Sumatra: Mechanizing Substructural Meta-theory in Rocq

Anonymous Author(s)

Abstract

Mechanizing substructural meta-theory requires careful management of contexts whose assumptions must be allocated among subderivations, consumed, and updated. While physically restructuring contexts complicates nameless syntax and simultaneous substitutions, representations that record an assumption's availability in an annotation simplify this bookkeeping. Existing frameworks, however, typically use one annotation for two distinct purposes: specifying an assumption's static *policy* (which structural rules it admits) and recording its dynamic *state* (whether, or how many times, it remains available). This conflation makes them harder to extend and reason about.

We present Sumatra, a Rocq library that separates these concerns into two independent annotations. On this foundation, the library develops a general theory of context operations, renaming, and simultaneous substitution, and proves a range of generic theorems about these operations once and for all. The library can then be instantiated to support specifying and reasoning about a wide range of object systems. Varying the two annotations yields the standard structural disciplines, mixtures of them within a single context, and quantitative grading. Five case studies, spanning natural-deduction and sequent calculi as well as a variety of proof techniques, demonstrate the scope of the library.

Keywords: proof assistants, mechanized meta-theory, substructural logic, simultaneous substitution, logical relations, de Bruijn indices, Rocq

1 Introduction

The structural rules of contraction, weakening, and exchange determine whether assumptions in a logical system may be duplicated, discarded, or rearranged. *Substructural* logics and type systems restrict one or more of these rules. Familiar examples include linear, affine, and relevant systems. Such systems play an increasingly central role in the study of resource-sensitive computation across a wide range of applications, including session-typed concurrency [10, 52], quantum programming [47], memory management [7, 31, 53], and data-flow analysis [23].

When mechanizing these systems, a proof assistant must represent how a context is divided between the premises of a rule, how assumptions are consumed, and how these operations interact with renaming and substitution. Most existing developments address these problems with infrastructure tailored to one object language, making the resulting definitions and lemmas difficult to reuse (see the review by Zackon et al. [57]). Rocq is widely used for mechanized meta-theory, but, to our knowledge, it has no general-purpose library for this bookkeeping. Existing Rocq developments, such as Emtst [13], Yalla [28], and the linear version of the Hybrid framework [30], have more limited scope. More general infrastructure has been developed for other proof assistants, notably Wood and Atkey's framework [55] in Agda and CARVe [57] in Beluga.

One solution to modelling substructural systems is to pair each assumption (viewed as a *resource*) with an annotation that records its availability (see for example [46, 57]). Rather than delete an assumption when it is consumed, one updates its annotation. In this representation, context splitting and consumption preserve a context's shape, making it a good fit for (well-scoped) de Bruijn syntax.

Existing annotation-based frameworks, however, commonly combine two concerns in one annotation. An assumption's structural *policy* specifies which structural rules it admits and remains fixed throughout a derivation. Its usage *state* records its current availability—whether, or how many times, it may still be used—and changes as a derivation allocates and consumes resources. Context operations update this state so that uses of the assumption obey its policy. When one annotation serves both roles, adding a policy or combining assumptions governed by different policies requires a richer annotation algebra, new interaction laws, and proofs of those laws. This complicates the reuse of meta-theoretic developments.

This paper presents Sumatra (**Su**bstructural **M**eta-theory with **A**djoin**t** **R**esource **A**nnotations), a Rocq library for substructural meta-theory. Each context entry carries the assumption itself together with two separately specified annotations:

- a *mode*, recording weakening and contraction permissions and belonging to a preorder governing dependencies between modes; and
- a *multiplicity*, recording current availability or quantity and updated as resources are split and consumed.

The mode preorder is inspired by adjoint logic [24, 42], where modes organize assumptions exhibiting different structural

CPP '27, January 11–12, 2027, Mexico City, Mexico

ACM ISBN 978-1-nnnn-nnnn-n/yy/mm
https://doi.org/nnnnnnn.nnnnnnn

behaviour and constrain their usage. However, users of Sumatra are not restricted to an adjoint object calculus. One can instantiate the mode and the multiplicity algebra appropriately to model a range of object languages.

Splitting a context in Sumatra consults both annotations: the multiplicity of a given entry may be divided between two premises only as its mode permits. Its resource and mode remain fixed. The two annotations can be varied independently. With the multiplicities fixed to a binary availability algebra, varying the mode system alone yields linear, affine, relevant, unrestricted, mixed, and arbitrary-mode developments. Varying the multiplicity algebra is also useful: a semiring of grades allows a split to divide quantities rather than mere availability.

Based on this design, Sumatra develops a theory of context operations, renaming, and simultaneous substitutions. In particular, its substitution judgment divides a target context among the images of a substitution and checks each image against a client-supplied condition. The library establishes the judgment's properties once, independently of any particular term syntax or typing judgment.

We make the following contributions:

- an annotation-based design that separates static structural policy from dynamic usage state;
- a Rocq library that develops, over that design, a general theory of context operations, renaming, and simultaneous substitution; and
- five mechanized case studies, including weak normalization for a dual-mode calculus, equivalence of adjoint natural deduction and sequent calculi, and preservation for a graded linear/affine calculus.

These case studies demonstrate the library's reuse across fixed, mixed, and arbitrary mode systems, binary and semiring-valued multiplicities, and various proof methods.

The artifact accompanying this paper is available at:

https://doi.org/10.5281/zenodo.22696854

2 Policy and state in contexts

In substructural calculi, contexts are collections of assumptions treated as *resources*. Each assumption is governed by a *structural policy*, namely the structural rules it admits. An assumption satisfying weakening (W) may be discarded unused; one satisfying contraction (C) may be duplicated across subderivations. The four classical policies are the four subsets of $\{\mathsf{W}, \mathsf{C}\}$: linear ($\emptyset$), affine ($\{\mathsf{W}\}$), relevant ($\{\mathsf{C}\}$), and unrestricted ($\{\mathsf{W}, \mathsf{C}\}$).[1]

An assumption's usage *state* is distinct from its policy: it records whether, or how many times, the assumption is available at a given stage of a derivation. Policy is *static* and fixed by the system's design. State is *dynamic*, evolving as a derivation allocates assumptions among premises and consumes them at leaves. In the simplest systems, the state is a single bit recording whether the assumption is currently available. Quantitative systems [4, 20, 38] instead record how many uses remain.

In a system with one fixed discipline, both dimensions can remain implicit. Linear context splitting, for example, can be inductively defined by:

$$\frac{}{\cdot = \cdot \,;\, \cdot} \qquad \frac{\Gamma = \Gamma_1 \,;\, \Gamma_2}{\Gamma, A = (\Gamma_1, A) \,;\, \Gamma_2} \qquad \frac{\Gamma = \Gamma_1 \,;\, \Gamma_2}{\Gamma, A = \Gamma_1 \,;\, (\Gamma_2, A)}$$

Each rule places the newly added A in exactly one premise context. Its membership therefore records both that it is governed by the underlying linear policy and that it is currently available to that premise.

