Coherent Specialization on GraalVM

From categorical algorithm specifications to polyglot kernels

Anonymous author

November 2026

Abstract

We formulate Truffle-style partial evaluation as restriction along static contexts and give a correctness proof for an executable residualizer. We construct a space of certified implementations over each context, explain how restriction transports these spaces coherently, and prove effective descent of certification for families in an ∞-topos. The executable artifact compiles typed specifications to Truffle call targets callable from JavaScript and Python. Domain front ends lower finite impulse-response filter banks, decision-tree ensembles, periodic free-electron evolution, and an interacting finite fermion field. Word semantics are exact modulo 2^{64}. Quantum dynamics are unitary over the mathematical model and executed in binary64 with independently checked numerical error. Time sweeps expose the spread of a free-electron packet and isolate the effect of the interaction on neighboring-particle probabilities.

1 Introduction

Graal's Truffle framework compiles language interpreters by specializing them with respect to the guest AST and stable interpreter state. It combines partial evaluation with profile-guided optimization and may fall back to interpretation when an assumption fails [1, 5]. The architecture realizes the first Futamura projection in a practical runtime. In the usual equation, specializing an interpreter I at a program p produces an executable I_p with $I_p(d) = I(p, d)$ on its observation domain [6, 7]. Binding a source program's static data before it reaches Truffle adds a separate specialization stage. We formalize how these stages fit together and give a library that exercises them.

Our main mathematical object is an implementation fiber over an observation. A residual program together with a proof of its observation defines a point of this fiber. Restricting the static context carries both the program and the proof. In the higher formulation, paths between proofs are retained and the comparison of restrictions is coherent. A descent theorem says when measurements of equivalence over a cover suffice to construct a global certified implementation. When all observations are set-valued, this framework recovers an ordinary correctness statement.

There are four domain compilers over a shared typed algebra. The FIR compiler exploits symmetry in a ring and specializes a filter bank at a fixed index. The forest compiler shares affine feature calculations and specializes static model parameters. The two quantum front ends generate finite-dimensional evolutions and lower their real and imaginary rows to separately callable scalar kernels. On the periodic lattice the free Hamiltonian is diagonal in momentum. In the field example, an occupation basis supports a hopping Hamiltonian with an interaction term. Both use the same runtime and guest-language call protocol.

The journal's prior work on Futamura coherence and higher e-graphs develops guarded local specialization, descent, and deployment [12, 13]. Its Caprice manuscript studies partial evaluation for a capture-aware regex language and a Cranelift back end [14]. We give a typed restriction model centered on the Truffle interpreter boundary and test reuse of the same generated executables across guest languages. Earlier modal accounts explain binding time by a type discipline [8]; here the index maps organize families of specializations and observations.

2 Typed specifications and static contexts

Let $W = \mathbb{Z}/2^{64}\mathbb{Z}$ and $B = \{\mathsf{false}, \mathsf{true}\}$. The signed interpretation $s : W \cong [-2^{63}, 2^{63} - 1] \cap \mathbb{Z}$ chooses the unique representative in that interval. Addition, subtraction, and multiplication take place in W. Equality compares residues, and less-than compares their images under s. Each primitive is total. A public call consumes and returns words. Boolean terms act as internal conditions. Java's signed `long` implements the word operations with 64-bit precision and wrapping [11, Section 4.2.2].

A context Γ is a finite typed list of variables with types W or B. Its carrier X_Γ is the product of its variable carriers. A single-result term is built from variables, literals, the arithmetic and comparison primitives, typed conditional expressions, and let bindings. A tuple of terms with common context Γ is a morphism $\Gamma \to \Delta$ when the term types match the entries of Δ. Composition is si-

multaneous capture-avoiding substitution, and identities are tuples of variables. We use the equational quotient with alpha-conversion, let-substitution, branch selection for literal conditions, and valid primitive computations on literals. Products are concatenation of contexts and the empty context is terminal. This presents a finite-product category of specifications. At the level of syntax, let bindings remain available to express sharing without duplicating a subterm.

The interpretation den sends contexts to carriers and terms to functions. Conditional interpretation evaluates the condition and selects the corresponding branch. The interpretation of $\mathsf{let}\ x = u\ \mathsf{in}\ v$ at γ evaluates v at $(\gamma, \mathrm{den}(u)(\gamma))$. By induction on the term, evaluation after simultaneous substitution equals function composition. For the let case, lift the substitution through the binder by extending it with the fresh variable and then apply the inductive hypothesis to the two subterms. Every defining equation is preserved. Thus den is a finite-product functor to **Set**. In particular, a compositional interpretation of a categorical specification is determined by its variable projections and its choices for the generators.

2.1 Restriction and residual syntax

Fix $t : \Gamma \to \tau$. A static assignment a supplies typed literals for some variables of Γ. Write Γ_a for the remaining context, $D_a = X_{\Gamma_a}$ for its carrier, and $k_a : D_a \to X_\Gamma$ for insertion of the literals in their original positions. If $a \leq b$, meaning that b agrees with a and supplies at least its literals, insertion of the additional literals defines $k_{a,b} : D_b \to D_a$. These maps satisfy $k_a \circ k_{a,b} = k_b$ and $k_{a,b} \circ k_{b,c} = k_{a,c}$. Write $t[a] : \Gamma_a \to \tau$ for the term obtained by syntactic substitution of the literals.

Define a structural residualizer r_a as follows. Replace known variables by literals. Descend into primitives and compute a literal result when all arguments are literal. For a conditional, specialize the condition first. If it becomes a Boolean literal, specialize the selected branch. Otherwise preserve the conditional and specialize both branches. For a let binding, specialize the right-hand side. Extend the static assignment by the freshly bound variable when the result is a literal and replace the let by the specialized body. Otherwise keep the binding and specialize the body with that variable dynamic. Alpha-rename binders before extending an assignment. Each recursive call operates on a proper subterm of the original input, including when the static environment is extended at a let. Hence this residualizer terminates on finite input terms.

Specialization theorem. For every t and a, the specialized term is well typed, and

$$\mathrm{den}(r_a(t)) = \mathrm{den}(t) \circ k_a. \tag{1}$$

For $a \leq b$, put $b/a = b|_{\Gamma \setminus \mathrm{dom}(a)}$. Direct and staged specialization satisfy

$$\mathrm{den}(r_{b/a}(r_a(t))) = \mathrm{den}(r_b(t)). \tag{2}$$

To prove (1), induct on the term. The variable case follows from the coordinate formula for k_a. A folded primitive evaluates to the same element of W or B by definition of its literal evaluation. A preserved primitive composes with the interpretations of its specialized arguments. In a conditional with a literal residual condition, the induction hypothesis fixes the same branch in both interpretations for every dynamic input. With a dynamic condition, it fixes the condition value and the result of each branch. For a let with a literal residual right-hand side, that literal is its evaluated value at every dynamic input. Applying the induction hypothesis to the body in the extended static assignment yields the source value. With a dynamic right-hand side, apply the induction hypothesis to the body at the dynamic context extended by the right-hand side's evaluated value. All replacements have the types of the subterms they replace. Applying (1) twice and using the insertion identity proves (2). This theorem concerns observable word or Boolean values. Different compilation orders can yield different residual artifacts and costs.

