Operator-Level Boolean Computation with Correspondence Matrices

Typed operand frames, local operator folding, fallback semantics, and empirical boundaries

Brian Theory

Compiler contract and evidence revision — 26 September 2026
Semantic-boundary addition and audit — 13 October 2026

Abstract

Correspondence matrices encode a binary Boolean connective as four bits in a declared operand frame. We specify a restricted compiler that normalizes signed and swapped variable occurrences, fuses tokens with matching frames, and distinguishes structural, tabulated, and hybrid derivations. An ordered strategy selects among those admissible derivations; failure of the selected strategy need not mean that no derivation exists. Soundness and completeness for an explicitly generated structural fragment are established using classical truth-function identities. The implementation contract separates syntactic variable occurrence from essential dependence, root pair success from discarded child constructions, and compact tokens from dense outputs over retained ambient axes. A pinned source snapshot and executable regression suite make these boundaries checkable. Recovered S1/S2 records show local reductions but a generic-optimizer tie; a separate ANF-dispatch study is negative overall. A prospectively frozen synthetic pair-token comparison on 64 held-out signed formulas also fails its primary utility rule: median direct-packed/CM production time is 0.248 (95% hierarchical bootstrap interval 0.246–0.249). A separate correctness screen of 71 admitted original EPFL primary-output cones found four pure structural roots, one fully retabulated root, and 66 fallbacks. In a subsequently frozen descriptive secondary timing run, 70 cones completed paired full-bitset Python timing and one failed calibration; the complete-case median direct-packed/CM producer ratio was 0.0376, with direct evaluation faster on every complete case. ABC/AIG truth checking completed for all 71, but its subprocess wall time is not a matched producer comparison. The broader protocol-v3 systems comparison remains open. The contribution is a bounded compiler specification and reproducible negative systems boundary, not a new truth-table algebra or a demonstrated performance advantage.

Draft status. This is the revised compiler-focused manuscript after the deliberate CM/LM publication split. The companion foundations paper *Correspondence and Logical Matrices: A Boolean Operator Calculus* owns the formula-valued LM theory, valuation, logical pairing, higher-arity tensors/maps, and the broader coherence calculus [21]. The present paper imports only the numeric binary contract required by the compiler. A narrower synthetic protocol-v4 confirmation is complete and negative. A natural-circuit admission/correctness gate and a separate, incomplete descriptive secondary timing run have since completed; the full timed protocol-v3 comparison has not.

1 Introduction and claim boundary

A binary Boolean connective contains four output values. A correspondence matrix stores those four values as a 2×2 array whose row and column axes are tied to declared operands. In isolation, that is only a reshaping of a truth table. The computational question is different: can a compiler use the small operator as a first-class token, normalize the token's operand frame, and combine compatible local operators before evaluating or expanding their operands?

The key example is deliberately simple. Consider

$$(X \Rightarrow Y) \Updownarrow (Y \Rightarrow X). \tag{1}$$

Both inner connectives are implication, but they do not initially refer to the same ordered operand frame. Treating their four-bit arrays as if corresponding entries described the same assignment would be a type error. Once the second occurrence is transported from frame (Y, X) to frame (X, Y) by transpose, entrywise XOR is sound:

$$[\Rightarrow] \mathbin{\widehat{\Updownarrow}} [\Rightarrow]^T = [\Updownarrow]. \tag{2}$$

The resulting token represents $X \Updownarrow Y$. Equation (2) is not important because XOR of two truth tables is difficult; it is important because it makes the *admission condition* explicit. A local table may be fused only when its coordinates have a common logical meaning.

This paper therefore treats operand-frame metadata as part of the compiler type. A successful pair compilation returns not just four bits, but a token tied to canonical row and column operands. Swaps and input polarities are normalized by fixed reindexings. Compatible tokens can then be fused by an outer Boolean connective before repeated operand work or polynomial expansion. If structural recognition fails, the implementation can exactly retabulate an admitted two-variable subtree, can combine such a retabulated descendant with structural parents while marking the result hybrid, or can pass the original expression to the ordinary builder. Successful child pair constructions are not retained when the root pair attempt fails.

The component operations have substantial prior art. Boolean matrices predate this work [5]; matrix and vector logics represent logical operators algebraically [19, 14, 15]; Cheng, Zhao, and Xu explicitly combine truth vectors entrywise under an arbitrary outer Boolean operation [3]; small truth functions, NPN transformations, LUTs, and DAG-aware rewriting are standard synthesis techniques [7, 23, 20]; and bit-parallel evaluators compute several assignments simultaneously [6]. Bricken's internally dated 1997 technical note is also a close antecedent for 2×2 logical matrices and bra–matrix–ket evaluation [1]. The present paper does not claim those ingredients as new.

The narrower contribution is computational and testable:

1. a **typed frame-aware compiler rule** in which a four-bit CM token is paired with operand metadata and normalized before same-frame fusion;
2. **sound multi-path semantics** separating pure structural compilation, local retabulation, hybrid composition, and ordinary fallback;
3. an **executable implementation contract** separating compact tokens, framed surrogates, general IR, lowered programs, and ambient dense outputs, with regression tests for strategy selection, fixed axes, and discarded child constructions;
4. **matched mechanism and negative evidence**: exploratory folding can reduce pre-expansion work, an equally capable generic optimizer can reproduce the same reductions, and frozen ANF-dispatch and synthetic pair-token studies show that local wins need not amortize; and
5. a **preserved broader evaluation design** with natural admission, correctness, and bounded secondary producer timing now checked, while synthesis comparability, reuse, alternative outputs, memory, and fallback costs still require matched measurement.

The scientific claim does not depend on CM folding being uniquely capable of a reduction. If a generic local truth-function optimizer reaches the same reduced form, that locates the mechanism in a broader family of small-function rewrites. The remaining CM-specific question is whether the representation and its frame metadata make recognition, legality, transformation, caching, or reuse sufficiently direct and inexpensive to matter in a full compiler.

2 Minimal CM compiler contract

The companion foundations paper develops the CM/LM calculus in detail [21]. Only the following binary numeric contract is required here.

Let

$$\mathbb{B} = \{0, 1\} \cong \mathbb{F}_2, \tag{3}$$

with multiplication interpreted as conjunction and addition written as XOR, $\Updownarrow$. For a bit $x \in \mathbb{B}$, use the true-first state

$$|x\rangle = \begin{bmatrix} x \\ \neg x \end{bmatrix}, \qquad \langle x| = \begin{bmatrix} x & \neg x \end{bmatrix}. \tag{4}$$

For a binary logical connective variable Θ, define the compact correspondence matrix

$$[\Theta] := \begin{bmatrix} \Theta_{11} & \Theta_{10} \\ \Theta_{01} & \Theta_{00} \end{bmatrix} = \begin{bmatrix} 1\Theta 1 & 1\Theta 0 \\ 0\Theta 1 & 0\Theta 0 \end{bmatrix}. \tag{5}$$

The row and column assignment order $(1, 0)$ is part of the representation contract.

For $C \in \mathbb{B}^{2\times 2}$, define the XOR–AND contraction

$$\mathcal{E}_C(x, y) = \langle x|C|y\rangle = \bigUpdownarrow_{r,s\in\{1,0\}} \chi_x(r) \wedge C_{rs} \wedge \chi_y(s), \tag{6}$$

where $\chi_x(r) = 1$ exactly when $r = x$.

Proposition 1 (Selection)**.** *For every binary connective* Θ *and* $x, y \in \mathbb{B}$*,*

$$\langle x|[\Theta]|y\rangle = x\Theta y. \tag{7}$$

Proof. Both states are one-hot. Every term in Equation (6) vanishes except the term at $(r, s) = (x, y)$. □

The compiler packs the four coefficients into a constant-size token. That token is not by itself sufficient to authorize a rewrite. An occurrence is also typed by an *operand frame*: the row expression, column expression, their signed polarities, and the truth-state order. In the implementation considered here, a successful pair surrogate ultimately records canonical row and column variable names together with the normalized token.

3 Typed normalization and same-frame fusion

Let

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = [\Updownarrow]. \tag{8}$$

Define $\Theta^\tau(x, y) := y\Theta x$. With the true-first convention,

$$[\Theta^\tau] = [\Theta]^T, \tag{9}$$

$$[\Theta(\neg X, Y)] = P[\Theta], \tag{10}$$

$$[\Theta(X, \neg Y)] = [\Theta]P, \tag{11}$$

where products with P are XOR–AND products and therefore act as row/column permutations. Output negation is entrywise complement.

For a binary outer connective Φ, define same-position fusion by

$$\left([\Theta_1]\widehat{\Phi}[\Theta_2]\right)_{rs} := (\Theta_1)_{rs}\Phi(\Theta_2)_{rs}. \tag{12}$$

This is not matrix multiplication.