When assumptions governed by different policies coexist, a common approach is to partition the context into policy-specific *zones*, each with its own discipline. The judgment $\Gamma \mid \Delta \vdash A$ of dual intuitionistic linear logic (DILL) [5], for example, separates unrestricted assumptions in Γ from linear assumptions in Δ; judgmental S4 [40] similarly separates valid from true assumptions. The mechanization cost is the duplication of infrastructure. If the zones use different namespaces, each needs its own renaming, substitution, lifting, composition, and normalization operations. Proofs connecting the zones must relate these parallel operations. With well-scoped de Bruijn syntax, every operation must also preserve the appropriate index space. Locally nameless syntax removes that particular obligation, but the splitting and substitution infrastructure must still be built and reasoned about for each zone.

Annotation-based representations take another route. Every context in a rule retains the same content while an annotation records whether and how it may be used. Splitting becomes a redistribution of annotations, and consuming an assumption becomes an annotation update rather than a deletion. Contexts thus retain their structure, which is a natural fit for well-scoped syntax. Existing annotation-based frameworks typically encode the policy and state in one annotation algebra. In CARVe [57], for example, the partial commutative monoid $\{0_L, 1_L\}$ with $1_L + 0_L = 0_L + 1_L = 1_L$ characterizes linear assumptions. To add unrestricted assumptions one extends the carrier with an element ω and decides how it interacts with the old elements. One may preserve partiality, defining only $\omega + \omega = \omega$, or make ω absorb every element. These choices encode different logics. Adding each further discipline requires extending the carrier, specifying interaction laws, and proving the resulting algebra well-behaved.

Sumatra retains the stable positions of an annotation-based representation but does not ask the multiplicity algebra alone to encode the discipline.

[1] We restrict attention to systems admitting exchange; ordered resources are discussed in subsection 8.1.

3 Design

Sumatra adopts an annotation-based representation, but separates the two dimensions of its bookkeeping. In addition to the resource itself, each context entry carries a *mode*, fixing its static policy and its relationship to other modes, and a *multiplicity*, recording its dynamic usage state.

A *mode system* consists of a set M, a preorder $\geq$, and a judgment $m \vDash S$ for $S \in \{\mathsf{W}, \mathsf{C}\}$. Structural permissions are monotone:

$$k \vDash S \quad \text{and} \quad m \geq k \quad \Longrightarrow \quad m \vDash S.$$

Modes generalize context zones, and the preorder determines when assumptions at one mode may support a conclusion at another. Because every assumption in the context carries its own mode, one context can mix assumptions from any number of disciplines. Users of Sumatra may choose to instantiate M concretely to model a specific object language, define a class of object languages by keeping M fully abstract, or constrain an abstract M to a subclass; our case studies do all three.

A *multiplicity algebra* has a zero 0 and a ternary, possibly partial addition $a +_d b \equiv c$, indexed by a proposition d that records whether contraction is permitted. At a context position whose mode is m, the index is instantiated with the proposition $m \vDash \mathsf{C}$. For each d, addition forms a positive partial commutative monoid: it is functional, associative, and commutative, 0 is a unit, and a sum can be 0 only when both summands are. Since structural permissions are determined by the mode, the multiplicity must record only the changing state. That state is binary availability in four case studies: 1 means that an assumption is available, while 0 means that it was never granted, has already been consumed, or was allocated to a different premise. In another case study, the state is a quantity.

We call an entry *spent* when its multiplicity is 0, and *live* otherwise.

3.1 Mode-indexed splitting

The context-split relation, $\Delta_1 \bowtie \Delta_2 \equiv \Delta$, is defined pointwise. Resources and modes agree across all three contexts, while the multiplicities at each position are related by addition "gated" by that position's contraction permission d. The standard carrier is binary availability $\mathsf{Avail} = \{0, 1\}$, with addition generated by

$$0 +_d 0 \equiv 0, \quad 0 +_d 1 \equiv 1, \quad 1 +_d 0 \equiv 1, \quad \frac{d}{1 +_d 1 \equiv 1}.$$

Read as a split, $a +_d b \equiv c$ says that an entry carrying multiplicity c in the conclusion may be passed to the premises carrying a and b. An available assumption may therefore occur in both premises exactly when its mode admits contraction. Read in the other direction, the same relation composes two compatible multiplicity assignments.

A mixed-policy example. Take two modes $\mathsf{U} \geq \mathsf{L}$, with U unrestricted and L linear. This is the usual DILL discipline within one scope. Consider the single context

$$\Delta_{\mathrm{ex}} = (\!| A, \mathsf{L}, 1 |\!),\ (\!| B, \mathsf{U}, 1 |\!),$$

where $(\!| T, m, a |\!)$ denotes an entry of resource T at mode m with multiplicity a. A split is constrained position by position by each entry's own mode. Since $\mathsf{L} \not\vDash \mathsf{C}$, A must go to exactly one premise; the other receives $(\!| A, \mathsf{L}, 0 |\!)$. Since $\mathsf{U} \vDash \mathsf{C}$, B may be duplicated. Thus both

$$((\!| A, \mathsf{L}, 1 |\!), (\!| B, \mathsf{U}, 1 |\!)) \bowtie ((\!| A, \mathsf{L}, 0 |\!), (\!| B, \mathsf{U}, 1 |\!)) \equiv \Delta_{\mathrm{ex}},$$
$$((\!| A, \mathsf{L}, 0 |\!), (\!| B, \mathsf{U}, 1 |\!)) \bowtie ((\!| A, \mathsf{L}, 1 |\!), (\!| B, \mathsf{U}, 1 |\!)) \equiv \Delta_{\mathrm{ex}}$$

are valid. Splitting is non-deterministic when read right-to-left, even though joining two fixed contexts has at most one result. Resources, modes, and positions remain aligned throughout.

3.2 Weakenability and independence

A substructural variable rule typically uses one assumption and discards the remainder of its context. This is allowed only when every remaining live entry admits weakening. Sumatra packages that side condition as

$$\mathsf{ctx\_weak}\,\Delta \iff \forall i.\ \mathsf{mult}(\Delta\, i) = 0 \ \vee\ \mathsf{mode}(\Delta\, i) \vDash \mathsf{W}.$$

That is, every entry is either already spent or sits at a mode admitting weakening. In the example above, $\mathsf{ctx\_weak}\,\Delta_{\mathrm{ex}}$ fails because the linear A is live. It holds after A is consumed (i.e., its multiplicity is set to 0), since the only remaining live entry is the unrestricted B.

Mode bounds express the *independence principle* of adjoint logic [24, 42]: a judgment at mode m may depend only on assumptions at modes at least m. Sumatra provides two forms. The predicate $\Delta \sqsupseteq m$ bounds every entry, whereas $\Delta \sqsupseteq^{+} m$ bounds only live entries. Our case studies use the latter form.

4 The Sumatra library

Sumatra is implemented in Rocq 9.1.1. Its core modules define modes, multiplicity algebras, contexts, splitting and renaming, simultaneous substitution, and proof automation. They are independent of object terms and judgments. The case studies use Autosubst 2 [48, 49] to generate syntax and syntactic substitution for languages with binders, but clients need not use its syntax generator. Binder-free languages may use the context infrastructure directly, and alternative representations of binding may supply their own renaming and substitution operations.

4.1 Modes and structural properties

The mode layer of section 3 is realized by a typeclass. In abbreviated form:

```
Variant Struct : Type := W | C.

Class ModeSystem (M : Type) := {
```

```
  mode_sat : Struct -> M -> Prop;
  mode_geq : M -> M -> Prop;
  mode_preorder : PreOrder mode_geq;
  mode_mono : forall m k S,
    mode_geq m k -> mode_sat S k -> mode_sat S m }.
```

We write $m \vDash S$ for `mode_sat S m`, and $m \geq k$ for the preorder. The familiar linear, affine, relevant, and unrestricted disciplines are induced by the `W/C` capabilities of a mode rather than represented as primitive cases.