The same proof applies to a binary64 fragment with pure, total addition, subtraction, and multiplication interpreted as Java `double` operations in the stated evaluation order. Its observable carrier distinguishes signed zeros and infinities and treats NaNs as a single semantic class. Literal substitution and folding must preserve that interpretation [11, Sections 4.2.3–4.2.4]. Real or modular arithmetic identities require new proofs in the binary64 carrier before using them as rewrites. For example, under Java evaluation, $((2^{53}+1) - 2^{53})$ evaluates to 0 whereas $(2^{53} + (1 - 2^{53}))$ evaluates to 1. For the circuit compiler, static construction supplies the complex coefficients and the scalar kernel applies the given sum-of-products order. The theorem preserves the chosen rounded operations. Approximation of exact Hilbert-space evolution is a further numerical question.

The first Futamura projection is another instance of restriction. If P is a set of well-formed first-order program descriptions with an agreed interface $D \to Y$, write $I : P \times D \to Y$ for a total interpreter on that interface. For $p \in P$, the denotation of a residual executable is $I \circ \langle p \circ !, \mathrm{id}_D \rangle$, with $p : 1 \to P$ and $! : D \to 1$. Partial evaluation tries to realize this restriction with interpreter dispatch removed. Truffle specializes an interpreter using a stable guest program and profiled state, while the typed residualizer above supplies a separate semantic model for source specialization [6, 7, 5, 1].

Correspondence with the implemented pass. The implementation allocates a distinct integer slot for each lexical binder, even when a nested binder reuses a name.

Each recursive result carries its free local slots. Correctness of that set follows from the singleton set for a residual local variable, the empty set for literals and inputs, union for ordinary constructors, and removal of the bound slot from the body of a retained let. The right-hand side is handled in the outer scope. Known slots produce only literals in residual terms. When the residual body has no free occurrence of the bound slot, its evaluation is independent of that coordinate. The right-hand side is pure and total, so the pass can erase the binding. The implementation also folds

$$x + 0, \quad 0 + x, \quad x - 0, \quad x \cdot 1, \quad 1 \cdot x$$

to x, and $x \cdot 0$ and $0 \cdot x$ to 0. These are identities in W, and purity and totality justify the erasure of an operand. The induction proving (1) therefore extends to `Specializer.residualize` on type-checked specifications.

Normalization and composition. In the raw, binder-free syntax, there is a stronger form of the composition law for a literal folder N that normalizes each child, then computes all-literal primitive applications and selects branches at literal conditions. Use a unique literal representation for each primitive value. For every well-typed simultaneous substitution σ,

$$N(t \circ \sigma) = N\big(N(t) \circ N(\sigma)\big). \tag{3}$$

Each elementary literal or branch rewrite preserves N. At the root this follows from the definition of N, including agreement of the rules on an all-literal conditional. In a larger term it follows because a node's normal form depends only on the normal forms of its children. Every term reduces to $N(t)$ by bottom-up rewriting. The rewrite rules are stable under substitution, since the literal data are closed and branch selection commutes with substitution into the selected branch. Thus $t \circ \sigma$ reduces first to $N(t) \circ \sigma$, then, by reducing substituted terms at each occurrence, to $N(t) \circ N(\sigma)$. This gives (3). Normalized tuples form a finite-product category with composition $g \star f = N(g \circ f)$: both bracketings of a triple composition become $N(h \circ g \circ f)$, identities and projections are variables, and tupling preserves normal form.

3 Coherent comparison of residual programs

The specialization theorem equates denotations. To retain a concrete reason why residual outputs agree, give each remaining context a category $\mathcal{R}_a$. Its objects are alpha-equivalence classes of terms of result type τ. Its arrows are finite sequences of elementary rewrite steps, with composition by concatenation. Use the contextual closures of arithmetic evaluation on literals, literal branch selection, hygienic let-substitution, and the listed word identities as rewrite rules. Take rewrite edges to record a rule instance and its position, and take the free category on this graph. Static substitution sends an instance to the same rule at the same position after substituting its open data, so it induces a functor $\mathcal{R}_a \to \mathcal{R}_b$ for $a \le b$. Here, substitution can be implemented by binding indices or by taking alpha-equivalence classes, so composition of these substitution functors is strict. A concrete $r_a(t)$ has a rewrite trace $p_a : t[a] \to r_a(t)$. For the static-let case, produce a trace for the right-hand side to its literal, apply the let rule, and continue with the body. A preserved let or conditional uses contextual rewrites. The let rule also erases a binding when its simplified body no longer uses it. This trace may be constructed from the structural algorithm independently of Truffle.

Let $K_a = |N\mathcal{R}_a|$ denote the space obtained by inverting all rewrite arrows. Equivalently, this is the groupoid completion of the rewrite category, a special case of relative localization [10]. Finite rewrite chains retain their classes as paths, and simplices of the nerve witness composition. Since each rewrite preserves denotation, interpretation factors through K_a to the discrete space of functions $D_a \to \mathrm{den}(\tau)$. The map to the discrete space is compatible with substitution. Its fibers can depend on the chosen rewrite presentation. For example, the category generated by two distinct parallel elementary rewrite edges between two objects has a circle as its groupoid completion. Additional proof relations can identify such paths when justified by a more refined rewrite theory.

Coherence theorem. Let P_t be the poset of static assignments ordered by refinement and let $K : P_t \to \mathcal{S}$ be the diagram $a \mapsto K_a$. The witnesses p_a extend to a coherent family of residual terms with equivalences over every refinement. Over $a \le b$, the comparison is represented by

$$p_b \circ k_{a,b}^*(p_a)^{-1} : k_{a,b}^* r_a(t) \simeq r_b(t). \tag{4}$$

The functorial family of unsimplified terms $a \mapsto t[a]$ gives a section of the left fibration $E \to NP_t$ classified by K via unstraightening [9, Section 2.2]. In each fiber the trace p_a gives a path from that section's value to the desired term. To see that all coherences can be chosen together, write $N = NP_t$ and $N_0 \subset N$ for its vertex subcomplex. Let J be the nerve of the groupoid on two isomorphic objects. A path in a Kan fiber extends to a map from J, since $\Delta^1 \hookrightarrow J$ is a trivial cofibration for the Kan model structure. The section and the chosen fiber paths define a map over N on

$$(N \times \{0\}) \cup_{N_0 \times \{0\}} (N_0 \times J).$$

The inclusion into $N \times J$ is a trivial cofibration in the Joyal model structure. It is the pushout product of $N_0 \subset N$ and the trivial cofibration $\{0\} \subset J$ [9, Corollary

2.2.5.4]. The map $E \to N$ is an inner fibration, and equivalences in the base, which are identities, lift. Hence it is a fibration between quasi-categories in the Joyal model structure and admits a lift on $N \times J$ [9, Corollary 2.4.6.5]. Its endpoint at 1 is the desired section and the homotopy gives the coherent comparisons. Transport in the left fibration applied to the resulting natural-equivalence square yields (4). In particular, for $a \le b \le c$, the middle witness cancels up to homotopy, and the lift supplies the compatibilities for longer chains. The trace from $k^*_{a,b} r_a(t)$ to $r_{b/a}(r_a(t))$ compares the staged residual to the direct residual through (4). Passing to discrete denotations recovers (2).

The theorem applies to the specified source rewrite theory. To include a Truffle optimization as another edge, establish its correctness judgment and compatibility with substitution for the chosen semantics.

