the initial denotation to the final denotation. This proves the representation statement. A binding moved into σ_* was introduced only when the corresponding equality or exact row-content equation was entailed by the current denotation; composing the preservation chain back to Φ_0 gives entailment. If $\sigma \vDash \Phi_0$, then every grounding of σ is represented by the final configuration. Thus applying σ to the residual variables witnesses $\sigma = \sigma \circ \sigma_*$ on the original variables.

Policy steps are one-way refinements by Lemma 5.1, so the same proof applies after adding the policy commitments to the initial system. □

Corollary 5.10 (Principal model recovered when no caps remain). If the final bound store contains no atom caps and no row caps, then σ_* itself is a least model of Φ_0 in the substitution preorder, modulo the same policy qualification for marker placement.

Proof. With no caps, the residual store contains only adjacencies, which are valid under universally quantified free variables at top or are represented by remaining variables. The representation theorem therefore says $\sigma_* \vDash \Phi_0$, and Theorem 5.9 gives the factoring property for every other model. □

6. Closing

Solving intentionally stops with residual caps. Closing turns a solved configuration into a total ground substitution. This is a policy stage, not a principal unification theorem.

Definition 6.1 (Closing policy).

1. Close leaf variables downward. A leaf dimension gets its saturated atom cap if any, otherwise $\mathbf{1}_\emptyset$; a parameter leaf without a cap errors. A leaf row gets the join of the ground parts of its row caps, holes filled with $\mathbf{1}_\emptyset$, and no further axes beyond cap extent; parameter holes error.
2. Re-solve after those bindings, because stored constraints are re-emitted.
3. Close all remaining interior variables upward to top ($\mathbf{1}_\emptyset$ or the empty row) and re-solve once more.

The row join in step 1 is Proposition 2.7. Hole filling and "no further axes" are policy commitments.

Proposition 6.2 (Closing terminates and is sound). Every re-solve launched by a closing commitment terminates. If closing finishes without error or fail, the resulting total ground substitution satisfies Φ_0.

Proof. A closing commitment binds a variable to a ground value. After such a binding, split the rank abstraction into variable-variable facts and facts with at least one ground side. Ground substitution cannot create new variable-variable rank facts; it only removes variables from constraints or turns constraints into absolute facts. Therefore the variable-variable fragment keeps the rank model supplied by Proposition 5.7 and Lemma 5.3.

The termination proof of Theorem 5.4 needs rank potentials only for lineages whose result side is still open. Those potentials live in the variable-variable fragment. Absolute facts can cause finite failure, but they cannot create the open-open feeding cycle responsible for unbounded growth. Thus each re-solve has finitely many deficits, finitely many mints, finitely many bindings, and terminates by the same pending-constraint measure.

Soundness follows from Lemma 5.1 for each re-solve and from the fact that each committed ground value is then checked by re-emission of its stored constraints. If no re-solve fails, all committed values satisfy the final denotation, which composes back to Φ_0 along the sound direction of Lemma 5.1. □

Example 6.3 (Incomplete but intentional leaf closing). Let

$$\Phi = \{\, 3_b \sqsubseteq \alpha,\ 5_b \sqsubseteq \beta,\ \gamma \sqsubseteq \alpha,\ \gamma \sqsubseteq \beta \,\}.$$

with α and β leaves and γ interior. The store is satisfiable: $\alpha = \beta = \mathbf{1}_\emptyset$ and $\gamma = 3_b$ is one solution. The deterministic leaf policy commits $\alpha = 3_b$ and $\beta = 5_b$. Re-solving then forces γ below two distinct atoms and fails. This is a policy rejection, not semantic unsatisfiability. It is useful behavior for OCANNL

because it avoids silently broadening a leaf tensor with a concrete lower bound unless the user asks for that shape.

Proposition 6.4 (Upward closing greatestness). The uniform-upward solution $\gamma_\uparrow$ of Proposition 5.7 is a solution of the final configuration, and pointwise greatest among its solutions with rows compared by $\preceq$ (Proposition 5.7). Moreover, for every solution of the rank-policy-strengthened initial system, $\gamma(x) \preceq \gamma_\uparrow(x)$ at row sort and $\gamma(x) \sqsubseteq \gamma_\uparrow(x)$ at dimension sort. There is no marked-order ($\sqsubseteq$) strengthening at row sort to state: by the remark after Proposition 5.7, re-placed markers already defeat it over the final configuration, hence over any larger solution set.