The library supplies the four one-mode systems `Lin`, `Aff`, `Rel`, and `Unr`; the two-mode systems DILL, judgmental S4, and a linear/affine variant; and `Sub`, in which all four policies coexist. In `Sub`, Unr ≥ Aff ≥ Lin and Unr ≥ Rel ≥ Lin, while `Aff` and `Rel` are incomparable. Clients may also remain abstract. The dual-mode study (subsection 6.1) assumes only a strictly ordered pair of lower and upper modes, whereas the adjoint (subsection 6.2) and multiplicative–additive (subsection 6.4) developments accept an arbitrary `ModeSystem`.

4.2 Multiplicity algebras

The dynamic state is encoded by `MultAlgebra` over a carrier `A`. For every proposition `d`, the core fields make `mult_add d` a positive partial commutative monoid. As anticipated in section 3, at context position i the proposition `d` is instantiated by the contraction condition $\mathrm{mode}(\Delta\, i) \vDash C$:

```
Class MultAlgebra (A : Type) := {
  mzero : A;
  mult_add : Prop -> A -> A -> A -> Prop;
  mult_add_0_l : forall d a,
    mult_add d mzero a a;
  mult_add_comm : forall d a b c,
    mult_add d a b c -> mult_add d b a c;
  mult_add_func : forall d a b c c',
    mult_add d a b c -> mult_add d a b c' -> c = c';
  mult_add_assoc : forall d a b c e1 e2,
    mult_add d a b e1 -> mult_add d e1 c e2 ->
    exists f, mult_add d b c f /\ mult_add d a f e2;
  mult_add_zero_inv : forall d a b,
    mult_add d a b mzero -> a = mzero /\ b = mzero }.
```

Positivity (`mult_add_zero_inv`) rules out instances in which non-zero multiplicities cancel to zero, which would allow a split to discard resources. Addition is relational, following the usual presentation of partial resource composition in separation algebras [18].

Instantiating the multiplicity type class with `Avail` yields the equations of subsection 3.1; every case study except the graded one uses this algebra. A `TrivialAlgebra` on `unit` recovers a setting with no availability tracking, useful when only the mode structure matters. The graded study instantiates the addition interface with semiring addition and extends it with scalar multiplication.

Beyond the core laws, optional type classes state the additional algebraic assumptions required by particular theorems. `DecZero` adds decidable equality with zero, so that definitions and proofs can distinguish spent from live entries. `ContractionGatedAddition` connects multiplicity addition to the structural policy of the enclosing mode: splitting a multiplicity into two non-zero parts requires permission to contract, while contraction permits any multiplicity to be duplicated. The availability algebra satisfies this capability, whereas natural-number addition does not. `Cancellative` supports cancellation of context joins. Natural-number addition is cancellative, but cancellativity is not assumed of arbitrary grade semirings.

4.3 Functional contexts

An entry packages an object resource, a mode, and a multiplicity; a context of length n is a total function out of $\mathrm{fin}\, n$, the finite type of natural numbers strictly below n:

```
Record Entry (R M A : Type) :=
  mk_entry { res : R; mode : M; mult : A }.

Definition fctx (n : nat) (R M A : Type) :=
  fin n -> Entry R M A.
```

The implementation writes `(| r, m, a |)` for `mk_entry r m a`. The field `res` contains the object-language content of an assumption (an object type `ty` in the case studies); M is the mode type and A the carrier of a `MultAlgebra`. When R, M, and A are fixed, we will sometimes write `ctx n` for `fctx n R M A`.

The same indices inhabit object terms, so contexts and syntax share a scope by construction. We use Autosubst 2's implementation of finite types as n-fold iterations of `option`, since the case studies use its well-scoped de Bruijn syntax. Nothing in our generic library, however, mentions object terms or their syntactic substitution operations. Porting to another de Bruijn representation, such as the Rocq standard library's bounded integers, would require replacing this interface.

Although contexts are functions, they retain a list-like interface. Autosubst's stream-cons operation adds an element in head position. The empty context is the function out of `fin 0`. Consumption is a pointwise update that sets one multiplicity to zero while preserving the resource and mode at that position, as well as every other entry.

This representation has two visible consequences: context equality uses functional extensionality, and the generic operations are insensitive to the ordering of positions. The latter makes exchange easy but rules out ordered resources.

4.4 Context predicates

Four pointwise predicates recur in the case studies:

Predicate	Meaning
ctx_is_zero Δ	every multiplicity is zero
ctx_weak Δ	every live entry admits weakening
$\Delta \sqsupseteq m$	every entry's mode is at least m
$\Delta \sqsupseteq^{+} m$	every *live* entry's mode is at least m

The proposition ctx_is_zero Δ should be distinguished from the operation ctx_zero Δ. The former says that Δ is exhausted, whereas the latter preserves every resource and mode in Δ while setting its multiplicities to zero. The final two predicates express the two forms of independence introduced in subsection 3.2.

4.5 Context joins

The implementation's `ctx_join` relation lifts multiplicity addition pointwise:

```
Definition ctx_join (Δ1 Δ2 Δ : ctx n) : Prop :=
  forall i,
    res  (Δ1 i) = res  (Δ i) /\
    res  (Δ2 i) = res  (Δ i) /\
    mode (Δ1 i) = mode (Δ i) /\
    mode (Δ2 i) = mode (Δ i) /\
    mult_add (mode (Δ i) ⊨ C)
      (mult (Δ1 i)) (mult (Δ2 i)) (mult (Δ i)).
```

Because resources and modes are forced to agree, a split redistributes only multiplicities. The algebraic theory of mult_add lifts uniformly to contexts. Context composition is functional, commutative, and associative; a zeroed context is a unit; and interchange regroups two nested splits. Concretely, if $\Delta_1 \bowtie \Delta_2 \equiv \Delta$ and each summand is split again, then the four pieces can be regrouped by premise, and the regrouped contexts again join to Δ.

Binary joins suffice for rules with two premises, but a simultaneous substitution allocates one target-context share to each source variable. For a family $\delta : \mathsf{fin}\, m \to \mathsf{ctx}\ n$, the judgment ctx_join_family δ Δ states, informally, that $\delta(0) \bowtie \delta(1) \bowtie \cdots \bowtie \delta(m-1) \equiv \Delta$. This notation abbreviates iterated binary joins; the empty family combines to an exhausted context. The canonical variable family maps each i to the context obtained from Δ by retaining the entry at i and setting every other multiplicity to zero; these shares combine back to Δ.

4.6 Renamings and exchange

A renaming $\pi : \mathsf{fin}\, m \to \mathsf{fin}\, n$ acts by precomposition, $\Delta\langle\pi\rangle \coloneqq \Delta \circ \pi$. Since joins and the predicates above are pointwise, every renaming preserves them. Bijective renamings are packaged with an inverse and cancellation laws, and commute with update and consumption. For any object-language judgment equipped with a renaming lemma, exchange follows by instantiating that lemma with a bijection that swaps the chosen indices.