4 Internal families, descent, and guarded execution

Let $\mathcal{X}$ be an ∞-topos and take a program denotation $F : S \times D \to Y$. In the slice $\mathcal{X}_{/S}$, a static context is an object $a : A \to S$. Its residual denotation is $F_a = F \circ (a \times \mathrm{id}_D)$. Equivalently F_a determines a map $A \times D \to A \times Y$ over A. For $u : (A', a') \to (A, a)$ in the slice, pullback identifies $F_{a'}$ with $u^* F_a$ through the specified homotopy between a' and $a \circ u$. Changing the static context gives a contravariant functor on denotations whose higher coherence is the functoriality of pullback. The term interpretation above takes values in the 0-truncated part of the model with $\mathcal{X} = \mathcal{S}$.

Consider a presheaf of candidate residual artifacts $R : (\mathcal{X}_{/S})^{\mathrm{op}} \to \mathcal{S}$ with a natural semantic map to

$$H(A) = \mathrm{Map}_{\mathcal{X}}(A \times D, Y).$$

Here a point of $R(A)$ is an artifact, and paths identify artifacts according to the chosen execution or proof semantics. The family F_a is a section $s_F : 1 \to H$. Define the presheaf $C_F = R \times_H 1$ using the homotopy pullback. An element of $C_F(A)$ carries both an implementation and a path from its denotation to F_a. Higher paths include compatibility between certificates. Its restriction maps follow simultaneously from R, H, and the section s_F.

Certification descent. If R satisfies descent along the Čech nerve of an effective epimorphism $U \to A$ in $\mathcal{X}_{/S}$, then so does C_F. Indeed, cartesian closure identifies $H(A)$ with $\mathrm{Map}_{\mathcal{X}}(A, Y^D)$. The Čech nerve $U_\bullet$ realizes to A, so mapping into Y^D turns it into a limit diagram [9, Corollary 6.2.3.5]. The constant terminal presheaf also has this property. Limits commute with the pullback defining C_F, giving

$$C_F(A) \simeq \lim_{[n] \in \Delta} C_F(U_n). \tag{5}$$

Thus an implementation with coherently matching certificates descends along the cover. Given a point of the limit, the space of global lifts equipped with identifications with that point is contractible, since it is the homotopy fiber of an equivalence. Applying the theorem to executable artifacts requires specifying their gluing semantics and establishing that it satisfies descent.

Representable implementation families. An instance that supplies this structure uses input, observation, and candidate families D_S, Y_S, C in $\mathcal{X}_{/S}$. These may vary with the static parameter. Interpret execution as $e : C \times_S D_S \to Y_S$, and write $f : D_S \to Y_S$ for the specification. Transposing in the locally cartesian closed slice gives $e^\sharp : C \to [D_S, Y_S]_S$ and $f^\sharp : S \to [D_S, Y_S]_S$. Model the family of witnessed residuals by the homotopy pullback

$$S \times_{[D_S, Y_S]_S} C.$$

Mapping into this object over S gives candidates and correctness paths over any context $A \to S$, by exponential adjunction and the defining property of the pullback. This assigns a space of witnesses to each context with all higher restriction maps induced by precomposition. It satisfies (5) because mapping out of a Čech colimit in the slice takes it to a limit. A relabeling between candidate representations lifts to a comparison between certified candidates when its execution homotopy is compatible with their correctness paths. Higher comparisons are given by the same homotopy pullback.

For a higher example take $\mathcal{X} = \mathcal{S}$, $S = S^2$, $D = *$, and $Y = K(\mathbb{Z}, 2)$. Let F represent the generator of $H^2(S^2; \mathbb{Z})$ and let $R = 1$ supply one candidate with constant basepoint denotation over every A. At a point of S, the certificate space is equivalent to $\Omega K(\mathbb{Z}, 2) \simeq K(\mathbb{Z}, 1)$ after choosing a path to the basepoint. Take U to be the disjoint union of the complements of the north and south poles, mapped to S^2. Each complement is contractible. The map $U \to S^2$ is an effective epimorphism in spaces because it is surjective on components. Both restrictions of F are nullhomotopic, so $C_F(U)$ is inhabited. A global certificate would nullhomotope a generator, which is impossible. Equation (5) identifies the obstruction as failure to complete these local certificates to a compatible Čech diagram. This example tests the semantics of higher certificates. The executable word-valued compiler uses the discrete instance.

Guarded refinement. For a fixed context A, let $g : A \times D \to 1 \sqcup 1$ be a decidable guard. Its two pullbacks split $A \times D$ as $G \sqcup \bar{G}$, since coproducts in an ∞-topos are disjoint and universal. Let $J : G \to Y$ be the fast branch, $s : \bar{G} \to Q$ a checkpoint on guard failure, and $j : Q \to Y$ its generic continuation. Given paths $J \simeq F_a|_G$ and $j \circ s \simeq F_a|_{\bar{G}}$, the dispatcher $[J, j \circ s]$ is observationally equivalent to F_a. The mapping space out of the coproduct is the product of the branch mapping

spaces, where the two certificates form the two components of the desired path. Both the guard decomposition and the assembly of certificates commute with pullback.

An entry guard can checkpoint the original arguments. After observable work, the checkpoint hypothesis demands a simulation relating optimized state to an interpreter state at a valid continuation, and preservation of any trace already produced. For instance, restarting the call after a write can repeat the write even when the return value agrees. A mutable speculative assumption also needs valid synchronization through its use when calls and invalidation can occur concurrently.

5 From residual syntax to Truffle calls

Two stages. The compiler takes a checked algorithm t and static parameters s in its declared scalar type. It substitutes the parameters, simplifies the resulting term, and lowers the residual tree to a Truffle `RootNode`. Each executable kernel holds the call target of that root. For dynamic inputs d, the call target executes the lowered tree through Truffle. Graal can specialize the interpreter with respect to the root and its stable structure. Schematically,

$$\begin{aligned} r_s &= \mathrm{simplify}(t[s]), \\ K_s &= \mathrm{callTarget}(\mathrm{lower}(r_s)), \\ K_s(d) &= \mathrm{exec}_{\mathrm{Truffle}}(r_s, d). \end{aligned}$$

The first stage fixes source-level parameters. The second stage uses the constructed AST as static interpreter state, in the sense of Truffle's first-Futamura-projection architecture [1, 5]. Constructing K_s yields an executable value immediately. Compilation to machine code depends on the engine, the runtime tier, the workload, and compilation heuristics [3].

Lowering invariant. Input and local nodes read distinct logical environments. Constants and input indices are fixed at construction. For `i64`, Java `long` addition, subtraction, and multiplication compute in $W = \mathbb{Z}/2^{64}\mathbb{Z}$ and comparison uses signed order. The `f64` case decodes the stored bits, performs the corresponding Java `double` operation, and re-encodes numerical results. The representation preserves signed zero and canonicalizes NaN payloads. The internal representations of false and true are 0 and 1, with the type checker controlling their use. A conditional evaluates its test and only the selected branch. A `LET` first evaluates its definition and writes the result to its assigned frame slot. References in the body read the frame slot for their lexical binder. A fresh execution frame prevents previous calls from supplying local values.

For each subterm u, let η map its free local variables to assigned slots, and let F be a frame with dynamic arguments d and slot values representing an environment ρ. The lowering invariant is

$$\mathrm{run}(\mathrm{lower}_\eta(u), F) = \mathrm{encode}(\mathrm{den}(u)(d, \rho)).$$

The frame after execution agrees with ρ on the slots for free variables of u. For a constant, input, or local, the invariant is the defining read operation. Arithmetic and comparisons follow from the inductive hypotheses for their operands and the declared scalar operations. For a conditional, the test invariant selects the same branch on both sides and that branch's invariant gives the result. For a binding, a slot distinct from the currently live slots becomes the location of the bound value. The body invariant then applies to the environment extended by that value. At the root the local environment is empty. Combined with specialization correctness, the invariant proves $K_s(d) = \mathrm{den}(t)(s, d)$ for the interpreter on well-typed inputs, assuming adequate runtime resources. This is a source-to-interpreter argument. The further claim about optimized execution depends on the Truffle and Graal implementation preserving the interpreter's behavior.