Theorem 1 (Same-frame fusion)**.** *If* $[\Theta_1]$ *and* $[\Theta_2]$ *are expressed in the same ordered operand frame, then for all* $x, y \in \mathbb{B}$*,*

$$(x\Theta_1 y)\Phi(x\Theta_2 y) = \left\langle x\middle|\left([\Theta_1]\widehat{\Phi}[\Theta_2]\right)\middle|y\right\rangle. \tag{13}$$

Proof. Both sides select position (x, y); the right-hand side stores the outer operation on exactly those two selected coefficients. No distributive law for Φ over XOR is required. □

The pointwise identity itself is classical; Cheng, Zhao, and Xu give the direct truth-vector antecedent in Proposition 3.3 [3]. The compiler-specific rule is therefore:

> **Admission rule.** Transport signed/swapped occurrences to one declared frame. Apply Equation (13) only after row operand, column operand, polarity interpretation, and truth-state order agree.

Figure 1 summarizes the resulting local pipeline.

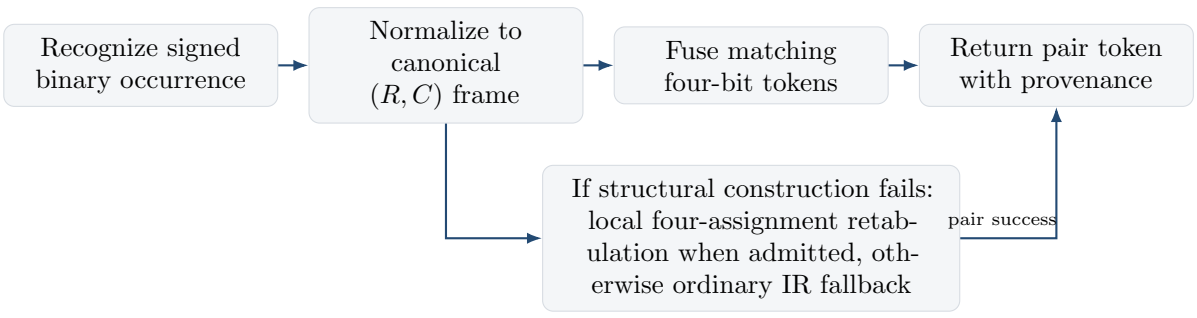


Figure 1: The compiler transformation. The operation performed on the four-bit token is trivial; the semantic work is the typed recognition and normalization that determines when fusion is legal and what fallback is required.

3.1 Why frame metadata matters

Return to Equation (1). The two implication tokens are numerically equal before typing, but the second occurrence's array coordinates refer to (Y, X) rather than (X, Y). Naively XORing the two unaligned arrays would return the zero token. Transposition moves the second operator into the target frame, after which Equation (2) correctly yields $[\Updownarrow]$. The error is therefore not a deficiency of truth tables; it is a failure to track which assignment each stored bit denotes.

This distinction is closely related to established NPN-style input permutation/negation and output-negation operations [7, 23]. The present use is intentionally narrower than a general NPN canonicalizer. It treats frame transport as a local typing obligation for a particular rewrite.

4 Pair compiler and soundness

4.1 Judgment, invariant, and provenance

Let expressions use the grammar

$$e ::= Z \mid \neg e \mid e \,\Phi\, e, \tag{14}$$

where Z is a variable and Φ is one of the currently supported binary connectives AND, OR, XOR, implication, and equivalence. Fix distinct canonical variables R and C and a fixed-substitution map F that binds neither variable on a returned pair path. Write $\Gamma = (R, C, F)$.

Let $\mathrm{Vars}(e)$ collect variable names occurring in the source syntax, recursively by singleton, identity through negation, and union through a binary node. Define

$$\mathrm{supp}_F(e) := \mathrm{Vars}(e) \setminus \mathrm{dom}(F). \tag{15}$$

This is *syntactic unfixed occurrence*, not essential functional dependence. No Boolean simplification is implicit: $X \vee (Y \Updownarrow Y)$ still has support $\{X, Y\}$ when F is empty, although its value depends only on X. Fixed names are excluded from this set and their values are supplied during evaluation.

An admissible typed pair derivation has judgment

$$\Gamma \vdash e \Downarrow^p \mathrm{Pair}(R, C, t), \qquad p \in \{S, T, H\}, \tag{16}$$

with the intended invariant

$$\mathcal{E}_t(r, c) = e[r/R, c/C] \quad \text{for all } r, c \in \mathbb{B}, \tag{17}$$

interpreted after the fixed substitutions in F.

Equation (16) can be read as follows: under compiler environment Γ, expression e admits a derivation of the four-bit token t in canonical frame (R, C), and p records how that token was obtained. Equation (17) is the semantic contract. Each of the four token positions must return exactly the value of the original expression under the corresponding assignment to R and C. The pair therefore carries both a compact Boolean function and the frame metadata needed to interpret its bit positions.

The provenance values are:

- S — **structural:** the returned subtree used signed primitive recognition, token complement, and/or same-frame fusion only.
- T — **tabulated root:** the returned subtree itself was produced by evaluating all four assignments. The symbol T avoids overloading R, which already denotes the row variable.
- H — **hybrid:** structural steps occur above at least one tabulated descendant.

The judgment is a relation, not a deterministic strategy. For example, $X \wedge Y$ admits both an S primitive derivation and a T tabulation derivation, with the same token 1000_2. Section 5.2 specifies which derivation is selected. The token API returns `None` with diagnostic `ordinary_fallback` if the selected procedure returns no pair; it does not itself execute the ordinary builder.

4.2 Rules, built up one step at a time

The inference-rule bar is read as implication: if the premises above the bar hold, the compiler may derive the conclusion below it. The rules perform four conceptually distinct jobs: create a normalized primitive token, complement an already compiled token, fuse two already aligned tokens, and tabulate an exact two-variable subtree. Their validity is independent of the order in which an implementation attempts them.

Primitive normalization. Let ℓ_1, ℓ_2 be signed literals whose underlying variables are exactly R and C in either order. If the declared normalization procedure transports the primitive occurrence $\Theta(\ell_1, \ell_2)$ into the canonical frame and returns token t, then the compiler may derive

$$\frac{\mathrm{Norm}_\Gamma(\Theta(\ell_1, \ell_2)) = t}{\Gamma \vdash \Theta(\ell_1, \ell_2) \Downarrow^S \mathrm{Pair}(R, C, t)}.$$

Normalization consists only of the declared operand exchange and row/column polarity transforms from Equation (11). This is the structural base case: it establishes a token whose four bit positions already refer to the canonical assignments $(R, C) = (1, 1), (1, 0), (0, 1), (0, 0)$.

Negation: Equation (18). Once a child has a correct token in the canonical frame, negating the expression requires only bitwise complement of that token:

$$\frac{\Gamma \vdash a \Downarrow^p \mathrm{Pair}(R, C, t_a)}{\Gamma \vdash \neg a \Downarrow^{\nu(p)} \mathrm{Pair}(R, C, \neg t_a)}. \tag{18}$$

The frame does not change. Here $\neg t_a$ means complement each of the four token bits. Provenance changes only to report ancestry: $\nu(S) = S$ and $\nu(T) = \nu(H) = H$. Thus a structural NOT above a locally tabulated child is correct but explicitly reported as hybrid.

Same-frame fusion: Equation (19). Two compiled children may be fused only when their pair metadata name the same canonical frame:

$$\frac{\Gamma \vdash a \Downarrow^p \mathrm{Pair}(R, C, t_a) \qquad \Gamma \vdash b \Downarrow^q \mathrm{Pair}(R, C, t_b)}{\Gamma \vdash a \Phi b \Downarrow^{p \sqcup q} \mathrm{Pair}(R, C, t_a \widehat{\Phi} t_b)}. \tag{19}$$

The token operation $t_a \widehat{\Phi} t_b$ applies Φ independently to corresponding token positions. The repeated $\mathrm{Pair}(R, C, \cdot)$ in the two premises is the essential admission certificate: position 11 in one child denotes the same assignment as position 11 in the other, and similarly for $10, 01, 00$. Define $S \sqcup S = S$; every other combination is H because tabulated ancestry is present below a structural fusion.

Exact local tabulation. If the syntactic unfixed support $\mathrm{supp}_F(e)$ is exactly $\{R, C\}$, define the true-first token

$$t_e := \big(e(1,1), e(1,0), e(0,1), e(0,0)\big). \tag{20}$$

Then the exact tabulation rule is

$$\frac{\mathrm{supp}_F(e) = \{R, C\} \qquad t = t_e}{\Gamma \vdash e \Downarrow^T \mathrm{Pair}(R, C, t)}.$$

This rule re-evaluates a two-variable subtree on its complete four-assignment domain. It is an exact semantic base case. Its rule has no premise requiring structural failure; that restriction belongs to the ordered hybrid procedure.