4.7 Proof automation

Substructural judgments, and meta-theoretic proofs about them, generate numerous side conditions on modes and multiplicities. Each is individually shallow, which makes them a natural target for automation. Sumatra's main tactics first saturate the proof context with consequences of splits and additions (e.g., a zero sum forces both summands to zero), and then invoke goal-directed solvers for joins, weakening, mode bounds, and related judgments. The umbrella tactic `sumatra_crush` combines both phases. If it cannot close the goal, it leaves the simplified remainder to the caller.

A second layer of tactics supports proofs of whole structural lemmas. Renaming, weakening by a discardable entry or context, and strengthening by removing a spent entry are properties that an object language must often establish before its main meta-theory. Their cases are routine enough to automate for the judgment forms currently supported. We supply a tactic per lemma, so the corresponding properties in the case studies reduce to a single line.

4.8 Trusted assumptions

The library assumes dependent functional extensionality to prove equalities between contexts. Some case-study proofs additionally use convenience tactics for dependent inversion; their generated proof terms may thus rely on uniqueness of identity proofs (UIP), equivalently Streicher's axiom K [50]. These dependencies are proof-engineering artifacts rather than requirements of the results. In representative cases, the appeal to UIP can be removed by reformulating the proof or strengthening the induction invariant. Neither the library nor the five case studies declares an axiom of its own, and their proofs contains no admitted results.

5 Simultaneous substitutions

Consider a simultaneous substitution $\sigma : \mathsf{fin}\, n \to \mathsf{tm}\ k$ from source scope n to target scope k. It assigns a target term $\sigma\, i$ to every source variable i, and the syntactic operation $e[\sigma]$ replaces each free occurrence of i in e by that image. In an ordinary structural calculus, a well-typed substitution from source context Δ to target context Ψ can be presented by:

$$\frac{\forall i : \mathsf{fin}\, n.\ \Psi \vdash \sigma\, i : \mathsf{res}(\Delta\, i)}{\Psi \vdash \sigma : \Delta}$$

Every image is checked under the same target context because its assumptions may be reused freely.

This rule is unsuitable for a substructural calculus. Checking every image under all of Ψ could let several images use the same linear assumption, for example. In addition, σ is a total map and therefore has an image even for a spent source variable. A well-typed term under Δ cannot use such a variable, so its substitution image need not be typed and must receive no part of Ψ.

Sumatra therefore divides Ψ into *shares*: contexts recording the portion assigned to each substitution image. Write

$\Psi \Vdash \sigma : \Delta$ when the images of σ collectively realize source context Δ using target context Ψ. The judgment is defined by the following rules:

$$\frac{\mathsf{ctx\_is\_zero}\,\Psi}{\Psi \Vdash \cdot : \cdot} \qquad \frac{\Psi' \Vdash \sigma : \Delta \qquad \Psi' \bowtie \Psi_e \equiv \Psi \qquad \mathsf{typedShare}(\Psi_e, t, e)}{\Psi \Vdash (\sigma, t) : (\Delta, e)} \tag{1}$$

The head condition has spent and live cases:

$$\frac{\mathsf{mult}(e) = 0 \quad \mathsf{ctx\_is\_zero}\,\Theta}{\mathsf{typedShare}(\Theta, t, e)}$$
$$\frac{\mathsf{mult}(e) \neq 0 \qquad \Theta \sqsupseteq^{+} \mathsf{mode}(e) \qquad \Theta \vdash t : \mathsf{res}(e)}{\mathsf{typedShare}(\Theta, t, e)} \tag{2}$$

A live source variable receives its own target share, in which its image is typed. The mode bound prevents an image at mode m from depending on live assumptions below m. A spent variable still has a syntactic image, but receives an exhausted share and imposes no typing obligation.

The corresponding substitution theorem retains the ordinary shape

$$\Delta \vdash e : T \quad \text{and} \quad \Psi \Vdash \sigma : \Delta \quad \Longrightarrow \quad \Psi \vdash e[\sigma] : T.$$

5.1 Abstracting the per-image condition

Typing is only one property that may be required of an image. Sumatra replaces the final premise of (2) by a parameter

$$P : \mathsf{ctx}\ k \to \mathsf{fin}\ n \to \mathsf{Prop}.$$

The proposition $P\ \Theta\ i$ may refer to σ, Δ, types, or other relations. For typed substitutions it is $\Theta \vdash \sigma\, i : \mathsf{res}(\Delta\, i)$. This abstraction is the judgment $\mathsf{subst\_alloc}\ \Psi\ \Delta\ P$.

The primary Rocq definition is a fixpoint, recursive on the size of the source context. For proofs that reason pointwise, Sumatra provides an equivalent indexed-family presentation:

$$\begin{aligned}&\mathsf{subst\_alloc}\ \Psi\ \Delta\ P\\ &\quad\Longleftrightarrow\quad \exists \psi.\ \mathsf{ctx\_join\_family}\ \psi\ \Psi\\ &\qquad\qquad \wedge \forall i.\ \mathsf{subst\_share}\ P\ i\ (\Delta\, i)\ (\psi\, i).\end{aligned}$$

Here `subst_share` is (2) with its typing premise replaced by $P\ (\psi\, i)\ i$. The fixpoint better supports inductive reasoning, while the family form exposes one target share per source index, matching the functional representation of σ.

5.2 Reusable proof principles

The theory of `subst_alloc` is developed parametrically in the per-image predicate P. Most of its results require only the core multiplicity-algebra laws and decidable zero. Three generic principles correspond to the principal cases of an object-language substitution proof.

Multiplicative rules. Suppose the source context of a substitution is split between two premises:

$$\Delta_1 \bowtie \Delta_2 \equiv \Delta \quad \text{and} \quad \mathsf{subst\_alloc}\ \Psi\ \Delta\ P.$$

The lemma `subst_split` constructs target contexts Ψ_1 and Ψ_2 such that

$$\Psi_1 \bowtie \Psi_2 \equiv \Psi \quad \text{and} \quad \mathsf{subst\_alloc}\ \Psi_j\ \Delta_j\ P \quad \text{for } j = 1, 2.$$

Its proof is pointwise in the join-family presentation of `subst_alloc`. If a source entry is spent, both of its split occurrences are spent and its allocated target share is zero. If it remains live in only one premise, that premise receives its whole target share and the other receives a zeroed copy. The case when the source entry remains live in both premises requires the optional capability `ContractionGatedAddition` (subsection 4.2). Accordingly, `subst_split` is not asserted for arbitrary grade semirings. The graded case study instead uses weighted substitutions, in which a source grade scales the context allocated to its image.

Binders. The lemma `subst_extend` extends both source and target contexts by a newly bound entry and lifts the existing images. The user must supply the two object-language facts that are specific to P: existing images continue to satisfy the predicate after extension by a spent entry, and the fresh variable satisfies the predicate under its singleton share.

Variables. The focusing lemma `subst_alloc_focus` isolates the share of one live source variable x. It states that whenever $\mathsf{subst\_alloc}\ \Psi\ \Delta\ P$ and $\mathsf{mult}(\Delta\, x) \neq 0$, there are contexts Ψ_x and Ψ_r with

$$\Psi_x \bowtie \Psi_r \equiv \Psi,\ \Psi_x \sqsupseteq^{+} \mathsf{mode}(\Delta\, x),\ P\ \Psi_x\ x, \text{ and}$$

$$\mathsf{subst\_alloc}\ \Psi_r\ (\mathsf{ctx\_consume}\ \Delta\ x)\ P.$$

Thus Ψ_x contains exactly the share allotted to the image of x and carries the condition imposed on that image, while Ψ_r is a target for the source context with x consumed. This is typically all that the variable case of a substitution proof needs: it uses the fact about the image under Ψ_x, then applies object-language weakening to restore the full target context.