Proof. Membership and greatestness for the final configuration are Proposition 5.7. The rank-policy-strengthened initial system maps to the final configuration by Lemma 5.1 and Theorem 5.9.

For the $\preceq$ statement, dimensions are immediate from topness or entailed ground bindings. For rows, if x is unsolved, $\gamma_\uparrow(x)$ is the empty row, and every rigidified flat row is below it. If x is solved by a term T, the flat content of T is entailed for every solution that respects the rank-policy commitments. Induct on T. For a closed row, compare equal-rank rigid flat content pointwise, using the dimension case. For an open row $l \cdot \langle\rho\rangle \cdot r$, the rigid flat content under γ is $\gamma(l) \cdot \mathrm{flat}(\gamma\rho) \cdot \gamma(r)$, while $\gamma_\uparrow(T)$ is $\gamma_\uparrow(l) \cdot \diamond \cdot \gamma_\uparrow(r)$ with the middle expanded by $\mathbf{1}_\emptyset$; flank positions compare by induction and middle positions compare against $\mathbf{1}_\emptyset$. □

The unqualified marked-order greatestness over $\mathrm{Sol}(\Phi_0)$ is false. The single constraint

$$\langle\rho\rangle \approx [] \cdot \diamond \cdot [3, 5].$$

has a solution with $\rho = [3] \cdot \diamond \cdot [5]$, incomparable with the implementation's inherited-split choice.

6.1 Marker Provenance and Surface Discharge

The abstract calculus admits placement-sensitive counterexamples because row equality is flat while row subtyping still observes marker placement on the upper/operand side. The two-constraint witness is

$$\langle\rho\rangle \approx [] \cdot \diamond \cdot [3, 5], \qquad [3] \cdot \diamond \cdot [9, 5] \preceq \langle\rho\rangle.$$

The flat equality permits $\rho = [3] \cdot \diamond \cdot [5]$, which satisfies the inequality, but the deterministic inherited-split policy commits $\rho = [] \cdot \diamond \cdot [3, 5]$, after which the inequality fails.

It is tempting to try to prove that this cannot happen for surface OCANNL programs. That claim should be split into two smaller statements.

Syntactic marker provenance. Every marker position appearing in a surface-generated constraint has a declared syntactic origin:

- external tensor rows and literal shapes are left-marked closed rows (`beg_dims = []`, all axes in `dims`);
- spec axes before an ellipsis become leading flanks and axes after an ellipsis become trailing flanks;
- batch slice prepends the sliced axis as a leading flank;
- substitution preserves provenance by splicing solved row values at the marker;
- closing uses explicit policies: inherited split, cap join, hole fill, or upward empty-row close.

This is an implementation-checkable invariant, and `tensor/shape.ml` supports it: batch slice is the only ordinary operation that manufactures a non-empty leading flank outside spec parsing; permute and einsum obtain leading flanks from axes written before an ellipsis.

Surface discharge. The stronger claim would say that provenance-respecting constraints never observe the solver's marker-placement choice. That is not a purely formal statement about the current core language. It depends on which surface programs are considered meaningful, and in particular on whether a user intended an equality to be a strict inference policy or merely a broadcast-checking condition. A formal proof can only target a specified surface fragment.

The defensible version is therefore conditional:

If a surface fragment uses row equalities only as inference-strengthening constraints for spec matching, and never later compares the same inferred flat content under a distinct declared marker split, then placement policy failures cannot occur in that fragment.

The proof is by provenance induction. Each equality binding inherits a marker split from one of the declared origins above. A later row-subtyping constraint can distinguish another split only when that inferred content appears on the upper/operand side and the same flat content is reintroduced with a different declared origin. Pointwise/compose inequalities reuse tensor-row origins; spec equalities reuse spec ellipsis origins; slice is the only construct that can deliberately shift an origin by prepending a leading axis. Thus slice and axes-before-ellipsis are the cases to audit for any concrete fragment. Without that fragment restriction, the abstract witness above can be encoded directly at the row-constraint level and is a real incompleteness.

7. Projection Inference for the Core

After shape solving and closing, each operation re-derives its own local constraints with fresh projection identifiers. Projection inference turns row and dimension constraints into co-iteration classes and fixed indices for code generation.

The proof in this section covers the core projection language. The current implementation also contains affine, convolution, and concatenation handling; those are staged extensions discussed in Section 8.

Definition 7.1 (Projection language). Projection atoms are:

$\mathrm{Proj}(p)$ is an axis projection id, $\mathrm{Sol}(i)$ is an externally supplied fixed index or symbol.