Worked derivation. Let $\Gamma = (X, Y, \varnothing)$ and

$$e = (X \Rightarrow \neg Y) \Updownarrow (\neg Y \Rightarrow X). \tag{21}$$

In true-first order $(11, 10, 01, 00)$, implication has token 1011_2. For the left child, the variables are already in (X, Y) order but the column literal is negated, so the column-polarity transform gives

$$X \Rightarrow \neg Y \quad \longmapsto \quad 0111_2.$$

For the right child, the source operands arrive as $(\neg Y, X)$. Operand exchange followed by polarity alignment gives

$$\neg Y \Rightarrow X \quad \longmapsto \quad 1110_2.$$

Both children now carry frame (X, Y), so Equation (19) is legal:

$$0111_2 \ \Updownarrow \ 1110_2 = 1001_2. \tag{22}$$

Hence

$$\Gamma \vdash e \Downarrow^S \mathrm{Pair}(X, Y, 1001_2).$$

The token 1001_2 is $[\Leftrightarrow]$ in true-first order, so the compiler has established

$$(X \Rightarrow \neg Y) \Updownarrow (\neg Y \Rightarrow X) = X \Leftrightarrow Y$$

without first expanding the source expression into an ANF or another larger symbolic representation. Applying Equation (18) once more complements 1001_2 to 0110_2, the XOR token.

An S derivation uses no tabulation; a T derivation ends with tabulation; an H derivation uses structural composition above a tabulated descendant. These are derivation properties. In particular, the strategy named `hybrid` can select an S, T, or H result, depending on the expression.

Theorem 2 (Pure structural soundness)**.** *If $\Gamma \vdash e \Downarrow^S \mathrm{Pair}(R, C, t)$, then Equation (17) holds and no four-assignment tabulation was used.*

Proof. Induct on the derivation. A signed primitive is correct by the frame transforms in Equation (11); complementing a correct child represents negation; and fusing two correct children with identical metadata is sound by Theorem 1. □

Definition 1 (Declared pure-structural fragment)**.** Fix canonical variables R, C. Let $\mathcal{S}_{R,C}$ be the smallest expression class containing every supported primitive $\Theta(\ell_1, \ell_2)$ whose signed literals have underlying variables exactly R, C in either order, and closed under negation and the five supported binary connectives.

Theorem 3 (Completeness for the declared pure-structural fragment)**.** *For every $e \in \mathcal{S}_{R,C}$, the `pure_structural` strategy derives a unique canonical-frame token t such that*

$$\Gamma \vdash e \Downarrow^S \mathrm{Pair}(R, C, t)$$

and Equation (17) holds.

Proof. Induct on the construction of e. Every primitive in $\mathcal{S}_{R,C}$ is admitted by primitive normalization. Negation preserves the frame and remains structural by Equation (18). If two children belong to $\mathcal{S}_{R,C}$, induction gives tokens in the same canonical frame, so Equation (19) applies. Uniqueness follows because a Boolean function on fixed variables R, C has one four-bit truth token in the declared true-first order. □

Remark 1. This is syntactic completeness for the deliberately declared compiler fragment, not semantic completeness for all formulas with two-variable support. A semantically equivalent expression outside $\mathcal{S}_{R,C}$ may still require exact tabulation or ordinary fallback.

Lemma 1 (Exact pair tabulation)**.** *Suppose the syntactic unfixed support $\mathrm{supp}_F(e)$ is exactly one distinct row variable R and one distinct column variable C. Evaluating e at $(1,1), (1,0), (0,1), (0,0)$ and storing those four outputs in true-first order produces the unique token satisfying Equation (17).*

Theorem 4 (Soundness of admitted pair compilation)**.** *Every derivable pair result with provenance $p \in \{S, T, H\}$ satisfies Equation (17). Separately, if the selected procedure returns no pair, the dense wrapper delegates correctness to the unchanged ordinary IR evaluator on the original expression, assuming that evaluator's contract.*

Proof. The cases S and T are the preceding theorem and lemma. For H, use structural induction with exact local tabulation as an additional sound base case; token complement and same-frame fusion preserve the invariant. Provenance changes the reported construction path, not the represented function. The fallback statement is an interface contract rather than a theorem about the pair compiler itself. □

4.3 Admissibility boundary

The current pure structural rule does not claim semantic recognition of arbitrary equivalent compound operands. Repeated-variable forms such as $X \Theta X$, opaque compound operands, and incompatible variable pairs remain outside the pure rule. Duplicate axes and overlapping row/column layouts are invalid inputs and raise an error; they are not ordinary pair misses. A retabulation rule may cover some two-variable cases exactly, but it is intentionally classified as a different mechanism.

This conservative boundary is useful. It allows the compiler to refuse a local rewrite rather than infer an unproved equivalence. It also gives the experiment a measurable fallback frequency instead of hiding unsupported cases in preprocessing.

4.4 Semantic boundary under real-valued relaxation

The invariant in Equation (17) is Boolean. A real-valued interpretation requires a separate contract. For a truth table $C = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in \mathbb{B}^{2\times 2}$ and $p, q \in [0,1]$, ordinary real contraction gives

$$B_C(p,q) = [p, 1-p]C[q, 1-q]^T = d + (b-d)p + (c-d)q + (a-b-c+d)pq. \tag{23}$$

For independent Bernoulli inputs this is the probability of the represented Boolean event. More generally, let e be a finite propositional expression over n distinct base-event identities and let B_e be the unique multilinear polynomial interpolating its Boolean truth table. Under the product Bernoulli model with marginal vector p, $B_e(p) = \Pr_p(e = 1)$. For a correlated model μ, the probability is instead $\sum_x e(x)\mu(x)$; marginals alone need not determine it. Weighted model counting supplies the established model-sum interpretation, and semantic loss uses the negative logarithm of the satisfying-assignment probability [24, 26]. These are ordinary real operations, distinct from the XOR–AND contraction.

This whole-table interpretation need not agree with recursively applying soft kernels to subexpressions with shared variables. PALO's approximate propositional semantics uses pq for conjunction, $p + q - pq$ for disjunction, $1 - p + pq$ for implication, and $1 - p - q + 2pq$ for primitive equivalence; it explicitly recognizes the independence approximation and the failure of several classical laws [25, Sections 3.2, 3.5–3.6]. In particular, the Boolean fusion already established in Equation (22),

$$(X \Rightarrow \neg Y) \Updownarrow (\neg Y \Rightarrow X) \equiv X \Leftrightarrow Y,$$

does not preserve recursive soft evaluation with negation $1 - p$ and XOR kernel $u + v - 2uv$. At $p = q = 1/2$, both implications have value $3/4$, so the source has value $3/8$, whereas the fused equivalence has value $1/2$. This compares the declared propositional kernels, not PALO's full quantified, batched, or learning semantics.

For expressions built from the bilinear kernels in Equation (23) and negation $1 - p$, let s_e be the recursive polynomial with one free coordinate per distinct base-event identity. A sufficient condition for $s_e = B_e$ is disjoint syntactic base-variable supports at every binary composition. More generally, equality on the full cube $[0,1]^n$ holds exactly when s_e is affine separately in every coordinate, after coefficient cancellation. Both polynomials agree at the Boolean corners; uniqueness of multilinear interpolation proves sufficiency, and equality on the full cube implies polynomial identity, proving necessity. This full-cube criterion need not be necessary on a restricted parameter image. Distinct independent events may share a parameter t: their conjunction then has probability t^2, whereas repeating one event gives probability t. Event identity must therefore be kept separate from parameter identity. Polynomial expansion can grow exponentially; same-frame metadata alone does not establish this additional contract.

A future numerical backend should distinguish Boolean equivalence, equality of the recursive soft function, almost-sure event equality under a specified joint model, and thresholded-decision equality on a declared domain. Equal marginals alone do not authorize contextual replacement: independent fair events X, Y both have probability $1/2$, but $\Pr(X \wedge X) = 1/2$ and $\Pr(X \wedge Y) = 1/4$. Almost-sure equality permits replacement in Boolean contexts evaluated under that same model. A decision contract must specify its threshold, tie rule, and placement. PALO inserts crispification at each propositional interpretation [25, Section 3.7]; at threshold $1/2$ with a $\geq$ tie rule, inputs $p = q = 3/5$ give crisp conjunction 1, while thresholding their final soft product $9/25$ gives 0.

For a common differentiable parameterization $p = \phi(\theta)$, identity $s_e \circ \phi = s_f \circ \phi$ on an open parameter domain is sufficient for equality of mathematical parameter gradients there. Equality only at selected points or on a lower-dimensional input set does not guarantee equality of ambient input gradients: p and pq agree on $q = 1$, but have gradients $(1, 0)$ and $(1, p)$ there. Function equality is not necessary for gradient equality; an additive constant is a counterexample. Custom backward rules and floating-point evaluation need separate contracts. These distinctions bound possible extensions of this Boolean compiler; no integrated differentiable backend, training improvement, or CM-specific advantage is claimed. Ordinary framed truth tables support the same checks.

5 Implementation architecture and cost model

5.1 Five artifacts, not one

Table 1 separates implementation objects that are easily conflated when all are described informally as a "CM."

A token-only endpoint cannot be compared as if it had already paid to construct an explicit 2^m-entry ambient result. Conversely, a comparator that has already prepared a reusable DAG or program should not be charged

Artifact	Meaning	Role
CM operator token	One binary truth function packed into four true-first bits $(11, 10, 01, 00)$.	Constant-size transform and fusion.
Pair surrogate	Token plus canonical row and column operand names/frame metadata.	Carries the legality invariant.
General CM/Boolean IR	Interned expression DAG for arbitrary supported formulas.	General symbolic fallback and compilation.
Flat program	Backend-dependent instruction sequence lowered from the IR.	Repeated packed/array evaluation.
Explicit dense output	Materialized truth relation over all retained ambient row/column axes.	2^m entries for m retained axes; one byte per NumPy Boolean entry.