The library also provides monotonicity in P, identity and composition, extension by a spent source entry, and transport of weakening (`subst_weak`) and live-mode bounds (`subst_live_modes_geq`) through `subst_alloc`. Autosubst separately supplies the syntactic equations for applying, lifting, and composing the substitution maps themselves.

6 Case studies

Five case studies, summarized in Table 1, exercise different parts of the design. This section presents the dual-mode, adjoint, and graded developments in detail; subsection 6.4 summarizes two additional studies.

6.1 A dual-mode λ-calculus

The first calculus has two distinct modes $\mathsf{Hi} \geq \mathsf{Lo}$ and types

$$T ::= \mathsf{base} \mid T \to T \mid \Box T.$$

Case study	Modes and multiplicities	Main mechanized results
Dual-mode substructural λ-calculus	Abstract ordered pair of modes; binary availability	Progress; preservation; two proofs of weak normalization
Adjoint natural deduction and sequent calculi	Arbitrary mode system; binary availability	Preservation; equivalence of presentations; precongruence via Howe's method
Linear/affine graded λ-calculus	Two modes; semiring-valued multiplicities	Progress; preservation; exact and bounded use for natural-number grades
Session-typed π-calculus	One linear mode; binary availability	Subject reduction; syntactic type safety
Multiplicative–additive λ-calculus	Arbitrary mode system; binary availability	Transport of typing along mode-system morphisms; type-preserving let elimination

Table 1. Summary of case studies.

Terms are those of the simply typed λ-calculus together with box e and let box e_1 in e_2. The typing judgment has no conclusion mode and uses binary availability.

In a single parametric development, the calculus generalizes two dual-context systems. With Lo linear and Hi unrestricted, $\to$ and $\Box$ form the $\multimap$/!-fragment of DILL. With both modes unrestricted but separated by the preorder, they are $\supset$ and $\Box$ of judgmental S4. It is, in effect, a substructural variant of Kavvos's dual-context modal calculus [27].

The calculus is parameterized by `DualModeSystem`, a subclass of `ModeSystem`. It fixes distinct modes Hi $\geq$ Lo and requires every mode to be one of these two. The multiplicity algebra is fixed to `Avail`. Monotonicity requires every structural rule admitted at Lo also to be admitted at Hi. Consequently, the same development covers four instances in which the modes have the same structural policy, including judgmental S4, and five in which they differ, including DILL. It rules out incomparable assignments such as a relevant lower mode and an affine upper mode: raising an assumption to the upper mode must never revoke a structural rule already permitted below.

Figure 1 gives the typing rules. They specialize to the corresponding typing rules of DILL and judgmental S4. The variable rule selects a live entry and requires the context that remains after consuming it to be weakenable. We deliberately omit a live-mode bound from this rule: adding one would depart from the judgmental S4 rules, which admit a true hypothesis in the presence of valid ones. As a consequence, this dual-mode calculus does not embed directly into the adjoint calculus of subsection 6.2.

Abstraction binds at Lo; application divides the context between its premises; and let box binds the unboxed value at Hi. The side conditions of BOX recover the two zoned rules without changing the context representation. The derivation of e uses only Δ_1, whose live assumptions must be at Hi; the unused share Δ_2 must be discardable. Under DILL, $\Delta_1 \sqsupseteq^{+}$ Hi forces its linear entries to be spent and ctx_weak Δ_2 forces any live entries of Δ_2 to be unrestricted. Under S4, the first premise excludes live lower-mode (true) assumptions, while the second is automatic because both modes admit weakening.

The artifact establishes progress and preservation for the case study's small-step reduction. The simultaneous substitution lemma for typing uses the properties of subsection 5.2: applications use `subst_split`, binders `subst_extend`, and the variable case `subst_alloc_focus`. In the BOX case, `subst_live_modes_geq` shows that the substituted context satisfies the same live-mode bound as the source. These are the only cases that differ from a standard proof using ordinary simultaneous substitution. Type preservation then follows by induction on the reduction derivation; its two beta cases invoke the substitution lemma. Multi-step type safety is the usual corollary.

Weak normalization. The central result is weak normalization for closed, well-typed terms. Our first proof uses a Tait-style predicate on closed terms:

$$\begin{aligned}\mathcal{R}_{\mathsf{base}}(e) &\iff \mathsf{Halts}(e),\\ \mathcal{R}_{T\to S}(e) &\iff \mathsf{Halts}(e) \wedge \forall u.\, \mathcal{R}_T(u) \Rightarrow \mathcal{R}_S(e\,u),\\ \mathcal{R}_{\Box T}(e) &\iff \mathsf{Halts}(e) \wedge \exists e'.\, e \longrightarrow^{*} \mathsf{box}\, e' \wedge \mathcal{R}_T(e').\end{aligned}$$

In the code, this is defined as a fixpoint by recursion over types. Backward closure is first proved for one reduction step, by induction on the type, and then lifted to multi-step reduction. This part follows the standard structural argument; the substructural content is concentrated in reducible substitutions,

$$\mathsf{RedSub}(\Delta, \sigma) \;\coloneqq\; \mathsf{subst\_alloc} \cdot \Delta \left(\lambda \_\, i.\, \mathcal{R}_{\mathsf{res}(\Delta i)}(\sigma\, i)\right).$$

The images are closed, so the allocation has an empty target context, and every share is also empty. The per-image condition therefore ignores its share argument, and $\mathsf{RedSub}(\Delta, \sigma)$ amounts to requiring a reducible image of every live entry of Δ.

$$\frac{\Delta\, x = (\!|\, T, m, 1 \,|\!) \qquad \mathsf{ctx\_weak}\,(\mathsf{ctx\_consume}\,\Delta\, x)}{\Delta \vdash x : T}\ \textsc{var}$$

$$\frac{\Delta, (\!|\, T, \mathsf{Lo}, 1 \,|\!) \vdash e : S}{\Delta \vdash \lambda e : T \to S}\ \textsc{lam} \qquad \frac{\Delta_1 \vdash e_1 : T \to S \qquad \Delta_2 \vdash e_2 : T \qquad \Delta_1 \bowtie \Delta_2 \equiv \Delta}{\Delta \vdash e_1\, e_2 : S}\ \textsc{app}$$

$$\frac{\Delta_1 \vdash e : T \qquad \Delta_1 \sqsupseteq^{+} \mathsf{Hi} \qquad \mathsf{ctx\_weak}\,\Delta_2 \qquad \Delta_1 \bowtie \Delta_2 \equiv \Delta}{\Delta \vdash \mathsf{box}\, e : \Box T}\ \textsc{box}$$

$$\frac{\Delta_1 \vdash e_1 : \Box S \qquad \Delta_2, (\!|\, S, \mathsf{Hi}, 1 \,|\!) \vdash e_2 : T \qquad \Delta_1 \bowtie \Delta_2 \equiv \Delta}{\Delta \vdash \mathsf{let\ box}\ e_1\ \mathsf{in}\ e_2 : T}\ \textsc{let-box}$$

Figure 1. Typing rules of the dual-mode calculus.