Runtime assumptions and transfer. Truffle treats the root of a compiling call target as a partial-evaluation constant. Stable fields reached from that root, including AST children declared with `@Child` and `@Children`, can make interpreter dispatch sites monomorphic and enable inlining. Dynamic arguments can select different paths in the same compiled kernel. In this prototype, each conditional passes its test value through a Truffle `CountingConditionProfile`. The profile records branch frequencies and returns the test value. The engine can speculate that an unobserved case remains untaken and invalidate that speculation if a later input exercises the branch [4, 1, 5].

For a simplified model of guarded optimization, let E be the interpreter configurations for a given residual term with observable result $O\colon E \to W$. Let $j\colon U \hookrightarrow E$ include the states satisfying a guard, and let $M\colon U \to W$ denote the guarded machine fragment. The fragment is sound on its domain exactly when $M = O \circ j$. If the guard fails, a deoptimization path must restore an interpreter configuration with the same pending observation. Thus if reentry maps an optimized state q to $\delta(q) \in E$, its semantic obligation is $O(\delta(q)) = O_{\mathrm{pending}}(q)$, where O_{pending} is specified by the source computation at the corresponding input and continuation. Together, this obligation and the domain equality suffice for observational equivalence of guarded execution with fallback and interpretation. The observation here records the returned word. The engine must implement the state transfer, including reentry at the appropriate continuation.

The prototype builds a fresh tree on a miss in the module's bounded cache of static tuples and fixes the

substituted constants for that kernel's lifetime. Truffle offers mechanisms for interpreters that maintain mutable speculation state. A write to a `@CompilationFinal` field on which compiled code relies must be accompanied by invalidation. The documentation describes local transfer via `transferToInterpreterAndInvalidate` and dependency tracking through a Truffle `Assumption` for externally invalidated facts [1].

The polyglot boundary. Evaluating a `catpe` specification supplies a module with `compile`. Calling `compile` with the static tuple yields the executable K_s, which exports Truffle interoperability messages. A guest can hold K_s and call it repeatedly with the dynamic tuple. JavaScript and Python use the same resident implementation through the executable-value protocol [2]. The boundary checks the arity and uses Truffle's `asLong` or `asDouble` to convert the guest arguments to the declared scalar type. The correctness statement compares values after those lossless conversions. JavaScript `Number` rounds some integers beyond its safe-integer range, so exact tests of those words require exact foreign representations and explicit checks on the returned value.

The executable value supplies a concrete guarded dispatch example. Its direct-call specialization is applicable when the receiver's call target equals the call node's cached target. The generic path uses an indirect call to the receiver's target. Both paths perform the same arity and word conversions. On the equality guard they invoke the same target with the same arguments, which supplies the local agreement needed for this dispatch optimization.

6 Compiling two families of algorithms

Let $R = \mathbb{Z}/2^{64}\mathbb{Z}$. An external signed word denotes its class in R. Order comparisons use $\operatorname{sgnrep}\colon R \to [-2^{63}, 2^{63}-1] \cap \mathbb{Z}$, the signed representative, followed by the usual integer order. The two domain compilers in `Workloads.java` construct typed source terms. Static data and dynamic samples use separate input ports. Both compilers build balanced trees for sums, using 0 for the empty sum. Associativity in R ensures that the chosen grouping preserves the result even on inputs whose ordinary integer sums would overflow.

6.1 A bank of finite impulse response filters

A specification fixes a window of length $m \geq 2$ and a finite, nonempty ordered bank. For a bank entry b, the shape gives a symmetry sign $\epsilon_b \in \{1, -1\}$, while the static inputs include the bank selector, every independent pair coefficient $c_{b,i}$, and the center coefficient if needed. Put $q = \lfloor m/2 \rfloor$ and $j(i) = m - 1 - i$. For $0 \leq i < q$, the coefficient vector satisfies

$$h_{b,i} = c_{b,i}, \qquad h_{b,j(i)} = \epsilon_b c_{b,i}.$$

An odd symmetric window takes its center coefficient as a static input. An odd antisymmetric window chooses center zero as part of the specification. This choice fixes the center even though $2c = 0$ also admits a nonzero class in R.

The generator emits a bank selector built from equality tests over the static bank index. Each pair is compiled to $c_{b,i}(x_i + \epsilon_b x_{j(i)})$, with addition or subtraction determined by the bank shape. A coefficient-zero conditional wraps each contribution and a static bank dispatch wraps the bank terms. Specialization therefore keeps a single bank branch and removes every pair whose coefficient is zero. The named-parameter binder validates the index. The generic source term is total, with default result zero for an index outside the bank.

For each valid static assignment and each dynamic sample window x, the convolution dot product agrees with the result of the selected residual function.

$$\begin{aligned}\sum_{i=0}^{m-1} h_{b,i} x_i &= \sum_{i=0}^{q-1} \bigl(c_{b,i} x_i + \epsilon_b c_{b,i} x_{j(i)}\bigr) + \delta_b(x) \\ &= \sum_{i=0}^{q-1} c_{b,i}(x_i + \epsilon_b x_{j(i)}) + \delta_b(x).\end{aligned}$$

Here $\delta_b(x)$ is the center contribution when present and zero otherwise. The first equality partitions the coordinate projections into pairs and a possible singleton. The second is distributivity in R. Each remaining summand uses at most one multiplication per pair. The domain compiler constructs this factored expression before CATPE specializes it.

6.2 An ensemble of decision trees

A forest specification fixes the number of sample ports, an ordered set F of features, and a list of finite binary tree shapes. Each feature $i \in F$ specifies two input projections π_{a_i} and π_{b_i}. Static scale s_i and bias d_i give the affine feature

$$\alpha_i(x) = \pi_{a_i}(x) + s_i \pi_{b_i}(x) + d_i \in R.$$

The shapes are the initial algebra of $X \mapsto 1 + F \times X \times X$ in sets. An occurrence of a leaf receives a static output word. An occurrence of a split receives a static cut word and compares the indicated feature to it. Each tree also has a static inclusion flag, with zero meaning disabled. The source uses a nested chain of `let` bindings for feature values. The branches refer to the bound features, so a surviving feature is shared by every split citing it.

For a fixed assignment to all settings, define the score of each tree recursively. A leaf yields its assigned word.

A split with feature i, cut τ, and children L, H yields L's score when $\mathrm{sgnrep}(\alpha_i(x)) < \mathrm{sgnrep}(\tau)$ and H's score otherwise. The ensemble score is the sum of all enabled scores. This semantics factors through the product map $x \mapsto (\alpha_i(x))_{i \in F}$. The lowering of a leaf is its parameter projection. The lowering of a split composes the relevant feature projection, the signed comparison, and source conditional selection. Induction on the initial-algebra constructors proves equality with the recursive score for every tree. Applying the flag tests and summing then proves equality for the forest. Static specialization removes disabled subtrees. If a feature is unused throughout the residual forest, purity and totality justify removing its binding.