Table 1: Compiler artifacts and their semantic boundaries. Only the first is the constant-size binary operator token.

for that preparation on every query when the CM arm is allowed resident reuse. Endpoint equality is part of benchmark correctness.

5.2 Ordered, strategy-dependent selection

The source snapshot is pinned to commit `0ab8ffd0c23ffa71ee951d375b1c170ffdcc084b` of the public implementation [10]. Let $\mathcal{R}, \mathcal{C}$ be ordered ambient row/column name lists, each duplicate-free and mutually disjoint. A pair frame consists of one unfixed name $R \in \mathcal{R}$ and one unfixed name $C \in \mathcal{C}$; these names are discovered from the expression, not chosen by lexicographic order. The full supported AST is validated and its occurrence, identity-node, and height statistics are computed before strategy selection. The intended input domain supplies every occurring name through a layout or F; other errors are not silently converted into pair misses.

```
Pair(e, row_layout, column_layout, F, allowRetab):
    if e is a supported binary connective over signed literals
       with distinct unfixed names on opposite layout sides:
        normalize operand order and polarities
        return Pair(R, C, token, S)
    if e = NOT a:
        child = Pair(a, ..., allowRetab)
        if child succeeds: return its complement, with nu provenance
    else if e = a Phi b:
        left  = Pair(a, ..., allowRetab)
        right = Pair(b, ..., allowRetab)  # attempted even if left fails
        if both succeed on the same frame:
            return their fusion, with joined provenance
    else if e is a variable: return NoPair
    else: raise unsupported-node error
    if allowRetab and supp_F(e) has exactly one row and one column name:
        return four-assignment token with provenance T
    return NoPair

Select(e, strategy):
    pure_structural -> Pair(e, ..., allowRetab=false)
    hybrid          -> Pair(e, ..., allowRetab=true)
    retabulate      -> tabulate root if support admits a pair, else NoPair
```

The compatibility name `structural` aliases `hybrid`, not `pure_structural`. For valid input and fixed strategy, the procedure selects one outcome. Every successful return supplies a derivation of Equation (16): this follows by induction over the executed recursive calls, using the corresponding inference rule at each return. Thus the relational soundness theorems apply to the selected result. A `NoPair` return means failure of that strategy, not non-derivability in the entire relation. For example, $X \vee (Y \wedge Y)$ has a T derivation but the pure strategy returns `NoPair`. Recursion terminates on finite acyclic ASTs/DAGs; the recursive pair walk can revisit identity-shared

nodes.

Selection, diagnostics, and commitment. The public token artifact stores the two names and token; provenance is returned in accompanying metrics. Selected S, T, H map respectively to `pure_structural`, `full_retabulation`, and `hybrid_pair`. `None` maps to `ordinary_fallback`, even in the token-only API, where no builder is called. The dense wrapper first checks the ambient output budget, then obtains the token result. On success it lifts that token to the ambient layout; on failure it calls the ordinary builder on the *original expression*.

Consequently, $e = (X \wedge Y) \vee Z$, with no fixed variables and layouts $[X, Z]$ and $[Y]$, constructs an S child for $X \wedge Y$ but returns no root pair. The successful child is discarded when the dense wrapper invokes the builder. The builder may independently optimize the original expression; this wrapper does not pass it a partially rewritten tree. Likewise, a root retabulation can supersede earlier child attempts. Counters such as `pair_collapses` measure local work performed, including discarded work, not committed rewrites in the returned object. Root outcome must accompany these counters; committed contributions require separate instrumentation. Persistent child rewriting would be a different compiler version.

5.3 Layout conversion and ownership

The compact token uses true-first order while the historical dense array path uses false-first axes. The conversion is an explicit reindexing. If T denotes the true-first 2×2 view and D the false-first view indexed by array positions $a, b \in \{0, 1\}$, then

$$D_{ab} = T_{1-a,1-b}. \tag{24}$$

The dense contract retains *every supplied ambient axis*, including fixed names and names on which the residual function is independent. Writing $m = |\mathcal{R}| + |\mathcal{C}|$, the returned shape is

$$2^{|\mathcal{R}|} \times 2^{|\mathcal{C}|}, \qquad \text{with } 2^m \text{ Boolean entries.} \tag{25}$$

For each false-first row/column index, decode its ambient assignment, override fixed names by F, and evaluate e. This defines the corresponding entry. Fixed values affect evaluation, while their axis positions remain and repeat that value; irrelevant axes are also broadcast. A fixed variable may be supplied by F without appearing in the layout.

For example, $(X \wedge Y) \vee Z$ with $F = \{Z \mapsto 0\}$, $\mathcal{R} = [X, Z]$, and $\mathcal{C} = [Y]$ returns

$$M = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \tag{26}$$

whose rows are $(X, Z) = 00, 01, 10, 11$. The residual function is $X \wedge Y$, but the result has eight entries. A projected four-entry relation would require the caller to request a reduced layout or a separately specified projection API. The current dense wrapper neither drops fixed axes nor minimizes essential support.

The materialized dense result is an owned writable array, not a writable broadcast view. Its NumPy Boolean entries occupy one byte each; a bit-packed relation is a different output representation. Expression nodes and pair surrogates are immutable under the compiler contract, and caller-owned layouts/fixed maps are read-only.

5.4 Cost model

Let N be AST occurrences, U unique DAG nodes, H AST height, S occurrences in a subtree sent to retabulation, D its evaluation depth, k the length of a signed literal's negation chain, and m the number of retained ambient output axes. The counts $|\operatorname{supp}_F(e)|$ and the number of essential variables can both be smaller than m.

- Inspecting a signed literal costs $O(k)$ followed by an $O(1)$ four-bit transform.
- A four-bit token fusion is $O(1)$.
- One exact pair retabulation costs $O(4S)$ time plus the evaluator's stack/workspace requirements.
- A fully successful structural pair tree is ordinarily $O(N)$ time and $O(H)$ stack space, although the current support-discovery strategy can rescan rejected subtrees and therefore has an $O(N^2)$ worst case.
- A sharing-aware evaluator may scale with U rather than N, which is why DAG-preserving packed/CSE comparators are mandatory.

- The current dense wrapper budgets and materializes 2^m entries even if many are repeated. Its output allocation/copy costs $\Theta(2^m)$ time and bytes; additional compiler workspace is separate. Projected or bit-packed outputs require separate accounting.

The last point prevents a common category error: constant-time token fusion is a local kernel fact, not an end-to-end complexity theorem.

6 Closest prior art and matched comparators

The compiler should be compared with operations that possess equivalent local information, not with deliberately weaker baselines.

Boolean matrices and truth vectors. Edwards developed Boolean-matrix logic well before the present work [5]. Cheng, Zhao, and Xu use XOR as addition and AND as multiplication and state the same-variable entrywise outer-operation law in Proposition 3.3 [3]. Reshaping such a vector into 2×2 form does not make that pointwise identity new.

Matrix/vector logic. Stern and Mizraji provide established operator/matrix representations of logic [19, 14, 15]. Bricken's technical note explicitly places logical matrices between two-component states in bra–matrix–ket form [1]. The companion foundations paper contains the appropriate broader comparison; the computational manuscript needs only enough of this history to prevent novelty from being attached to notation or representation alone.

NPN, LUT, and AIG practice. Small truth functions are routinely packed into machine words, transformed under input permutation/negation and output negation, classified up to NPN equivalence, and used in cut-based rewriting. DAG-aware AIG rewriting specifically exploits graph sharing rather than treating a formula as an unshared tree [7, 23, 20]. CM transposition, row/column reversal, complement, and four-bit storage are therefore familiar small-function operations. Kitty directly provides truth-table Boolean operations, variable swaps/flips, dependence tests, cofactors, and base expansion [8]. These primitives do not distinguish this compiler.

The closest architectural comparison also carries operand identity: mockturtle cut rewriting supplies a truth table together with an ordered iterator range of leaves [9]. An explicit frame is therefore not an unprecedented addition to a truth function. DAG-aware AIG rewriting evaluates candidate replacement gain against shared graph structure before committing a selected replacement [7]. The present wrapper instead attempts a restricted variable-pair root and abandons child constructions on root failure; it neither enumerates general cuts nor selects network replacements by sharing-aware area gain.

Comparison	Shared capability	Boundary in this paper
Cheng et al.; kitty	Aligned truth-function combination and table manipulation.	No algebraic or primitive-operation novelty claim.
Cut/AIG rewriting	Truth functions with leaf identities, input transforms, local candidates.	No general cut enumeration, gain search, or persistent graph replacement here.
Direct packed/DAG evaluator	Same four values; identity sharing can be preserved.	Primary end-to-end comparator; recognition overhead must be paid.
This contract artifact	Frame, strategy, provenance, failure, ownership, and output size specified together.	Regression-tested integration; synthetic primary and bounded natural secondary comparisons are negative.

Table 2: Closest comparisons and the remaining, deliberately limited methods contribution. No kitty or AIG runtime comparison was executed for this revision.