Two key closure properties of RedSub, `RedSub_split` and `RedSub_extend`, state that a reducible substitution serves both halves of a context join, and that it may be extended with a reducible term:

$$\begin{aligned}&\mathsf{RedSub}(\Delta, \sigma) \wedge \Delta_1 \bowtie \Delta_2 \equiv \Delta\\ &\Longrightarrow \mathsf{RedSub}(\Delta_1, \sigma) \wedge \mathsf{RedSub}(\Delta_2, \sigma),\end{aligned}$$

$$\begin{aligned}&\mathsf{RedSub}(\Delta, \sigma) \wedge \mathcal{R}_T(u)\\ &\Longrightarrow \mathsf{RedSub}(\Delta, (\!|\, T, m, a \,|\!),\ \sigma, u).\end{aligned}$$

Both follow from the generic lemmas of subsection 5.2.

The fundamental lemma,

$$\Delta \vdash e : T \ \wedge\ \mathsf{RedSub}(\Delta, \sigma) \Longrightarrow \mathcal{R}_T(e[\sigma]),$$

is proved by induction on the typing derivation. We outline the proof in detail to illustrate the use of the reusable proof principles from subsection 5.2.

Variables. Inversion of VAR gives $\Delta\, x = (\!|\, T, m, 1 \,|\!)$, so x is live. Applying `subst_alloc_focus` then gives $\mathcal{R}_{\mathsf{res}(\Delta\, x)}(\sigma\, x)$, that is, $\mathcal{R}_T(\sigma\, x)$. Since $x[\sigma] = \sigma\, x$, this is the goal.

Abstraction. We must show $\mathcal{R}_{T\to S}(\lambda\, e[\sigma^{\Uparrow}])$, writing $\sigma^{\Uparrow}$ for Autosubst's lifting of σ under a binder. An abstraction is a value, so it halts. Given u with $\mathcal{R}_T(u)$, one step of β-reduction gives $(\lambda\, e[\sigma^{\Uparrow}])\, u \longrightarrow e[\sigma, u]$, so backward closure reduces the goal to $\mathcal{R}_S(e[\sigma, u])$. That is the induction hypothesis for the premise $\Delta, (\!|\, T, \mathsf{Lo}, 1 \,|\!) \vdash e : S$, applied to the extension of RedSub supplied by `RedSub_extend`.

Application. Inversion gives $\Delta_1 \vdash e_1 : T \to S$, $\Delta_2 \vdash e_2 : T$, and $\Delta_1 \bowtie \Delta_2 \equiv \Delta$. `RedSub_split` and the two induction hypotheses give the chain

$$\begin{aligned}\mathsf{RedSub}(\Delta, \sigma) &\Longrightarrow \mathsf{RedSub}(\Delta_1, \sigma) \wedge \mathsf{RedSub}(\Delta_2, \sigma)\\ &\Longrightarrow \mathcal{R}_{T\to S}(e_1[\sigma]) \wedge \mathcal{R}_T(e_2[\sigma])\\ &\Longrightarrow \mathcal{R}_S(e_1[\sigma]\, e_2[\sigma]) \Longrightarrow \mathcal{R}_S((e_1\, e_2)[\sigma]).\end{aligned}$$

Box. Here $\Delta_1 \vdash e : T$ with $\Delta_1 \bowtie \Delta_2 \equiv \Delta$, so only the left half of the split is used; $(\mathsf{box}\, e)[\sigma] = \mathsf{box}\,(e[\sigma])$ is already a value, and the induction hypothesis gives $\mathcal{R}_T(e[\sigma])$.

Let box. Suppose the scrutinee has type $\Box T$ and the continuation has result type S. `RedSub_split` separates their source contexts. The scrutinee's induction hypothesis yields $e_1[\sigma] \longrightarrow^{*} \mathsf{box}\, v$ with $\mathcal{R}_T(v)$, and so

$$\mathsf{let\ box}\ e_1[\sigma]\ \mathsf{in}\ e_2[\sigma^{\Uparrow}] \longrightarrow^{*} e_2[\sigma, v].$$

Extension adds v to the substitution for the continuation; the second induction hypothesis gives $\mathcal{R}_S(e_2[\sigma, v])$, and multi-step backward closure transports it back along that reduction.

Instantiating the fundamental lemma with the empty context and the empty substitution yields weak normalization.

A second proof. The artifact also adapts Stark's open-term, Kripke-style approach [48]. It separates a type-indexed value predicate L_T from its evaluation closure E_T. At base type L_T holds of any value, and at function and box types it requires the appropriate canonical form with recursively related components. The closure $E_T(e)$ holds when e reduces to a term satisfying L_T. The proof uses the predicate $G\,\Delta\,\sigma \coloneqq \forall i.\, E_{\mathsf{res}(\Delta\, i)}(\sigma\, i)$. Unlike RedSub, this condition constrains every image, including the image of a spent source entry. Since G ignores multiplicities, the same σ satisfies $G\,\Delta_1\,\sigma$ and $G\,\Delta_2\,\sigma$ whenever $\Delta_1 \bowtie \Delta_2 \equiv \Delta$ and $G\,\Delta\,\sigma$; no `subst_alloc` argument is needed.

6.2 Adjoint natural deduction and sequent calculi

The second calculus follows the implicit adjoint natural-deduction calculus of Jang et al. [24]. It is parametric in an arbitrary mode system, and its judgment $\Delta \vdash_{nd} e : (T, m)$ records the conclusion mode. Types are

$$T ::= \mathbf{1} \mid T \multimap T \mid \uparrow_k^m T \mid \downarrow_m^k T.$$

The term language combines the ordinary unit and function forms with introduction and elimination for each shift:

$$\begin{aligned}e ::= {} & x \mid \lambda x.\, e \mid e\, e \mid \langle\rangle \mid \mathsf{let}\ \langle\rangle = e\ \mathsf{in}\ e\\ & \mid \mathsf{susp}\, e \mid \mathsf{force}\, e \mid \mathsf{down}\, e \mid \mathsf{let\text{-}down}\, e\, e.\end{aligned}$$

The shifts internalize movement between modes. For $m \geq k$, $\uparrow_k^m T$ packages a k-mode type at m; for $k \geq m$, $\downarrow_m^k T$ packages a k-mode type at m. We collapse Jang et al.'s checking and synthesis judgments and state their independence requirements as explicit side conditions. Representative rules appear in Figure 2.

$$\frac{\Delta\, x = (\!|\, T, m, 1 \,|\!) \quad \mathsf{ctx\_weak}(\mathsf{ctx\_consume}\,\Delta\, x) \quad \Delta \sqsupseteq^{+} m}{\Delta \vdash_{nd} x : (T, m)}\ \textsc{var}$$

$$\frac{m \geq k \quad \Delta \sqsupseteq^{+} m \quad \Delta \vdash_{nd} e : (T, k)}{\Delta \vdash_{nd} \mathsf{susp}\, e : (\uparrow_k^m T, m)}\ \textsc{up-i}$$

$$\frac{m \geq k \qquad \Delta_1 \vdash_{nd} s : (\uparrow_k^m T, m) \qquad \Delta_2 \sqsupseteq^{+} k \qquad \mathsf{ctx\_weak}\,\Delta_2 \qquad \Delta_1 \bowtie \Delta_2 \equiv \Delta}{\Delta \vdash_{nd} \mathsf{force}\, s : (T, k)}\ \textsc{up-e}$$

Figure 2. Selected adjoint natural-deduction rules.