The two examples require different categorical structure in the lowering. The filter expresses a ring polynomial using paired coordinate projections. The forest folds a branching inductive shape into a family of morphisms and factors the result through a common product of derived features. Each construction lands in the same typed source fragment, and each residual uses the same Truffle invocation interface.

7 Compiling finite-dimensional quantum evolution

Let H be a finite-dimensional Hermitian matrix and fix units with $\hbar = 1$. A time interval t specifies $U(t) = \exp(-itH) \colon \mathcal{H} \to \mathcal{H}$. By the spectral theorem, $U(t)^* U(t) = I$ and $U(t+s) = U(t)U(s)$. Hamiltonians with different spectra induce different families of specialization constants. The state is dynamic, so an individual compiled evolution can be applied to any state of the required dimension or repeatedly reused in a time-stepping loop.

The functor of restriction of scalars from complex to real vector spaces sends $U = A + iB$ to

$$\mathcal{R}(U) = \begin{pmatrix} A & -B \\ B & A \end{pmatrix}.$$

The ordered basis used by the code interleaves the real and imaginary coordinates. The displayed block order and the stored order differ by a permutation of coordinates. For $z = x + iy$, $Uz = (Ax - By) + i(Ay + Bx)$. This identity follows by bilinearity and $i^2 = -1$. It implies $\mathcal{R}(UV) = \mathcal{R}(U)\mathcal{R}(V)$ and $\mathcal{R}(U^*) = \mathcal{R}(U)^\top$. In particular realification takes a unitary to an orthogonal operator. This algebraic property justifies the translation from a complex map to paired real linear forms.

The compiler constructs a pair of term templates per row. For row j the two static parameter lists hold its A_{jk} and B_{jk}. The real and imaginary outputs are built from

$$r_j(x, y) = \sum_k (A_{jk} x_k - B_{jk} y_k),$$
$$i_j(x, y) = \sum_k (A_{jk} y_k + B_{jk} x_k).$$

The specification uses a balanced sum with a fixed operation order. The runtime binds the coefficients, producing $2 \dim \mathcal{H}$ executable roots. A guest callable vector step holds these roots and supplies the same state to each. The method uses dense storage and arithmetic, $O((\dim \mathcal{H})^2)$ for a step after coefficient generation. The precomputation of $U(t)$ is part of the respective domain compiler. Changing t binds new coefficients.

In binary64 this construction evaluates the arithmetic expression that its source specifies. Orthogonality and group composition hold in exact real arithmetic. Computing the coefficients, performing products and sums, and applying many steps introduce rounding. We therefore measure defects of norm preservation, inverse recovery, and composition and compare both examples to separately computed dynamical references. Both quantum examples also measure energy conservation.

8 Compiling free-electron time evolution

With $\hbar = 1$, consider the spatial wavefunction of one free electron on a periodic grid with $N \geq 3$ sites, spacing $a > 0$, and mass $m > 0$. The Hamiltonian is spin independent, so this evolution acts on either spin component. The input to each compiled evolution is a state $\psi \in \mathbb{C}^N$ with interleaved real and imaginary coordinates. The discretization uses the centered second difference on $\mathbb{Z}/N\mathbb{Z}$. Indices are cyclic and the Hamiltonian is

$$(H\psi)_j = \frac{2\psi_j - \psi_{j-1} - \psi_{j+1}}{2ma^2}.$$

The lattice evolution solves $i\partial_t \psi = H\psi$ in continuous time. The grid and finite volume specify the spatial model. In particular, for a fixed physical wavenumber $\kappa = 2\pi k/(Na)$ along grids of fixed length, its dispersion approaches the continuum value $\kappa^2/(2m)$ with an error of order a^2 as $a \to 0$.

The normalized Fourier basis diagonalizes every circulant operator [15]. For $\omega = \exp(2\pi i/N)$ and $v_k(j) = N^{-1/2}\omega^{jk}$, the two cyclic shifts multiply v_k by ω^k and ω^{-k}. This gives

$$Hv_k = E_k v_k, \qquad E_k = \frac{2}{ma^2} \sin^2\left(\frac{\pi k}{N}\right).$$

Let F denote the unitary forward Fourier transform

whose k, j entry is $N^{-1/2}\omega^{-kj}$. Then

$$U(t) = F^* \operatorname{diag}(e^{-itE_k})F,$$
$$U(t)_{j\ell} = \frac{1}{N}\sum_{k=0}^{N-1} \omega^{k(j-\ell)} e^{-itE_k}.$$

The first identity uses the scalar differential equation in each mode. The second gives a kernel depending only on the displacement $j - \ell$ modulo N. The compiler evaluates the N displacement coefficients before binding the static parameters for time t.

Exactly, $U(0) = I$, $U(t + s) = U(t)U(s)$, and $U(t)^* = U(-t)$. These equalities follow by multiplying diagonal factors. They give a continuous functor $B(\mathbb{R}, +) \to B\mathrm{U}(N)$ between one-object topological categories. Passing to their nerves yields coherent composition of the time-indexed algorithms. The distinction between a static time setting and a dynamic state is visible in the family $\mathbb{R} \times \mathbb{C}^N \to \mathbb{C}^N$ with value $(t, \psi) \mapsto U(t)\psi$. It is the state argument that remains after specialization.

Writing $U = A + iB$, the real-linear image of U sends (x, y) to $(Ax - By, Bx + Ay)$. This realification preserves composition and sends the complex adjoint to the real transpose. Matrix entries are bound as binary64 static data, with one residual scalar kernel per real or imaginary output coordinate. This factorization separates the Fourier calculation of the operator from repeat invocation on different input states. The source emits a fixed order of floating-point operations. The exact functorial equalities belong to the mathematical operator, and the binary64 kernels approximate them to numerical tolerances on the measured inputs.

An independent reference path evolves a state by performing the forward transform, phase rotation in frequency space, and inverse transform directly. For a test with a nonconstant phase and a localized envelope, the generator supplies the normalized state

$$\psi_j = C \exp\left(-\frac{d(j, c)^2}{4\sigma^2}\right)\omega^{pj},$$

where $d(j, c) = \min(|j - c|, N - |j - c|)$, c is its center, $\sigma > 0$ is its width in sites, and p is a carrier mode. The exact evolution preserves $\|\psi\|^2$ and $\langle\psi, H\psi\rangle$ since $U(t)$ is unitary and commutes with H. The energy can be checked in position space using the separate formula

$$\langle\psi, H\psi\rangle = \frac{1}{2ma^2}\sum_{j=0}^{N-1} |\psi_{j+1} - \psi_j|^2.$$

Expanding the squares and using cyclic indexing proves this identity. The direct energy computation and the reference Fourier calculation exercise different numerical routes through the model. Applying the negative-time operator gives a further check of reconstruction. The lattice calculation uses no temporal stepper, so discrepancies in these checks reflect floating-point effects or implementation defects.

9 Interacting fermions on an open chain

The field compiler starts from a finite lattice Hamiltonian with L spinless sites. The fermionic Fock space is $\bigoplus_{p=0}^{L} \bigwedge^p \mathbb{C}^L$. Occupation basis vectors are indexed by bit masks m, with the jth bit indicating whether site j is occupied. Ordered creation operators fix the sign convention. Let c_j annihilate a fermion at site j and let $c_j^\dagger$ create one there. On a basis vector, an admissible action multiplies by $(-1)^{\sum_{k<j} m_k}$ before changing bit j. An inadmissible action yields zero [16].