Direct packed evaluation. A strong simple comparator evaluates the AST once over four packed masks to obtain all four assignments in parallel. With identity memoization it also preserves explicit DAG sharing. It returns exactly the same four-bit function required by a pair-token endpoint and does not need CM-specific frame metadata. This is the closest baseline for deciding whether structural recognition pays for itself.

Generic local truth-function optimization. A generic optimizer given the same local support/function information can tabulate or combine a small cut without using CM notation. The S1/S2 evidence below shows why this comparator is scientifically necessary: it reaches the same reduced symbolic outputs as CM folding on the admitted all-arm cases.

ROBDDs and broader rewrite search. ROBDDs provide canonical function representations for a fixed variable order [2]. Equality saturation and synthesis-driven superoptimization retain or search broader classes of equivalent expressions [22, 18]. These systems solve a more general problem than the fixed two-operand rule; comparison should therefore charge recognition/search/extraction and compare delivered artifacts appropriate to the task.

7 Experimental questions and evidence hierarchy

The computational claim is meaningful only if the benchmark answers the same question for every arm. Four distinctions are especially important.

1. **Kernel versus compiler.** Four-bit fusion time is not compilation time.
2. **Prepared versus one-shot.** A resident token/program may amortize setup; one-shot use may not.
3. **Compact versus materialized output.** A token endpoint is not a complete 2^m-entry ambient truth relation.
4. **Task contract.** ANF production, complete truth-vector evaluation, repeated restriction, counting, and multi-root output are different tasks and cannot share one headline ratio.

Figure 2 shows the evidence levels used in this draft.

1. Formal/finite correctness: frame transforms, fusion invariant, retabulation, provenance, implementation tests.

2. Exploratory mechanism: S1/S2 pre-expansion work reductions and exact tie with matched generic local optimization.

3. Completed different endpoint: P14-PY0 ANF dispatcher, negative overall under its frozen primary rule.

4. Separate project context: later current-source complete-relation/restriction/multi-root campaigns; useful for methodology, not pair-compiler attribution.

5. Pair-token evidence: frozen synthetic v4 primary comparison is negative; bounded natural v5 secondary timing is negative; broader v3 scope remains open.

Figure 2: Evidence hierarchy. A favorable result on a different endpoint is not promoted into pair-compiler evidence.

7.1 Confirmatory workload design

The preserved, pre-freeze protocol-v3 design [11] separates exact-frame formulas, signed/swapped frames, local retabulation cases, hybrid mixed-child cases, multiple operand pairs, compound-operand cases outside the structural claim, and natural formulas/circuits. Balanced generated trees use AST occurrence counts $15, 63, 255, 1023, 4095$; skewed trees stop at 255 occurrences and measured height at most 128. Cases cross the five supported outer connectives, reuse counts $1, 10, 100, 1000$, and sharing conditions that distinguish identity-shared nodes, structurally equal but separately allocated subtrees, and no sharing. Every case records both N and U.

Requested outputs include compact tokens, scalar evaluation, packed batches, and dense outputs where the output budget permits them. Preparation and query costs are reported separately. Once two arms have produced the same four-bit token and call the same lookup function, the query cost is common; any advantage must arise earlier or through a measured reuse policy.

The sole primary cell is intentionally narrow: signed/permuted pure-structural balanced formulas with $N = 1023$, $U = N$, no fixed variables, token output, reuse one, compared end to end with direct packed AST evaluation. The proposed primary success rule requires zero correctness disagreements, a median competitor-to-CM ratio of at least 1.20 whose 95% interval lies entirely above one, and no more than a 10% median peak-memory increase. The interval requirement establishes superiority, not a lower confidence guarantee of 1.20×. All other cells are secondary. This prevents a favorable post hoc workload from replacing a failed primary comparison.

A dated protocol-v4 successor fixed the synthetic primary case count and seeds, resident cache-reset timing, formula-level aggregation, and resampling before timing held-out formulas; Section 8.4 reports its negative result. A separately frozen EPFL primary-output admission and correctness screen reports root outcomes by strategy (Section 8.6). A later frozen descriptive v5 run measures matched Python full-bitset production on 70 of the 71 selected cones, retaining one calibration failure; ABC process wall remains an unmatched diagnostic. Admission addresses applicability rather than timing superiority.

The 1.20× ratio and 10% memory allowance are engineering acceptance targets, not consequences of soundness or a power calculation. Their purpose is to demand a material gain while limiting memory growth; a 1.20× ratio corresponds to 16.7% less elapsed time. The preserved plan did not establish an empirical power calculation. The v4 pilot used disjoint seeds and met the 5% repeatability criterion on its eight cases; the 64 formula/case seeds, rather than timing repetitions, supply the independent units for the reported synthetic interval. Its negative result is retained under the originally declared threshold.

Initialization and cache regimes. The source builds composition lookup tables when `cm_token` is imported, memoizes `cm_compose`, and caches permutation/lift metadata in `cm_normalize`. A fresh-process one-shot measurement must charge startup/import and table initialization. V4 instead uses a fresh process per formula but times repeated calls after common imports, clearing the CM memoization before each CM call and the bitset-environment cache before each direct call. The direct timed call builds its environment; imported lookup tables remain resident and their setup is recorded separately. This is a post-import cache-reset comparison, not a full process-start comparison. Warm regimes require a declared priming workload and cache lifetime. Charge support discovery, recognition, unsuccessful pair work, retabulation, fallback, and required materialization to their respective production arms; separate common token queries from preparation.

8 Current evidence

8.1 Correctness and implementation assurance

The historical record reports 116 focused tests on 19 September 2026 and a separate 1,048,576-case signed-fusion check. Those reports do not by themselves bind that historical execution to the source revision inspected here. This revision supplies a fresh, separately identified run against the unmodified pinned source: 90 upstream tests from `test_cm_pair_alignment.py` and `test_output_budget.py`, plus 33 contract regression cases, all passed. The latter include an exhaustive comparison over all 16 input tokens in each of eight signed frames, both children, all five supported outer connectives, and all four assignments: 327,680 scalar comparisons. This is implementation correctness evidence, not a timing campaign or a rerun of the historical 116-test selection.

Expression / boundary	**Pure**	**Hybrid**	**Retabulate**
$X \wedge Y$	S, 1000	S, 1000	T, 1000
$X \vee (Y \wedge Y)$	NoPair	T, 1110	T, 1110
$\neg(X \vee (Y \wedge Y))$	NoPair	H, 0001	T, 0001
$X \vee (Y \Updownarrow Y)$	NoPair	T, 1100	T, 1100
$(X \wedge Y) \vee (Z \Updownarrow Z)$	NoPair	NoPair	NoPair
$(X \wedge Y) \vee Z$, $F(Z) = 0$	Builder	T, 1000	T, 1000

Table 3: Contract examples. Tokens use true-first order. Layouts are $[X], [Y]$ for the first four rows and $[X, Z], [Y]$ for the last two. "Builder" means the dense wrapper executes ordinary fallback; its token API returns NoPair. All dense strategies in the last row return the eight-entry matrix in Equation (26).

The suite also checks constant-valued redundant formulas, identity-shared versus separately allocated equal trees, invalid layouts, retained irrelevant axes, owned output, input nonmutation, budget refusal before pair

compilation, token failure without builder execution, and identity of the original expression passed to fallback after successful child construction. These finite checks make the operational boundary reusable and falsifiable. They neither prove unrestricted program correctness nor demonstrate an advantage over existing synthesis libraries.

The subsequently supplied full audit includes a separate standard-library mathematical/reference-model checker. Rerunning its unmodified script reproduces the supplied results exactly: 1,048,576 fusion assignment checks over all 16 abstract outer functions, 8,080 bounded structural expressions, and 15,000 seeded strategy runs with 6,615 successful tokens. These are reference-model checks, not additional production-source tests. The audit archive and its 21 verified member hashes are preserved separately from the implementation evidence.

8.2 S1/S2: mechanism evidence and the generic tie

A fresh recount of the recovered canonical exploratory S1/S2 CSV confirms 10,500 generated cases. Of these, 10,389 completed in all four arms: direct ANF, sharing-aware common-subexpression evaluation, a generic local truth-function optimizer, and CM folding. The all-arm population contains 3,750 S1 and 6,639 S2 cases. The remaining 111 are retained rather than silently filtered: 108 fail in all arms, two complete only in the generic and CM arms, and one only in the sharing arm. No completed correctness disagreement was recorded.

On every admitted all-arm case, CM and generic local optimization agree on seven principal symbolic-output metrics: final degree, final monomial count, multiplication count, output unique nodes, pair terms, peak monomials, and serialized bytes. The generic arm records more folds on 6,010 cases and CM more on none; fold count is not runtime. For corrected S2, median direct/CM multiplication counts at requested sharing fractions 0, 0.25, and 0.5 are respectively 35/35, 17/12, and 9/0.

The correct interpretation is therefore two-sided. Local small-function folding can remove work before polynomial expansion in some structures. The observed reduction is not uniquely CM-specific: an equally capable generic local optimizer reaches the same reduced outputs. This is mechanism evidence, not an end-to-end speed result.