Equivalence with the sequent calculus. The sequent judgment $\Delta \vdash_s (T, m)$ has identity and cut, together with left and right rules for the four connectives. Two of its rules will be useful below. Cut is standard:

$$\frac{\Delta_1 \vdash_s (T, m) \quad \Delta_2, (\!|\, T, m, 1 \,|\!) \vdash_s (S, r) \quad \Delta_1 \bowtie \Delta_2 \equiv \Delta}{\Delta \vdash_s (S, r)}\ \textsc{cut}$$

The left rule for $\uparrow$ consumes a selected assumption $\uparrow_k^m S$ and makes S available to its continuation at the lower mode k:

$$\frac{\Delta\, x = (\!|\, \uparrow_k^m S, m, 1 \,|\!) \qquad m \geq k \qquad \mathsf{ctx\_consume}\,\Delta\, x, (\!|\, S, k, 1 \,|\!) \vdash_s (T, r)}{\Delta \vdash_s (T, r)}\ \textsc{up-l}$$

We prove that the two systems coincide:

$$(\exists e.\ \Delta \vdash_{nd} e : (T, m)) \iff \Delta \vdash_s (T, m).$$

The natural-deduction-to-sequent direction (soundness) is a direct induction on the typing derivation. Introductions become right rules. Eliminations use cut against a continuation derived with the corresponding left rule (e.g., for the displayed UP-E rule we obtain $\Delta_2, (\!|\, \uparrow_k^m T, m, 1 \,|\!) \vdash_s (T, k)$ from UP-L and identity).

For the converse, we strengthen the statement by transforming a sequent derivation after an arbitrary well-typed substitution:

$$\Delta \vdash_s (T, r) \ \wedge\ \Psi \Vdash \sigma : \Delta \implies \exists e.\ \Psi \vdash_{nd} e : (T, r).$$

The proof is by induction on the sequent derivation. The two rules displayed above illustrate how the substitution properties connect the calculi.

Cut. Suppose the final rule cuts $\Delta_1 \vdash_s (A, m)$ against $\Delta_2, (\!|\, A, m, 1 \,|\!) \vdash_s (C, r)$, where $\Delta_1 \bowtie \Delta_2 \equiv \Delta$. Then `subst_split` divides the target into $\Psi_1 \bowtie \Psi_2 \equiv \Psi$ and supplies a well-typed substitution for each premise. The first induction hypothesis produces $\Psi_1 \vdash_{nd} e_1 : (A, m)$. After extending the second substitution beneath its fresh assumption, the second induction hypothesis produces a continuation typed under $\Psi_2, (\!|\, A, m, 1 \,|\!)$. A substitution lemma replaces that assumption by e_1 and recombines Ψ_1 and Ψ_2.

Upshift (left). For UP-L, focusing similarly gives $\Psi_x \bowtie \Psi_r \equiv \Psi$, a derivation $\Psi_x \vdash_{nd} \sigma\, x : (\uparrow_k^m T, m)$, and a well-typed substitution from $\mathsf{ctx\_consume}\,\Delta\, x$ into Ψ_r. Extending the latter with a fresh (T, k) assumption lets the induction hypothesis transform the rule's continuation premise. A derived natural-deduction rule types $\mathsf{force}(\sigma\, x)$ under Ψ_x; the substitution lemma substitutes this term for the continuation's fresh variable and recombines Ψ_x and Ψ_r.

The remaining left rules use the same pattern. The $\multimap$-L case also uses `subst_split` to divide the consumed remainder between its argument and continuation premises. Identity uses focusing together with weakening of the unused target remainder. The right rules are more direct: the induction hypothesis supplies their natural-deduction premises; $\multimap$-R extends the substitution under its binder, $\downarrow$-R splits it along the rule's context join, and the generic substitution lemmas establish the required side conditions.

Completeness follows by taking $\Psi = \Delta$ and σ to be the identity substitution; a lemma proves that this substitution is well typed from any context to itself. Together with the forward translation, this yields the stated equivalence.

Other meta-theory. The artifact also proves preservation for call-by-name evaluation and the independence theorem $\Delta \vdash_{nd} e : (T, m) \Rightarrow \Delta \sqsupseteq^{+} m$.

We also adapt the Beluga development of Momigliano et al. [34] to prove by Howe's method [22] that open applicative similarity is a precongruence for this calculus. Closed similarity is coinductive and indexed by type and mode; open similarity lifts it along well-typed closing substitutions, and the Howe closure is its compatible closure. The central substitutivity lemma again instantiates `subst_alloc`, now with a predicate requiring the two images to be Howe-related in their common share. Generic split and extension reorganize both substitutions simultaneously. This step uses the facts that Howe-relatedness enjoys weakening and that a variable is Howe-related to itself. Coinduction proves that the Howe closure is a simulation and hence coincides with open similarity. Our development gives an answer to Crole's question, "How linear is Howe?" [16]: aside from the substitutivity lemma, the proof largely retains its usual shape.

6.3 Linear and affine graded use

The previous case studies varied the mode system while fixing the multiplicity algebra. The third study replaces binary availability with quantitative state.

Grades range over a `LinearGradeAlgebra`. Following Choudhury et al. [15], this is a semiring with $1 \neq 0$ in which 1 is additively indecomposable ($r + s = 1$ forces one summand to be 1 and the other 0) and multiplicatively prime ($rs = 1$ forces $r = s = 1$). These conditions allow grade 1 to represent an ordinary single use.

The two modes are $\mathsf{A} \geq \mathsf{L}$. Mode A admits weakening; L does not; neither grants contraction. Semiring addition, however, is total at either mode because the multiplicity itself counts uses. The relevant policy difference is therefore weakening: a grade q must be fully accounted for at L, while

unused grade may be discarded at A. Thus, in the natural-number instance, an assumption of grade q represents exact usage ("exactly q times") at mode L, but bounded usage ("at most q times") at mode A.

The typing judgment $\Delta \vdash a : A$ carries no conclusion mode. Instead, mode annotations appear on $\multimap$ and $\Box$:

$$S, T ::= \mathbf{1} \mid S \multimap_k T \mid \Box_{q,k} T.$$

A function's argument enters the context at grade 1; a graded arrow is the derived form $S \to_{q,k} T \coloneqq \Box_{q,k} S \multimap_k T$.

The variable rule consumes one unit of grade at the selected position. Any residual context must be discardable:

$$\frac{\mathsf{res}(\Delta\, x) = T \qquad \mathsf{ctx\_weak}\ \Delta_r \qquad \mathsf{ctx\_unit\_singleton}\ \Delta\ x \bowtie \Delta_r \equiv \Delta}{\Delta \vdash x : T}\ \textsc{var}$$

Here ctx_unit_singleton $\Delta\ x$ assigns grade 1 to x and 0 everywhere else in Δ. Box introduction scales the context: if $\Delta \vdash e : T$, then $q \bullet \Delta \vdash \mathsf{box}_q\ e : \Box_{q,k} T$. Eliminating $\Box_{q,k} T$ makes T available at grade q and mode k.