Write $n_j = c_j^\dagger c_j$. With real hopping g, interaction v, and onsite energies μ_j, the open-chain Hamiltonian is

$$H = \sum_{j=0}^{L-1} \mu_j n_j + v\sum_{j=0}^{L-2} n_j n_{j+1} - g\sum_{j=0}^{L-2}(c_{j+1}^\dagger c_j + c_j^\dagger c_{j+1}).$$

The density-density term assigns energy to pairs of occupied adjacent sites and is quartic in the field operators. For an allowed hop from j to k, the compiler computes the annihilation sign in the original occupation and the creation sign in the intermediate occupation. Both signs matter even though their product is positive for adjacent sites in this ordering. The onsite and interaction terms preserve occupation. Each hop removes one particle and adds one, so H preserves every eigenspace of $\sum_j n_j$. On the p-particle sector $\mathcal{H}_p$, the matrix H_p has dimension $\binom{L}{p}$ and is real symmetric. The inclusion $\iota_p : \mathcal{H}_p \hookrightarrow \mathcal{F}$ satisfies $H\iota_p = \iota_p H_p$. Taking powers and summing the exponential series proves

$$e^{-i\tau H}\iota_p = \iota_p e^{-i\tau H_p}.$$

The compiler therefore works in the chosen sector without approximating transitions to other particle numbers.

In exact arithmetic, $U_p(\tau) = \exp(-i\tau H_p)$ is a unitary evolution, with $U_p(0) = I$ and $U_p(\tau + \sigma) = U_p(\tau)U_p(\sigma)$. This is a functor from the one-object category of the additive time group to the category of finite-dimensional complex Hilbert spaces and linear maps. Realification sends a complex-linear operator to the real-linear operator on its real and imaginary coordinates and preserves composition and adjoints. If $U_p = A + iB$, its action in those coordinates is

$$(x, y) \longmapsto (Ax - By, Bx + Ay).$$

This yields the scalar-output lowering. The interface interleaves real and imaginary coordinates and evaluates one CatPE `f64` kernel for each output coordinate. The dynamics still act on all amplitudes together through the shared input tuple.

The static phase uses Apache Commons Math to diagonalize the real-symmetric H_p [17]. If $H_p = Q\Lambda Q^\top$ with orthogonal Q, then $U_p(\tau) = Q\exp(-i\tau\Lambda)Q^\top$. The compiler binds the resulting binary64 matrix entries as static coefficients. The dynamic phase receives only the input amplitudes. Arithmetic rounding applies to the eigendecomposition, phase evaluation, and generated dot products. Consequently, the numerical implementation checks conservation and agreement within tolerances. The identities for number sectors and composition of the exact operators are mathematical statements about the model.

The reproducible fixture uses $L = 4$, $p = 2$, $g = 0.8$, $v = 1.25$, and onsite energies $(0.1, -0.2, 0.3, -0.1)$. Its six masks, in order, are $(3, 5, 6, 9, 10, 12)$. The initial vector for mask 3 places two fermions on the first two sites. An independent check evaluates the degree-48 Taylor polynomial in $-i\tau H_p$ by repeated matrix-vector products. For $r = \|\tau H_p\|_\infty \le 4$, its exact remainder is at most $e^r r^{49}\|\psi\|_\infty/49!$. Indeed, submultiplicativity bounds the tail by $\|\psi\|_\infty \sum_{k\ge 49} r^k/k!$, and $(49 + j)! \ge 49!j!$. This bounds the reference truncation error separately from floating-point roundoff. The construction targets finite lattices and dense evolution matrices. It enumerates 2^L masks and considers $O(Ld)$ hops for $d = \binom{L}{p}$. Dense matrix storage is quadratic in d, and spectral construction scales cubically. The sector dimension grows exponentially at fixed filling strictly between zero and one.

10 Time-resolved observables and an interaction control

Both figures sample $t = 0, 0.04, \ldots, 4$. Each sample binds the matrix coefficients for that absolute time and applies the resulting CatPE kernel family to the original state $\psi(0)$. Thus the sampling interval sets the resolution of the plots. The independent reference calculations provide separate checks of the amplitudes. The displayed probabilities are computed from the Graal execution as squared moduli, which retain the population of each configuration while summing to one for normalized inputs under exact evolution.

Free packet on the ring. Figure 1 tracks $\rho_j(t) = |\psi_j(t)|^2$ on the 12-site ring. The initial packet is centered at site 3 with width 1.3 sites and carrier mode 2. With the orientation from j to $j + 1$, the bond probability current and discrete continuity equation are

$$J_j(t) = \frac{\mathrm{Im}(\overline{\psi_j(t)}\psi_{j+1}(t))}{ma^2},$$
$$\frac{d}{dt}\rho_j(t) = J_{j-1}(t) - J_j(t).$$

These follow by differentiating $|\psi_j|^2$ and inserting the centered second-difference Hamiltonian. Summing the differences over the periodic grid cancels the current entering and leaving each bond. The conserved net oriented current $\sum_j J_j$ can also be computed from the Fourier weights since each mode's magnitude remains constant.

The strongest density moves from site 3 to site 10 between the first and last samples. Its maximum decreases from about 0.307 to 0.213 as the packet spreads. The dimensionless carrier wavenumber $q_0 = 2\pi(2/12)$ has positive slope in the dispersion relation, $E'(q_0) = \sin(q_0)/(ma^2)$, which gives about 1.77 sites per unit time. This number describes the carrier's group velocity. Each Fourier mode has its own phase evolution, so the peak location and width reflect the full mode distribution. Site indices are interpreted modulo 12 throughout.

Joint occupancies in the field. For the four-site chain with two particles, let $\psi_{jk}(t)$ be the occupation-basis amplitude with sites $j < k$ occupied. The joint occupation probability and the marginal at site j are

$$P_{jk}(t) = |\psi_{jk}(t)|^2 = \langle n_j n_k\rangle_t,$$
$$\langle n_j\rangle_t = \sum_{k<j} P_{kj}(t) + \sum_{k>j} P_{jk}(t).$$

Each configuration contributes to two site marginals, so the joint probabilities sum to one and the marginals sum to two. The associated connected correlation is $P_{jk} - \langle n_j\rangle_t\langle n_k\rangle_t$. Fixed particle number can give a nonzero connected correlation even at $v = 0$. Summing $P_{j,j+1}$ gives the probability $A_v(t)$ of an adjacent pair. Equivalently, $A_v(t) = \langle D\rangle_t$ for $D = \sum_{j=0}^{2} n_j n_{j+1}$, a projection in this sector. Figure 2 compares this observable at $v = 1.25$ and $v = 0$, using $g = 0.8$, identical onsite potentials, and the same initial mask 3 with occupied sites $0, 1$. Initial energies in the two Hamiltonians differ by v because $D\psi(0) = \psi(0)$. Each run conserves its own energy.