8.3 P14-PY0: a completed negative dispatcher boundary

P14-PY0 measures a different endpoint: recipe-to-owned-ANF compilation. Direct lowering (D) lowers the ordinary source DAG; a minimal observer (F) records dispatch information; exact dispatch (E) decides whether to lower directly or construct folding metadata; always-fold (A) and a generic signature optimizer (G) are diagnostics. Preparation required by an arm is included in its production timer. This is not the direct packed token task of protocol v3.

The recovered historical campaign report used 384 synthetic cases from 12 families and 107 natural cones from 14 previously uninspected EPFL design families, for 491 frozen cases. Three fresh sequential workers produced 51,102 measured rows including diagnostics/canonicalization; equality checks passed the 491 frozen cases and 512 targeted correctness cases. Exact dispatch selected folding on five synthetic cases and no natural cone.

Corpus	Ratio	Geometric mean	98.75% marginal interval
Synthetic	E/F	0.9606	[0.9455, 0.9811]
Synthetic	E/D	1.0508	[0.9683, 1.1075]
Natural	E/F	0.9610	[0.9411, 1.0001]
Natural	E/D	1.0607	[1.0176, 1.0927]

Table 4: Historical P14-PY0 ANF recipe endpoint, not the pair-token protocol. Ratios below one favor the numerator. Bonferroni-adjusted 98.75% marginal bootstrap intervals target 95% familywise coverage over the four primary comparisons, subject to component coverage.

The frozen primary rule required E/F geometric mean at most 0.99, E/D at most 0.95, and the relevant multiplicity-adjusted upper bound below one on both corpora. It did not pass. At this endpoint exact dispatch was about 5.1% slower than direct lowering on the synthetic corpus and 6.1% slower on the natural corpus. The five selected synthetic cases themselves were highly profitable, with E/D ratios between 0.075 and 0.389, but those local wins did not offset aggregate policy overhead. The preregistered case-balanced geometric mean is retained even though an alternative ratio of summed synthetic medians was favorable. This revision reconstructs the four point estimates and all 17 reported interval sets from the 51,102 raw timing rows using the archived 10,000-draw bootstrap seed, resampling order, and quantile rule. The frozen analysis uses the 0.00625 and 0.99375 quantiles for each primary comparison, clarifying the coverage label in Table 4.

The recovered freeze records Windows 10, Python 3.10.11, 12 logical CPUs, Balanced power mode, and uncontrolled affinity. The local freeze reduced the draft's 30 workers/100 ms batches to three workers/10 ms target batches with a 40 ms slow-arm cap. Preserved amendments cover a pre-timing per-arm correctness timeout, an extended wall-time budget after the first worker, and an analyzer indexing optimization. The original analyzer and holdout hashes match the recovered freeze. These records improve traceability but do not establish the complete historical runtime dependency closure or reproduce its timing on a new host.

8.4 Frozen synthetic pair-token confirmation: negative utility boundary

The disclosed protocol-v4 successor [12] preserved v3's sole synthetic primary cell and four-part decision rule while limiting the executed claim to signed/permuted, unshared two-variable formulas. An eight-formula pilot with disjoint seeds met the 5% within-case repeatability target before the 64 held-out formula seeds and analysis were frozen. Every held-out formula has $N = U = 1023$ AST nodes, 205 primitive row/column operators, 204 signed literal occurrences, balanced combination, and token output at reuse one. Independent scalar evaluation of all four assignments and both production arms agreed; every CM root was pure structural with zero local retabulations.

One fresh process per formula supplied 30 randomized paired repetitions, yielding 3,840 arm/repetition rows, plus 128 separate process-memory observations and no recorded failures. The compared timer begins after common imports and expression construction. Before each CM call, composition and normalization memoization are cleared; before each direct call, the bitset-environment cache is cleared and environment construction is timed. Import-time CM lookup-table setup is recorded but excluded, so this is a resident post-import comparison and not a process-start one-shot claim. Both producers return the same four-bit token and use the same query function; query cost is excluded from both production timers.

The formula-median direct-packed/CM time ratio is 0.2479, with a 95% hierarchical bootstrap interval $[0.2462, 0.2495]$ from 10,000 prespecified draws. The median absolute formula times are 0.588 ms for direct packed evaluation and 2.377 ms for CM compilation; these separate medians are descriptive and their quotient is not the primary statistic. The median CM peak-working-set increase is -1.56% under the declared fresh-process metric. Common interpreter/import memory dominates these small allocations, so that metric does not prove a material memory advantage. The premeasurement environment record omitted the power scheme; a post-run read showed Windows Balanced but does not establish that setting throughout the run. Correctness and the memory tolerance passed, but both the 1.20 timing threshold and the interval-above-one criterion failed. Under this exact implementation, grammar, output, and cache regime, direct packed evaluation is about four times faster; the measurements do not isolate which compiler phase causes the gap.

This is a negative result on the actual pair-token endpoint, distinct from P14's ANF endpoint. It does not measure natural circuits, ABC/AIG competition, identity-shared graphs, dense output, full process startup, or prepared reuse. The separate natural correctness and descriptive timing work below do not retroactively widen this frozen primary comparison.

8.5 Later project-wide results: relevant context, different contracts

Later project reports distinguish favorable small kernels from costly wrappers and show that evaluator rankings change with output and reuse contracts [16, 17]. Their methodological relevance is to require direct packed/CSE controls, explicit preparation costs, and matched outputs. Their numerical ratios are not reproduced here because they measure other tasks and do not fill the protocol-v3 pair-compiler result table.

8.6 Natural-circuit admission, correctness, and secondary timing

The narrow synthetic primary cell has a negative answer under v4's post-import cache-reset policy. A separate, prospective natural admission screen fixed its rule before any CM compilation [13]. It screened all 2,861 original primary outputs of 20 arithmetic and random/control EPFL BLIFs at a pinned Git revision. Cones with two to twelve syntactic support variables, at most 128 source nodes and depth at most 128 were eligible; up to eight were hash-ranked per design, with no internal cone substitution or post-selection backfill. This admitted 71 original primary-output cones from ten designs. Ten designs had no eligible primary output. Because the source families overlap prior P1–P14 work, this is an applicability cohort, not an independent uninspected-family holdout.

The copied BLIF bytes match the pinned upstream Git blobs. Translation to the immutable AST and one-output cone export were checked exhaustively against each source BLIF. An ABC 1.01 executable from the pinned 29 July 2026 OSS CAD Suite Windows release imported, structurally hashed, optimized with `dc2`, and exported an

AIG for all 71 cones. Exhaustive AIG truth tables matched the original BLIFs. Before the first CM compilation, a separate correctness run froze the source closure, scripts, executable, input corpus, AIGs, case timeout, and initially empty result file. In fresh case processes, direct packed AST evaluation, ordinary CM IR packed evaluation, and every returned pair token agreed with the BLIF truth table on all 71 cases; there were no failures or timeouts.

Under the requested `hybrid` strategy, four of 71 selected roots were pure structural, one was a whole-root retabulation, and 66 returned ordinary fallback. All five non-fallback roots had two syntactic support variables; the other 66 had three to twelve. Under `pure_structural`, the outcome counts were four pure and 67 fallback; under explicit `retabulate`, they were five whole-root retabulations and 66 fallback. These fractions describe the prespecified selected cohort, not all EPFL outputs or circuits generally.

A separate v5 protocol was frozen before natural-case timing. It compares resident Python production of the *full packed truth bitset* from the unchanged AST: direct packed evaluation against hybrid pair compilation and, where the root has no pair, ordinary CM IR construction and evaluation. Both arms include their own production and clear documented caches outside the timer; every timed invocation checks the complete bitset against the source BLIF oracle. Each case was assigned 15 randomized paired repetitions after warm-up and calibration. One adder output, `f[1]`, failed the frozen 100 ms calibration floor after its single allowed retarget and has no timed repetitions. The failure remains in the denominator. The other 70 cases have 2,100 complete arm/repetition rows, and no truth disagreement or timed-process timeout. This is a descriptive secondary result: cases cluster within ten designs, overlap earlier P1–P14 families, and do not supply independent population draws or a new primary utility test.

Across the 70 complete cases, the median of case-level direct/CM producer-time ratios is 0.0376; direct packed evaluation is faster on all 70. Within the frozen hybrid-root strata, the median ratios are 0.240 for four pure structural cases, 0.112 for the one whole-root retabulation, and 0.0360 for 65 ordinary fallbacks. These are ratios of each case's median per-call times, not a ratio of pooled time totals. Fresh-process peak working set was recorded for both Python arms on all 71 cases; the median relative CM increase is -0.15%, too small against shared interpreter/import memory to establish an allocation advantage. Power scheme and affinity matched the before/after records, without proving constancy throughout measurement.

The pinned ABC executable also ran fresh-process `read_blif; strash; dc2; write_aiger` for 15 repetitions on each of the 71 cones. All 1,065 exported AIGs passed exhaustive truth checking. The median case-level ABC subprocess wall time is 304 ms, including startup and export; the coarse per-phase diagnostics often round to zero. Python AIGER parsing and exhaustive simulation were measured separately and are not native ABC query time. These ABC measurements cannot form a speed ratio with the resident Python producer times. ABC process memory, an applicable ROBDD/generic comparator, sharing, prepared reuse, alternative output contracts, independent natural families, and a complete matched protocol-v3 systems campaign remain open. No secondary result replaces the failed synthetic primary rule.