Core equations are:

$$\mathrm{Eq}(q_1, q_2), \qquad \mathrm{Iter}(q).$$

Each projection id has a solved positive size.

Definition 7.2 (Elaboration from closed core constraints).

- $d_1 = d_2$ emits $\mathrm{Eq}(d_1, d_2)$.
- $d_{\mathrm{res}} \sqsubseteq d_{\mathrm{op}}$ with $\mathrm{size}(d_{\mathrm{op}}) = 1$ emits $\mathrm{Eq}(d_{\mathrm{op}}, \mathrm{Fix}(0))$: the operand broadcasts and is pinned.
- Other $d_{\mathrm{res}} \sqsubseteq d_{\mathrm{op}}$ emits $\mathrm{Eq}(d_{\mathrm{res}}, d_{\mathrm{op}})$.
- $R_1 \approx R_2$ aligns flat rows and emits equality equations per pair.
- $R_{\mathrm{res}} \preceq R_{\mathrm{op}}$ aligns explicit material from the outer edges; surplus result axes are iterated; operand axes of size one are pinned to zero as above.
- Every terminal axis emits $\mathrm{Iter}(\mathit{axis})$.

The side condition for fixed indices is $0 \leq c < \mathrm{size}(\mathit{axis})$.

Definition 7.3 (Projection solver). The core solver maintains:

- a union-find partition of projection ids;
- a partial pin map from classes to fixed indices;
- a set of classes required to iterate.

$\mathrm{Eq}(\mathrm{Proj}(p), \mathrm{Proj}(q))$ unions equal-size classes and fails on size mismatch. $\mathrm{Eq}(\mathrm{Proj}(p), \mathrm{Sol}(i))$ pins the class and fails on conflicting pins. $\mathrm{Eq}(\mathrm{Sol}(i), \mathrm{Sol}(j))$ checks equality. $\mathrm{Iter}(\mathrm{Proj}(p))$ marks the class. At the end, pinned classes use their pin; unpinned iterated classes of size greater than one get fresh iterator symbols; all remaining classes use $\mathrm{Fix}(0)$.

Theorem 7.4 (Projection canonicity). For a finite core projection equation set, the partition, pin map, iteration set, and final labels are independent of processing order. The solver fails exactly when an equality class contains different sizes, receives conflicting pins, or contains a false equality between two solved indices.

Proof. The partition is the least equivalence relation generated by projection-projection equations, restricted by equal-size checks. Union-find processing order cannot change the least equivalence closure. Pins are a partial function

on equivalence classes; order can only change which conflicting pin is observed first, not whether a conflict exists. The iterate set is the union of all classes mentioned by Iter, transported through the same equivalence closure. Labeling is a deterministic function of these three objects, up to bijective renaming of freshly allocated iterator symbols. □

Definition 7.5 (Product space and reductions). For every fresh iterator symbol s_j assigned to a class of size n_j, the product space contains the factor $\{0, \dots, n_j - 1\}$. A factor is a reduction axis iff no component of the left-hand-side index map uses its iterator.

Proposition 7.6 (Coverage and reduction characterization).

1. The LHS index map is injective iff every product factor appears in some LHS component.
2. The LHS index map is surjective onto the LHS index domain iff every LHS axis of size greater than one is iterated and no two such LHS axes use the same iterator.
3. Accumulating read-modify-write is required iff a reduction axis exists. Pre-initialization is required iff the LHS map is non-surjective, or iff an erasing accumulator is used with a non-injective LHS map.

Proof. With only Iter and Fix, two product points differ exactly in at least one product factor. If a factor appears in an LHS component, changing it changes the address; if a factor appears nowhere on the LHS, two product points differing only there map to the same LHS address. This proves injectivity and the reduction characterization.

For surjectivity, an LHS axis of size greater than one cannot be covered by Fix(0); it must be labeled by an iterator ranging over the same size. If two LHS axes share an iterator, the image is diagonal and misses off-diagonal addresses. If every non-unit LHS axis has a distinct iterator, every LHS address is realized by choosing those iterator values and arbitrary values for non-LHS factors. The initialization statement follows from whether every LHS cell is written exactly once, at least once, or potentially multiple times. □

Lemma 7.7 (Local re-solve and locality, core statement). If global shape solving and closing succeed, then for each core operation the per-operation re-derivation against the closed participating shapes has a solution, and all spec variables are eliminated. The resulting projection equivalence depends only on that operation's re-derived constraints.

