Lateralus Language Specification v3.0

Formal edition: operational semantics, type soundness, and pipeline calculus

Lateralus Language

bad-antics · April 2026 · Lateralus Language Research

ABSTRACT This document is the formal specification companion to the Lateralus Language Specification v3. Where the main specification (lateralus-language-spec-v3) is normative and implementation-oriented, this document presents the formal foundations: a small-step operational semantics for the core language, a proof of type soundness via progress and preservation, the pipeline calculus denotational semantics, and the formal definition of the effect system. This document is intended for language researchers, implementers seeking to verify their implementation, and users of the formal verification tools.

1. Syntax of the Core Calculus

We define a core calculus λ_pipe, a subset of Lateralus sufficient to study the pipeline operators formally. The full language desugars to λ_pipe.

```
-- Terms
t ::= x                       -- variable
    | λx:τ. t                -- abstraction
    | t t                     -- application
    | t |> t                  -- total pipeline
    | t |?> t                 -- fallible pipeline
    | ok(t) | err(t)          -- Result constructors
    | match t {ok(x)=>t, err(x)=>t}  -- elimination
    | let x = t in t          -- let binding

-- Values
v ::= λx:τ. t | ok(v) | err(v)
```

2. Small-Step Operational Semantics

The reduction relation $t \to t'$ is defined by the following rules:

```
-- Beta reduction
(λx:τ. t) v  →  t[v/x]                        [E-App]

-- Total pipeline (desugar to application)
v |> w  →  w v                                 [E-Pipe]

-- Fallible pipeline: ok case
ok(v) |?> w  →  w v                            [E-PipeOk]

-- Fallible pipeline: err case (propagate without calling w)
err(v) |?> w  →  err(v)                        [E-PipeErr]

-- Congruence rules (evaluation contexts)
t → t'
■■■■■■■■■■
t |> u → t' |> u                               [E-PipeCong]
```

3. Type System

The typing judgment Γ ■ $t : \tau$ is defined inductively:

```
x : τ ∈ Γ
■■■■■■■■■■■■  [T-Var]
Γ ■ x : τ
```

```
Γ, x:τ■ ■ t : τ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■  [T-Abs]
Γ ■ λx:τ■. t : τ■ → τ■

Γ ■ t■ : A → B    Γ ■ t■ : A
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■  [T-App]
Γ ■ t■ t■ : B

Γ ■ t■ : A    Γ ■ t■ : A → B
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■  [T-Pipe]
Γ ■ t■ |> t■ : B

Γ ■ t■ : Result<A,E>    Γ ■ t■ : A → Result<B,E>
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■  [T-PipeFallible]
Γ ■ t■ |?> t■ : Result<B,E>
```

4. Type Soundness

Theorem (Soundness): If Γ ■ t : τ and t →* t', then either t' is a value of type τ, or t' → t'' for some t''.

Proof sketch via Progress and Preservation:

```
Lemma (Progress): If ■ t : τ (closed term), then either
  (a) t is a value, or
  (b) t → t' for some t'.
Proof: by induction on the typing derivation.
  Case T-Pipe: t = t■ |> t■, ■ t■ : A, ■ t■ : A → B.
    By IH on t■: t■ is a value v, or t■ → t■'.
    If t■ is a value v: t■ is a value (λx. body) by IH,
      so v |> (λx. body) → (λx. body) v by E-Pipe. (b) ✓
    If t■ → t■': t■ |> t■ → t■' |> t■ by E-PipeCong. (b) ✓
```

```
Lemma (Preservation): If Γ ■ t : τ and t → t', then Γ ■ t' : τ.
Proof: by induction on the reduction derivation.
  Case E-PipeErr: err(v) |?> w → err(v).
    Premise type: ■ err(v) |?> w : Result<B,E>.
    We need: ■ err(v) : Result<B,E>.
    The original err(v) had type Result<A,E>; E is the same.
    By T-Err with coercion: ■ err(v) : Result<B,E>. ✓
```

5. Denotational Semantics: Pipeline Calculus

The denotational model interprets λ_pipe in a cartesian closed category C with a monad M for the fallible case:

```
■τ■ → τ■■        = C(■τ■■, ■τ■■)   (hom-set)
■Result<A,E>■     = M_E(■A■)          (M_E = error monad)
■t■ |> t■■       = ■t■■ ■ ■t■■       (composition)
■t■ |?> t■■      = bind(■t■■, ■t■■)  (monadic bind)
```

Consequence: the total pipeline operator is composition in C; the fallible pipeline operator is monadic bind. This gives Lateralus pipelines the algebraic structure of a Kleisli category over the error monad M_E.

6. Effect System Formal Semantics

The effect system extends the type system with effect rows ε. The effect judgment Γ ■ t : τ ! ε reads: 't has type τ with effects ε'.

```
-- Pure computation: empty effect row
Γ ■ t : τ ! {}  (no effects)

-- IO effect introduction
Γ ■ sys_call(n) : u64 ! {IO}

-- Effect polymorphism in higher-order functions
Γ, x:A ■ t[x] : B ! ε
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
Γ ■ λx:A. t : A → B ! ε    [T-Abs-Effect]

-- Effect row subtyping
{IO} ≤ {IO, State S}   (subsumption: more effects is a supertype)
```

7. Equational Laws

The pipeline calculus satisfies 14 equational laws. The five most important for compiler optimization are:

```
-- L1: Identity (id is the unit of |>)
v |> id  =  v
id |> f  =  f

-- L2: Associativity (pipeline is associative)
(v |> f) |> g  =  v |> (f >> g)
where (f >> g)(x) = f(x) |> g

-- L3: Fusion (compose stages without allocation)
v |> map(f) |> map(g)  =  v |> map(f >> g)

-- L4: Fallible short-circuit
err(e) |?> f  =  err(e)

-- L5: Fallible unit
ok(v) |?> ok  =  ok(v)
```

8. Mechanization Status

The formal specification has been partially mechanized in Lean 4. The current status of the mechanization:

```
Component                         Status
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
Core calculus syntax              ✓ Complete
Operational semantics             ✓ Complete
Type system rules                 ✓ Complete
Progress lemma                    ✓ Proved
Preservation lemma                ✓ Proved
Denotational semantics            ■ In progress
Effect system                     ■ In progress
14 equational laws                ✓ 9/14 proved
Full language (desugaring)        ■ Planned
```

The mechanization source is available at github.com/bad-antics/lateralus-lean4. Contributions from the formal methods community are welcome.