S1/S2, P14-PY0, and later project-wide architecture runs answer other questions; none is used to overwrite the failed synthetic primary rule.

9 Discussion

9.1 What the CM representation specifically contributes

The raw four truth values are not new. The computationally useful structure is the combination of a compact token with an explicit frame. That representation provides four practical properties.

First, *Boolean legality is local*: for valid child tokens, equality of canonical row/column metadata certifies same-frame Boolean fusion. The numerical contracts in Section 4.4 require additional checks. Second, signed normalization is fixed and cheap after recognition: transpose and row/column swaps act directly on four bits. Third, the fused result remains another four-bit token in the same frame, so the rewrite composes without expanding the operands. Fourth, provenance can say whether the token was derived structurally, retabulated, or assembled from both mechanisms.

A generic optimizer can implement the same semantic reduction with a different representation. The question is therefore not whether CM has exclusive access to the reduction, but whether the CM/frame organization lowers the engineering cost of recognizing and composing it in a larger compiler. That engineering benefit is a research question, not an established result. The demonstrated contribution here is the explicit contract plus regression cases that expose otherwise easy-to-miss integration errors; the elementary identities and fragment induction do not establish substantial algorithmic novelty.

9.2 When folding can help

Folding is most plausible when the same two live operands recur through multiple signed/order variants, when expansion would otherwise repeat symbolic work, when the result will be reused, or when an admitted root can be collapsed before an expensive downstream representation is built. Exploiting pair children below a rejected root would require a persistent rewrite pass beyond this wrapper. The S1/S2 sharing strata provide a direct illustration: the largest observed reductions occur where repeated/local structure gives the small-function rewrite something to eliminate.

The same mechanism becomes less attractive when compatible pairs are rare, support discovery rescans rejected subtrees, a sharing-aware baseline evaluates unique DAG nodes once, a one-shot call cannot amortize preparation, or the requested output forces a much larger materialization. The later project-wide wrapper and BitSet results show that these overheads are not theoretical nuisances; they can dominate otherwise favorable kernel operations.

9.3 Retabulation is useful but conceptually different

Evaluating four assignments is exact for a two-variable function, but it should not be allowed to make a structural recognition rule look more powerful than it is. Retabulation consumes the subtree as a black box and reconstructs its local truth function. Structural folding instead proves that a token can be derived compositionally from recognized children and frame transforms. Both are valid compiler techniques; their costs and explanatory value differ.

This is why the hybrid path has its own provenance. It allows the practical compiler to continue after a local retabulation while preserving an auditable distinction between recognition and re-evaluation.

9.4 Sharing and output type change the break-even point

An AST occurrence count can dramatically overstate work when a comparator preserves identity sharing. Reporting both N and U is therefore mandatory. Equal-but-separately-allocated subtrees must also be distinguished from identity-shared nodes because hash-consing or structural CSE may discover the former while simple identity memoization does not.

Output type matters just as much. A four-bit token is a legitimate endpoint for a small-function compiler decision; it is not equivalent to returning a dense relation over many variables. The dense wrapper retains fixed and irrelevant ambient axes, so its allocation can exceed the table over essential variables. Conversely, if a caller asks for repeated scalar or residual queries, preserving a prepared compact representation may be more important than materializing the full vector. Break-even reuse should therefore be derived from measured setup and per-query costs, not from an isolated token-fusion benchmark.

In particular, if both arms return the same token and use the same lookup cost L, then

$$T_{\mathrm{CM}}(q) = C_{\mathrm{CM}} + qL, \qquad T_{\mathrm{base}}(q) = C_{\mathrm{base}} + qL.$$

Their difference is $C_{\mathrm{CM}} - C_{\mathrm{base}}$ for every q: more lookups alone cannot reverse the preparation-time ordering. A genuine reuse crossover requires different retained work or per-query costs, not simply amortizing two equal lookup routines.

9.5 What the negative synthetic result means

The negative v4 result does not invalidate the soundness theorem or the observation that local folding can reduce symbolic work. It shows that, for this synthetic signed grammar and token-output task, the current frame-aware implementation does not repay its production cost against direct packed evaluation. The separate natural v5 result shows an even wider descriptive gap for full-bitset production in the selected cohort, dominated by ordinary fallback. Neither comparison isolates which compiler phase causes the gap. Together they narrow the useful engineering claim and make the paper a compiler-mechanism and evaluation-boundary study.

Conversely, a positive result should remain scoped to the exact output, reuse, sharing, and workload region that passes the frozen rule. The later architecture campaigns demonstrate why a task map is more informative than a universal ratio.

10 Limitations

The current compiler intentionally recognizes a narrow class of two-operand local structure. It does not solve general Boolean equivalence, NPN canonicalization, LUT mapping, AIG rewriting, or broad superoptimization. Its support-discovery implementation can have a quadratic worst case on rejected structures. The implementation uses a host-language recursive traversal and therefore imposes an explicit height bound on skewed benchmark cells rather than relying on an increased recursion limit.

The S1/S2 corpus is synthetic/exploratory and contains only sixteen possible semantic functions on a fixed two-variable support; large two-variable trees measure syntactic redundancy and compiler behavior rather than growing semantic complexity. The v4 primary cases are likewise synthetic, deliberately unshared and restricted to two live variables; their narrow negative comparison does not establish the effect on natural circuits or prepared workloads. V4 excludes import/startup time from both production timers and uses process peak working set, which is insensitive to small allocations beneath a shared interpreter baseline. P14-PY0 measures ANF lowering rather than the packed-token endpoint. The later August/September project records use still other task contracts. The selected EPFL cohort supplies one bounded natural admission and secondary producer-timing result, but not a general prevalence or performance estimate. One of 71 cones failed timing calibration, and ABC startup-inclusive process wall is not task-matched to Python production time.

Finally, a full relation over n unfixed variables has 2^n entries, but this API retains m ambient axes and materializes 2^m entries, even when fixed or irrelevant coordinates repeat. This is an output-contract cost, not a claim that the repeated data are information-theoretically necessary. CM layout does not remove the cost of the requested explicit representation. Compression, symbolic sharing, and partial-query benefits must be evaluated as separate representation or task properties.

11 Conclusion

The computational role of correspondence matrices is most defensible when stated narrowly. A four-bit binary operator token becomes useful to a compiler when its operand frame is typed. Signed and swapped occurrences can then be transported to a canonical frame, compatible tokens can be fused before operand evaluation or expansion, and unsupported structures can be retabulated or returned to an ordinary IR without compromising correctness.

The evidence draws an equally important boundary. Local small-function folding can reduce pre-expansion work, but a matched generic optimizer can reproduce the same symbolic reductions. A frozen dispatcher campaign shows that even very profitable local choices can lose overall once decision overhead is charged. The frozen synthetic pair-token comparison now adds a direct negative result: on its $N = U = 1023$ signed primary cell, the direct packed evaluator produces the same token at roughly one quarter of CM's time. Later project-wide benchmarks reinforce the need for task-matched endpoints, but none changes this primary outcome.

The next empirical question is therefore precise: whether a different natural corpus admits this structural path often enough to matter, and whether any output, sharing, or reuse regime repays its recognition and preparation costs. The selected EPFL cohort shows limited root admission and slower CM full-bitset production on every timed cone under the present wrapper. A completed broader, task-matched circuit study can test other contracts without requiring any claim that the underlying truth-table operations are uniquely CM-specific.

A Compact proof details

A.1 Signed transformations

Left multiplication by P exchanges the two rows of $[\Theta]$, so the row-x, column-y entry becomes the original row-$\neg x$, column-y entry, which is $(\neg x)\Theta y$. Right multiplication exchanges columns. Transposition exchanges the two assignments. Because P is a permutation matrix under XOR–AND multiplication, no multi-term cancellation occurs in these reindexings.

A.2 Hybrid induction

For pure structural derivations, primitive normalization is the base case and Equations (18)–(19) preserve Equation (17). For a hybrid derivation, exact local tabulation supplies an additional sound base case. Every

parent above that base is again token complement or same-frame fusion. Thus hybrid provenance records how the token was obtained but does not alter its denotation.

A.3 Uniqueness of tabulated token

Fix R, C. Any binary Boolean function on those two variables is determined by its four values on $(1,1),(1,0),(0,1),(0,0)$. Storing those values in the declared true-first token order therefore produces the unique token that agrees with the source expression on every assignment.