Proof. The operation's local constraints are the same schema that contributed to the global constraints, but with closed shapes and fresh projection identifiers. Restrict the successful global ground substitution to the participating shapes and rename the local spec variables accordingly; this is a solution of the local shape constraints. At closed shapes, flat row equalities have fixed rank and row variables in specs are pinned by their surrounding concrete rows. Dimension label variables are then unified with closed operand axes. Locality follows be-

cause projection identifiers are freshened per operation, and no constraint from another operation mentions them. □

The proof above is complete for the mathematical core. The implementation bookkeeping in `tensor/shape.ml` should still be audited line-by-line for the full staged language; this is one of the remaining engineering proof obligations, not a known bug.

Theorem 7.8 (Projection soundness). For a closed core operation whose local projection solve succeeds:

1. every generated tensor address is within the solved shape bounds;
2. co-iteration classes are size-uniform;
3. co-iteration is local to the operation.

Proof. For an iterator class, Theorem 7.4 only creates a product factor with the class size; all class members have that size by the failure condition of union. Thus $0 \leq s_j < \text{size}(axis)$ for every iterated axis using that symbol. Pinned $\text{Fix}(c)$ indices satisfy the side condition in Definition 7.2, and unpinned uniterated classes use $\text{Fix}(0)$, which is valid for every positive axis size.

Size-uniformity is exactly the successful projection-projection union check. Leastness of co-iteration follows because the partition is the least equivalence generated by this operation's equations. Locality is Lemma 7.7: fresh projection ids prevent constraints from other operations from entering the closure. □

8. Staged Extensions and Remaining Proof Obligations

The core excludes several constructs used by OCANNL's implementation. The current status is:

Proposition 8.1 (Padding fold canonicity). The padding margin accumulated for an axis is the maximum over finitely many operation-local padding demands. Because max is associative, commutative, and idempotent, the accumulated margin is independent of processing order. □

Open 8.2 (Affine and convolution projection determinacy). The implementation evaluates affine and convolution projection terms after the core classes are known. The expected proof is stratified: core classes are canonical by Theorem 7.4, derived affine terms are deterministic functions of those classes, and deferred affine equality checks are pure checks. The proof still needs the exact equations for valid convolution and size-preserving padding modes and an invariant saying generation, stored terms, and final checking use the same mode.

Open 8.3 (Concat projection determinacy). The implementation has a connected-component style coupling for Concat projections. A proof should show that structural concat equations generate an order-independent closure and that concat targets receive deterministic composite indices after their components are

solved. This is plausible from the current `solve_proj_equations` structure, but it is not covered by the core Theorem 7.4.

Open 8.4 (Two-unit shape closing). Full OCANNL has discardable variables that can close to a size-zero additive unit, while ordinary broadcast holes close to $\mathbf{1}_\emptyset$. Soundness should follow if the discardability test proves that every zero-closed occurrence is projected away. The interaction with row instantiation remains to be formalized.

Open 8.5 (Relative completeness of staged solving). Unrestricted completeness is not expected. The right theorem is relative: characterize the surface fragment where stage triggers guess only forced or solution-irrelevant values, and exhibit counterexamples just outside it.

9. Counterexamples to Broad Completeness

Rather than trying to prove a broad relative-completeness theorem, the current evidence points to a boundary: OCANNL's deterministic policies are useful, but there are small programs or solver inputs where the intended solution is clear and the solver either rejects it or requires an additional user constraint.

9.1 Periodic Row-Permutation Shape

The probe in `test/einsum/test_inference_counterexamples.ml` compares two programs:

```
let%op x = { x = uniform1 (); o = [ 3; 5; 3 ] } in
let%op y = x + (x ++ "a,b,..r.. => ..r..,b,a" [ "a"; "b"; "r" ])
```

and

```
let%op x = { x = uniform1 () } in
let%op y = x + (x ++ "a,b,..r.. => ..r..,b,a" [ "a"; "b"; "r" ]) in
Shape.set_dim a 3;
Shape.set_dim b 5
```

The concrete program succeeds: the input has shape `[3,5,3]`, so the row variable can be `..r.. = [3]` and the permutation is shape-preserving. The abstract-parameter version currently fails with `Shape_error: solved dimensions for axis: mismatch`, even though the same solution is evident: infer the parameter shape as `[3,5,3]`.

This is not a marker-provenance failure. It is the RE-same-rot policy showing through a surface program. The equation induced by adding the tensor to its cyclic permutation has a non-empty periodic solution for `..r..`; the deterministic solver's least-material closure does not search that periodic family.