Both adjacent-pair curves start at one and initially satisfy $A_v(t) = 1 - g^2t^2 + O(t^4)$, since only the hop from site 1 to site 2 can immediately create a separated pair. The interaction first contributes to the difference at fourth order. Set $\delta = \mu_1 - \mu_2$. Expanding the amplitude of the separated configuration $\{0, 2\}$ through cubic order gives

$$A_v(t) - A_0(t) = \frac{g^2}{12}\big((\delta + v)^2 - \delta^2\big)t^4 + O(t^6).$$

Indeed, the linear amplitude is the shared hop. At quadratic and cubic order the diagonal energies of the incident configurations enter, with gap $\delta + v$, while pure hopping contributions are shared. Other separated configurations are at least two hops away and have interaction-independent quartic probabilities. Reality of the Hamiltonian and the initial state makes each probability even in time. The parameters above give a leading

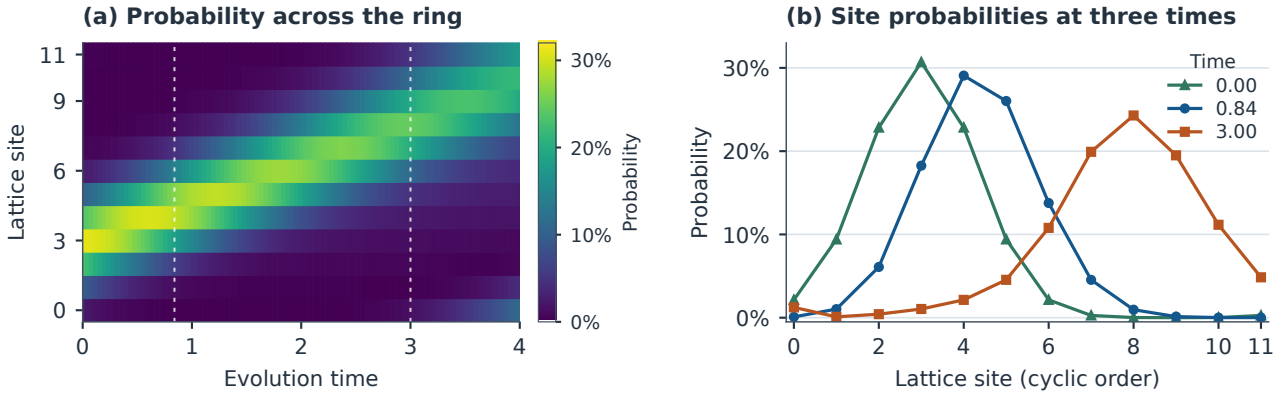


Figure 1: Probability from the compiled periodic free-electron evolution. (a) Site probability $|\psi_j(t)|^2$ at all 101 sampled times. The upper and lower sites are neighbors on the ring. (b) Three slices show the evolving spatial profile. The dashed lines in (a) locate the later two slices.

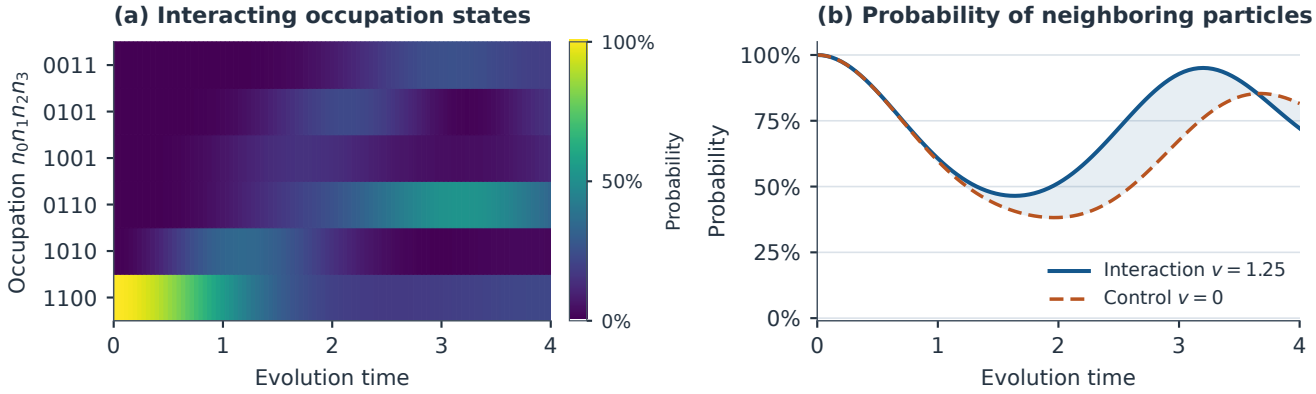


Figure 2: Probability from the compiled two-particle field. (a) The probability for each occupation of the four sites with interaction $v = 1.25$. The digits in $n_0n_1n_2n_3$ list sites from left to right. (b) The probabilities that both particles occupy neighboring sites for $v = 1.25$ and the $v = 0$ control. Both runs begin with sites 0 and 1 occupied and use the same hopping and onsite energies.

difference of $t^4/60$, consistent with the observed difference of 4.262×10^{-8} at $t = 0.04$. At the largest sampled separation, at $t = 2.76$, the adjacent-pair probabilities are about 0.847 with interaction and 0.566 in the control. At $t = 4$ the probabilities are 0.721 and 0.816, respectively. The effect measured here is a redistribution among occupation configurations in a closed, coherently evolving finite chain.

10.1 Time sweeps as restrictions of observations

Fix a Hamiltonian H, a dynamic state space D, and an ordered set of observation labels L with an orthogonal projector P_ℓ for each $\ell \in L$. A label can denote a site, an unordered occupation pair, or the subspace spanned by all adjacent pairs. The static context consists of a time and an observation label. In the notation of Section 4, take $S = \mathbb{R} \times L$ and

$$F((t, \ell), \psi) = \|P_\ell U_H(t)\psi\|_2^2.$$

For a finite sample $A \subseteq S$, let $O_A : D \to \mathbb{R}^A$ collect the observations indexed by A. Time order and the chosen label order give the display order. Inclusions $A \subseteq B$ in the sampling poset give coordinate restrictions $\pi_{B,A} : \mathbb{R}^B \to \mathbb{R}^A$, with $\pi_{B,A}O_B = O_A$. The identity follows coordinate by coordinate, and restrictions compose along every chain of inclusions. The compiler implements the time restriction by binding the matrix coefficients for each sampled time. The residual rows still accept the state. A plotted sweep supplies the same initial state to those rows at each time and then takes the listed observations. Slicing the plot and evaluating only the selected coordinates therefore agree when the compiler inputs, arithmetic order, and observation rule are held fixed. The same commutation holds for the rounded

values produced by this independent evaluation of each time.

Time composition makes a different comparison. The exact propagators satisfy $U_H(t+s) = U_H(t)U_H(s)$. To compare a sequence of rounded steps with this law, write $x_0 = \psi$, $x_i = K_{\delta_i}(x_{i-1})$ for executions of compiled kernels and define $e_i = \|x_i - U_H(\delta_i)x_{i-1}\|_2$. With $y_i = U_H(\delta_1 + \cdots + \delta_i)\psi$,

$$\begin{aligned} x_i - y_i &= x_i - U_H(\delta_i)x_{i-1} \\ &\quad + U_H(\delta_i)(x_{i-1} - y_{i-1}), \\ \|x_n - y_n\|_2 &\le \sum_{i=1}^{n} e_i. \end{aligned}$$

The bound uses the triangle inequality and the exact unitarity of $U_H(\delta_i)$. Each e_i includes coefficient generation and arithmetic error at that step's actual input. The discrepancy with one compiled application at $t = \sum_i \delta_i$ is bounded by adding $\|K_t(\psi) - U_H(t)\psi\|_2$. For normalized ψ and a state error at most ε, the change in an exact projector probability is at most $(2+\varepsilon)\varepsilon$ by Cauchy–Schwarz and $\|P_\ell\| \le 1$. Computing that probability in binary64 adds its own rounding error. These bounds explain what the observed composition defects measure on a tested trajectory. A bound over all inputs would require uniform control of the local errors.