B Artifact availability, evidence status, and remaining gate

The dated revision package contains this source/PDF, the received summary and full audit bundle, a finding-ID resolution matrix, and a `reproducibility` directory. The latter includes a source/import closure at the pinned commit (Git blob and SHA-256 checks), the immutable protocol-v3 plan, 33 contract cases, two upstream test files, byte-preserved historical evidence, and fresh test/recount logs with environment versions. A separate `03_STUDY_DESIGN_20260926` directory contains the frozen v4 protocol, 64-case manifest, source/environment hashes, append-only timing and memory rows, analyzer, independent recount, and historical P14 bootstrap replay. The `04_NATURAL_CIRCUIT_EXTENSION_20260926` directory contains the original EPFL source files, admission screen, pinned ABC tool, 71 exported AIGs, and frozen raw correctness outcomes. The `05_NATURAL_TIMING_20260926` directory contains the frozen v5 secondary protocol, raw Python and ABC timing rows, retained failure, descriptive analysis, and an independent row recount. From the revision's reproducibility directory, the portable correctness/recount command is:

```
python run_checks.py
```

The runner verifies snapshot hashes, runs the 123 selected tests, recounts the recovered records, and reruns the supplied reference checker without executing historical timing workers. The complete runner passed in a fresh Python 3.10.11 virtual environment without system site packages or optional backends; `requirements-checks-lock.txt` records its transitive dependency versions. Separate analysis and reference-check commands are `python check_historical.py` and `python check_audit.py`. The P14 interval replay uses the archived seed and raw rows without importing its historical compiler; the v4 study has its own frozen run and analysis files. V5's `verify_run001.py` independently checks raw-row counts, pair coverage, truth digests, hashes, and the reported median. `verify_manifest.py`, run from the revision directory, checks the delivered file manifest; `build.py` builds the manuscript with pdfLaTeX. The obsolete checker under the preserved historical manuscript is not the entry point for this revision.

A separate supplementary diagnostic under `reproducibility/semantic_boundary_20261013` contains `semantic_contract_checker.py` and its exact-rational result record. It checks 12,288 bounded compositions and examples supporting Section 4.4; its 42,918 assertions concern these elementary semantic bridges, not the pinned compiler implementation, training, novelty, or a numerical performance advantage. Run `python semantic_contract_checker.py` from that supplementary directory. The original 123-test compiler suite and historical freezes retain their distinct scope. A subsequent adversarial audit is recorded under `reproducibility/semantic_boundary_audit_20261013`, including exact checks of the domain, event-identity, gradient and crispification distinctions; it preserves the preceding supplement as historical evidence.

Evidence	Established in this revision	Still not established
Compiler contract	Pinned source, 123 passing selected tests, explicit failures and output shapes.	Historical 116-test execution identity; unrestricted runtime assurance.
S1/S2	Corrected CSV, generators/scripts recovered; counts and seven-metric tie recounted.	New timing evidence or CM-specific superiority.
P14-PY0	Raw rows, freezes and analyzers recovered; all 17 bootstrap interval sets reproduced; eight freeze hashes match.	Historical imported-runtime execution identity remains uncertified.
Protocol v4	64 held-out synthetic cases, 3,840 timing rows, 128 memory rows; failed primary utility rule.	Natural findings cannot replace this primary result; sharing, reuse and full process-start costs.
Natural gate and v5	2,861 original outputs screened; 71 admitted cones and ABC timing/correctness complete; 70 paired Python timing cases, one calibration failure, all 70 favor direct packed.	Matched ABC/ROBDD producer costs and memory, broader prevalence, alternative contracts, independent families.
Protocol v3	Original broad plan preserved.	Natural/circuit campaign and broader workload map.

Table 5: Evidence recovery is distinguished from new experimental execution.

The current package closes the local manuscript–contract-test–source chain, the historical raw-record–bootstrap analysis, the synthetic v4 primary study, and a bounded natural admission, correctness, and secondary producer-timing gate. It is not a complete natural-circuit performance artifact release. Before serious submission, measure task-matched synthesis comparators and other secondary contracts on separately frozen runs, identify the historical imported runtime used for P14 timing, and provide a stable public archive identifier with dependency and corpus licensing/provenance. No public release was performed for this revision.

References

[1] William Bricken. *Notes on Matrix Techniques for Logic.* Technical note internally dated March 1997; original public-release date and peer-reviewed status not established. https://wbricken.com/pdfs/01bm/01math/03math-supporting/math-tangential/04matrix-tech.pdf.

[2] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

[3] Daizhan Cheng, Yin Zhao, and Xiangru Xu. Matrix approach to Boolean calculus. In *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference*, pages 6950–6955, 2011. doi:10.1109/CDC.2011.6160289.

[4] Brian Droncheff. *Correspondence Matrices; Algorithms for Propositional Logic.* ResearchGate preprint, 2018. doi:10.13140/RG.2.2.28036.37764.

[5] C. R. Edwards. The logic of Boolean matrices. *The Computer Journal*, 15(3):247–253, 1972. doi:10.1093/comjnl/15.3.247.

[6] Siang-Yun Lee, Heinz Riener, Alan Mishchenko, Robert K. Brayton, and Giovanni De Micheli. A simulation-guided paradigm for logic synthesis and verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2573–2586, 2022. doi:10.1109/TCAD.2021.3108704.

[7] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *Proceedings of the 43rd Design Automation Conference*, pages 532–535, 2006. doi:10.1145/1146909.1147048.

[8] Kitty project. *Operations: truth-table predicates and manipulation.* Official documentation, accessed 26 September 2026; documentation version is not a benchmarked binary version. https://libkitty.readthedocs.io/en/latest/operations.html.

[9] Mockturtle project. *Cut rewriting*. Official documentation, accessed 26 September 2026. https://mockturtle.readthedocs.io/en/latest/algorithms/cut_rewriting.html.

[10] Correspondence Matrices Project. *Pair compiler implementation*, commit 0ab8ffd0c23ffa71ee951d375b1c170ffdcc084b. https://github.com/Relative0/Correspondence_Matrices/tree/0ab8ffd0c23ffa71ee951d375b1c170ffdcc084b. Source closure and hash manifest included in this revision package.

[11] Correspondence Matrices Project. *Evaluation protocol v3*, 15 September 2026, pre-freeze/unexecuted. paper_program/01_audit/EVALUATION_PROTOCOL_V3.md, at the commit in [10]; byte-identical copy included in the source snapshot.

[12] Brian Theory. *Protocol v4: signed pair compiler, synthetic primary cell*, 26 September 2026. Dated local study package 03_STUDY_DESIGN_20260926; frozen plan, manifest, raw rows and analysis included.

[13] Brian Theory. *Natural-circuit admission and correctness gate*, 26 September 2026. Dated local study package 04_NATURAL_CIRCUIT_EXTENSION_20260926; source, admission screen, ABC/AIG exports, frozen correctness run and raw outcomes included.

[14] Eduardo Mizraji. Vector logics: The matrix–vector representation of logical calculus. *Fuzzy Sets and Systems*, 50(2):179–185, 1992. doi:10.1016/0165-0114(92)90216-Q.

[15] Eduardo Mizraji. Vector logic: A natural algebraic representation of the fundamental logical gates. *Journal of Logic and Computation*, 18(1):97–121, 2008. doi:10.1093/logcom/exm057.

[16] Correspondence Matrices Project. *CM benchmark audit correction — 2026-08-25*. Public technical record, 25 August 2026. https://github.com/Relative0/Correspondence_Matrices/blob/main/deliverables_n22_24/corrections_2026_08_25/CM_BENCHMARK_AUDIT_CORRECTION_REPORT_2026-08-25.md.

[17] Correspondence Matrices Project. *Architecture-aware CM comparison refresh after C37*. Public technical record, 3 September 2026; updated 4 September 2026. https://github.com/Relative0/Correspondence_Matrices/blob/main/docs/research/CM_ARCHITECTURE_AWARE_COMPARISON_REFRESH_AFTER_C37_2026_09_03.md.

[18] Raimondas Sasnauskas et al. Souper: A synthesizing superoptimizer. arXiv:1711.04422, 2017.

[19] August Stern. *Matrix Logic: Theory and Applications*. North-Holland, 1988.

[20] Eleonora Testa, Mathias Soeken, Luca Gaetano Amarù, and Giovanni De Micheli. Logic synthesis for established and emerging computing. *Proceedings of the IEEE*, 107(1):165–184, 2019. doi:10.1109/JPROC.2018.2869760.

[21] Brian Theory. *Correspondence and Logical Matrices: A Boolean Operator Calculus: Formula-valued logical matrices, valuation, pairing, and coherent operator superposition*. Companion research draft, 22 September 2026.

[22] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages* (POPL), 2021. arXiv:2004.03082.

[23] Xuegong Zhou, Lingli Wang, and Alan Mishchenko. Fast adjustable NPN classification using generalized symmetries. *ACM Transactions on Reconfigurable Technology and Systems*, 12(2):7, 2019. doi:10.1145/3313917.

[24] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *Journal of Applied Logic*, 22:46–62, 2017. doi:10.1016/j.jal.2016.11.031.

[25] Mark-Oliver Stehr, Minyoung Kim, and Carolyn L. Talcott. A probabilistic approximate logic for neuro-symbolic learning and reasoning. *Journal of Logical and Algebraic Methods in Programming*, 124:100719, 2022. doi:10.1016/j.jlamp.2021.100719.

[26] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. *Proceedings of Machine Learning Research*, 80:5502–5511, 2018. https://proceedings.mlr.press/v80/xu18h.html.