A user can make the program inferable by providing more shape information, for example by using a concrete input shape or otherwise pinning the row. That is exactly the intended relative-completeness boundary: the code conveys a

natural intent, but the current inference policy does not solve the associated word equation.

9.2 Leaf Downward Closing

The lower-level probe in `test/einsum/test_closing_order.ml` contains the dimension store

$$3_b \sqsubseteq \alpha, \qquad 5_b \sqsubseteq \beta, \qquad \gamma \sqsubseteq \alpha, \qquad \gamma \sqsubseteq \beta.$$

with α and β treated as leaves. The store is satisfiable by raising at least one leaf to the broadcast top, for example $\alpha = \beta = \mathbf{1}_\emptyset$. Deterministic leaf closing instead commits the capped leaves downward to $\alpha = 3_b$ and $\beta = 5_b$, then fails when γ is forced below two incompatible atoms.

This is not a likely tensor-literal program when the leaf shapes come from actual arrays, because actual array extents are exact. It is nevertheless a real policy counterexample for generated or parameter-like leaves whose caps are lower bounds rather than exact facts. The user's intent can be clear in contexts such as "choose a common broadcastable shape for these two partially specified leaves"; deterministic downward closing chooses parsimony instead.

9.3 Cross-Surplus and Placement Policy

`test/einsum/test_row_self_reference.ml` records a row-level mirror cross-surplus case that is flat-satisfiable but rejected by the implementation because the inherited split and the conservative trailing guard select the wrong representative. This remains a row-constraint counterexample rather than a confirmed ordinary `%op` counterexample. To lift it to the surface, one would need a spec or slice program that both:

1. pins a row variable's flat content through a split-surplus equality; and
2. later compares the same flat content under a different declared marker split.

That is precisely the marker-provenance audit target from Section 6.1.

10. Implementation Correspondence

The current implementation aligns with the core formalism in the following places:

- `tensor/row.ml` represents rows as `beg_dims`, `dims`, and `bcast`, exactly the two flanks plus marker.
- `unify_row` compares closed rows by flattened content (`beg_dims @ dims`), matching $\approx$.

- Open-vs-closed row equality matches explicit open flanks against the closed flat list and then commits an inherited split for the bound row variable.
- Shifted same-variable equality and row-subtyping constraints are deferred until another constraint solves the variable or stage-6 upward closing supplies the empty middle.
- `solve_row_ineq` records a persistent rank graph with non-negative edges for RG-bind, RG-equal-flanks, and RG-deficit. It deliberately records no edge for the RI-cap surplus case.
- Closed-closed row-subtyping compares the operand's explicit leading and trailing material against the result's flat row from the outer edges, so shifted explicit axes are checked rather than silently accepted.
- `tensor/shape.ml` emits row-subtyping constraints for pointwise, transpose, and compose-style broadcasting, and emits row equalities for batch slice, permute, and einsum templates.
- Batch slice prepends the slice dimension to `beg_dims`, confirming the formal motivation for leading flanks.
- Projection identifiers are freshened for solved dimensions before local projection solving, supporting the locality theorem for the core.

The code also implements constructs outside this report's core: affine dimensions, convolution padding, concatenation, total element constraints, discardable zero dimensions, and staged closing heuristics. These are not known to contradict the core formalism, but the full soundness theorem for them remains future work as described in Section 8.

11. Summary of Proved and Open Results

Proved in this report:

- dimension lattice facts and non-distributivity;
- row partial order, top, joins, and expansion monotonicity;
- incompatibility of flat equality with the marked order;
- two-sorted rigid-row order and row subtype/refinement relation;
- solution preservation for all semantic dimension and row rules;
- soundness and exactness of the rank graph for recorded facts;
- termination for finite inputs under the persistent rank-cycle guard;
- the Detection Lemma for rank-unsatisfiable divergent runs;
- satisfiability and greatestness of the residual solved store;
- representation and most-generality for semantic runs, policy-qualified for deterministic runs;
- termination and soundness of closing;
- core projection canonicity, coverage, and address safety.

Open or deliberately incomplete:

- the syntactic marker-provenance invariant is plausible and implementation checkable, but the broader surface-discharge claim is only meaningful after

choosing a precise surface fragment;

- unconditional completeness for RE-cross and RE-same-rot, unless the solver is extended with finite branching over the relevant word-equation choices;
- full implementation-level audit of per-operation re-derivation for the staged language;
- soundness and determinacy proofs for affine, convolution, concatenation, total-element, and zero-discard extensions;
- broad relative completeness of the staged closing heuristics; Section 9 records current counterexamples instead.