11 Reproducible checks

The artifact was tested on Linux x86-64 using GraalVM Community 24.0.2 and Truffle, GraalJS, and GraalPy 24.2.2. All eight JUnit tests passed in the traced build. The trace test requested synchronous compilation only for CatPE kernel roots. The Graal log reported completed tier-2 compilations of all three word kernels and of every output row in both quantum examples. In the forest run, the trace also recorded a deoptimization and a subsequent successful compilation. The test continued to compare results to the independent forest calculation.

Table 1: Word kernels on 4,096 input tuples per row.

Kernel	Source	Residual	Mismatches
Symmetric FIR	346	67	0
Antisymmetric FIR	346	65	0
Decision forest	91	57	0

The word tests use a fixed pseudorandom seed and alternate between full-width signed values and small values. Each tuple is evaluated by the unspecialized interpreter and the Truffle kernel, and both values are compared to a separate domain oracle. Table 1 reports node counts in source and residual terms. Both guest languages call all three word kernels on fixed samples. They also exercise an identity kernel on the signed maximum through foreign numeric representations, with the host recovering the exact word.

The free-electron test uses a 12-site lattice with $a = 0.7$, $m = 1$, and $t = 0.83$. Its initial packet has center 3, width 1.3, and carrier mode 2. The field test uses the six-dimensional sector and initial occupation of Section 9 at $t = 0.6$. The field matrix is also compared with explicit entries for the fixture. The experiments compare the output of the reusable compiled rows to the independent references in Sections 8 and 9. GraalJS and GraalPy invoke the same compiled row objects and return every component of both outputs. Their results agree component by component with the host calls.

Table 2: Absolute error for the quantum fixtures.

Measurement	Free electron	Fermion field
Reference	$1.665\,10^{-16}$	$6.661\,10^{-16}$
Norm squared	$2.220\,10^{-16}$	$8.882\,10^{-16}$
Energy	$2.220\,10^{-16}$	$8.882\,10^{-16}$
Inverse	$1.110\,10^{-16}$	$8.882\,10^{-16}$
Composition	$1.665\,10^{-16}$	$4.441\,10^{-16}$
Unitarity	$2.220\,10^{-16}$	$1.776\,10^{-15}$

Inverse error compares $U(-t)U(t)\psi$ to ψ. Composition error compares $U(t)^2\psi$ to $U(2t)\psi$. Vector discrepancies use the largest absolute difference in an interleaved real or imaginary component. To test unitarity across the entire finite basis, the test applies the compiled kernels to every real unit vector in the complex basis, reconstructs the complex columns, and computes the largest entrywise magnitude of $U^*U - I$. Table 2 reports the observed defects. Each check uses absolute tolerance $2\,10^{-12}$.

11.1 Validation across the time grid

The runtime generated the plotted values by evaluating the specialized CatPE rows at each time. For every sample we compared the amplitudes against a reference, checked the squared norm against one, and checked the energy against its initial value. The electron reference uses the direct Fourier action. The field references use successive applications of the 48th-degree Taylor polynomial for $\Delta t = 0.04$ to the initial vector. The spectral propagator used to generate the field kernels and the Taylor reference share a Hamiltonian matrix and use different numerical methods. The Taylor reference uses one step per sampling interval.

Table 3: Maximum discrepancies over 101 times for the compiled dynamics.

Model	Reference	Norm	Energy
Free electron	$3.89\,10^{-16}$	$6.66\,10^{-16}$	$8.88\,10^{-16}$
Field, $v = 1.25$	$2.40\,10^{-15}$	$1.11\,10^{-15}$	$1.55\,10^{-15}$
Field, $v = 0$	$1.44\,10^{-15}$	$1.78\,10^{-15}$	$2.91\,10^{-16}$

Table 3 gives maximum errors across the 303 compiled states from the three sweeps. Reference error uses the largest absolute difference of a real or imaginary amplitude component. The reproducibility archive includes the original CSVs, the Java exporter with an absolute tolerance of $2\,10^{-12}$, and the script that renders Figures 1 and 2.

12 Scope and limitations

The certification and gluing statements are mathematical theorems. The implementation realizes a first-order fragment with finite contexts, read-only inputs and static data, and lexical bindings. The software executes ordinary typed arithmetic. The higher path witnesses used in the theorems are mathematical objects rather than proof terms serialized by the runtime. Extending the language with effects, exceptions, unbounded iteration, or observation of machine representation would require a revised semantic equivalence and proof of the residual transformations.

The numerical examples are finite quantum simulations. The single electron is spinless with periodic boundary conditions and a finite-difference Hamiltonian. The interacting field models spinless fermions in a finite set of sites and a fixed number sector. Full continuum field simulation, spin, and electromagnetic radiation require different state spaces and operators. The size of a fixed-number sector grows binomially with the number of sites, which limits dense compilation.

The code measures the size of residual syntax and checks specified observations. A JIT compilation trace records the actual compilation event. Source and residual node counts do not estimate machine instruction counts or execution time. Correctness of the machine-code generation, the guest interoperability implementations, and their exception handling is an obligation of the underlying JVM and GraalVM implementations.

Artifact and disclosure. The accompanying source archive contains the implementations, guest scripts, build descriptor, test suite, simulation data, figure generator, and LaTeX sources. The manuscript and software were produced with AI assistance, including drafting, code generation, and verification. The responsible submitter uses an anonymous public byline.

References

[1] GraalVM. *Partial Evaluation in Truffle*. GraalVM documentation.

[2] GraalVM. *InteropLibrary*. Truffle API documentation.

[3] GraalVM. *RootNode*. Truffle API documentation.

[4] GraalVM. *CountingConditionProfile*. Truffle API documentation.

[5] T. Würthinger, C. Wimmer, C. Humer, A. Wöß, L. Stadler, C. Seaton, G. Duboscq, D. Simon, and M. Grimmer. Practical partial evaluation for high-performance dynamic language runtimes. *PLDI*, pp. 662–676, 2017. DOI.

[6] Y. Futamura. Partial evaluation of computation process—An approach to a compiler-compiler. *Computer Software* 21(5), pp. 343–351, 2004. DOI.

[7] N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993. Author's edition.

[8] R. Davies and F. Pfenning. A modal analysis of staged computation. *J. ACM* 48(3), pp. 555–604, 2001. Author's copy.

[9] J. Lurie. *Higher Topos Theory*, Annals of Mathematics Studies 170, Princeton University Press, 2009. Author's preprint.

[10] W. G. Dwyer and D. M. Kan. Calculating simplicial localizations. *J. Pure Appl. Algebra* 18(1), pp. 17–35, 1980. DOI.

[11] Oracle. *The Java Language Specification, Java SE 24 Edition*, Sections 4.2.2–4.2.4.

[12] Third Projection Collective. Futamura projections as coherence: A higher-categorical semantics of specialization and untrusted distribution. *Higher Categorical Semantics*, proposal, 2026. Journal record.

[13] Anonymous. Higher e-graphs for Futamura projections: Coherence obstructions, descent, and verified deployment. *Higher Categorical Semantics*, 2026. Journal record.

[14] Anonymous author. Caprice: Higher-categorical semantics of capture-aware partial evaluation and JIT matching. *Higher Categorical Semantics*, 2026. Journal record.

[15] G. Strang. The discrete cosine transform. *SIAM Review* 41(1), pp. 135–147, 1999. DOI.

[16] University of Oxford. *C6 lecture notes*, Section 1.2.1, equations 17–18, 2017. Lecture notes.

[17] Apache Commons Math 3.6.1. *EigenDecomposition*. API documentation.