The mechanization proves that typing is preserved by bijective renaming, derives exchange and weakening, and establishes progress and preservation. It also validates the intended reading for natural-number grades. Let uses $t\ i$ count free occurrences of i in t, multiplying the count by q under box_q. Then

$$\Delta \vdash t : T \implies \forall i.\ \begin{cases} \mathsf{uses}\ t\ i = \mathsf{mult}(\Delta\, i), & \mathsf{mode}(\Delta\, i) = \mathsf{L}, \\ \mathsf{uses}\ t\ i \le \mathsf{mult}(\Delta\, i), & \mathsf{mode}(\Delta\, i) = \mathsf{A}. \end{cases}$$

The Rocq proof is an induction on typing whose arithmetic obligations are discharged by Micromega's [8] `lia` and `nia` tactics. Over an arbitrary semiring, the analogous affine statement would use the additive preorder $u \preceq q \coloneqq \exists r.\, u + r = q$.

Preservation uses *weighted simultaneous substitution*. If a source variable i has grade q_i and its image is typed under Ψ_i, that image contributes the scaled context $q_i \bullet \Psi_i$ to the target. We specialize Sumatra's ordinary allocation judgment with scaled image contexts; semiring distributivity ensures that these allocations are preserved when grades are split.

6.4 Further case studies

A session-typed π-calculus. Unlike the mode-parametric developments above, this calculus is fixed to a single linear mode. We mechanize a variant of the linearity challenge from the Concurrent Calculi Formalisation Benchmark [12], based on the finite name-passing session calculus of Gay and Vasconcelos [19]. Session types distinguish terminated, receiving, and sending endpoints; the process syntax has corresponding wait, close, input, and output prefixes, together with parallel composition and restriction. Typing assigns dual types to the names bound by $(\nu xy)P$ and splits the context at parallel composition. Because there is no separate class of base values, every transmitted name is itself a linear resource.

The artifact proves the Benchmark's subject-reduction and syntactic type-safety theorems. In the principal communication case, a renaming redirects the receiver's bound placeholder to the transmitted name; Sumatra's update, bijection, and join laws discharge the context bookkeeping. Type safety states that, modulo structural congruence, whenever two parallel processes are prefixed at the endpoints bound by one restriction, those prefixes are complementary: wait/close or input/output. The theorem allows an arbitrary surrounding context, slightly strengthening the Benchmark statement, because the property inspects only those locally bound endpoints.

A multiplicative–additive λ-calculus. This study covers the full multiplicative–additive propositional fragment over an arbitrary `ModeSystem`. Every connective remains at a single mode, so the conclusion mode is fixed while the context may contain assumptions at higher modes. Because modes do not occur in the term syntax, the same term is interpreted under different structural disciplines merely by choosing the mode system. Instantiating with `Sub` (subsection 4.1) places all four standard disciplines in one calculus.

Its main theorem concerns a map F between mode systems that preserves both the preorder and the permissions to weaken and contract. Applying F to every mode in a context preserves typing:

$$\Delta \vdash e : (T, m) \quad \Longrightarrow \quad \mathsf{map\_ctx}\ F\ \Delta \vdash e : (T, F(m)).$$

Thus each one-mode discipline embeds into `Sub`, while mapping every mode to the unrestricted mode erases all structural restrictions. A second result extends the syntax with let $x = e_1$ in e_2, translates that form recursively to $(\lambda x.\, e_2)\, e_1$, and proves that the translation preserves typing.

7 Related work

The idea that logical consequence and provability depend on how information is *used* predates linear logic and runs through the history of logic, philosophy, and mathematics (see Restall's [44] historical overview). Sumatra lies at the intersection of three lines of work: proof systems that organize assumptions by structural discipline, algebraic accounts of quantitative resource use, and mechanized frameworks for meta-theory.

7.1 Organizing structural policy

Zoned and mixed systems. In Girard's original formulation of linear logic, exponentials distinguish assumptions that admit weakening and contraction from those that do not. Constructive calculi often express this distinction structurally by separating assumptions with different structural behaviour into distinct *zones*. The most influential example is DILL [5], which separates unrestricted from linear assumptions. Related approaches include the judgmental reconstruction of S4 [40] and contextual type theory [35].

While conceptually clean, this approach scales poorly as further structural regimes are added: strict [33], ordered [41], or other resources require additional zones and increasingly elaborate judgments.

Benton's LNL [6] instead treats the linear and non-linear fragments as separate worlds related by a symmetric monoidal adjunction. Here structural regimes are organized explicitly and connected by modalities. This view led to the notion of adjoint logic, which we discuss next.

Adjoint logic. Pruiksma et al.'s adjoint logic [42] organizes (intuitionistic) logics by a preorder of modes, each with monotone weakening and contraction permissions. Each mode has its own copy of the connectives, and adjoint shifts connect comparable modes. The preorder enforces *independence*: a judgment at mode m may depend only on assumptions at modes above m. They prove identity expansion, cut elimination, and focusing, recovering, among others, LNL, judgmental S4, and intuitionistic subexponential linear logic [42]. Jang et al. relate a natural-deduction presentation to the sequent calculus and develop its computational interpretation [24].

Licata et al.'s [29] mode theories are even more general. Terms of an abstract mode theory constrain context use, with instances including non-associative and ordered structure, n-linear variables, and bunched implications. Identity and cut are proved parametrically in the mode theory.

Sumatra operates at a narrower level. Its `ModeSystem` retains only the preorder and structural permissions. Its context theory assumes exchange and associativity and provides neither mode-indexed connectives, shifts, nor a generic proof theory. Users decide how modes enter their calculi. The adjoint case study reconstructs object-level shifts and independence, while other clients reuse the context operations without adopting adjoint logic.

Recent work has combined modes and grades. Vollmer et al. use modalities between graded and graded-linear fragments to factor a graded exponential [51]. Grass gives each mode a grade algebra and grade-sensitive structural permissions [21]. Sumatra's graded case study shows that it can combine one semiring with various modes.

The Snax compiler [39] illustrates the practical use of mode-indexed structural disciplines. It supports mixed substructural programming based on adjoint types. Like Sumatra, Snax organizes structural regimes by modes, but applies this structure to programming and compilation rather than mechanized meta-theory.

Subexponentials. Sumatra's mode interface has a close precedent in subexponential linear logic. Inspired by Danos et al.'s study of non-canonical exponentials [17], Sell [36] uses signatures $\langle I, \preceq, W, C\rangle$, consisting of labels indexing families of exponentials, a preorder, and upward-closed sets of labels admitting weakening and contraction. Labels can be read as named zones, while the preorder constrains promotion and dependencies between zones.

Subexponentials, combined with focusing, have also been used for specifying proof systems [37], including intuitionistic and modal calculi. Tools such as Tatu derive meta-properties from meta-logical criteria on such specifications; subsequent work includes more direct meta-proof analyzers like Sequoia [43]. Recent extensions cover non-commutative systems [26] and multi-modal logical frameworks [56].

Both Sell and Sumatra order structural policies to determine which parts of a context may support a judgment or rule application. Sell is a proof system and internalizes labels and their preorder through indexed exponentials; Sumatra exposes analogous information at the meta-level through reusable mode predicates, together with a notion of dynamic state. Since intuitionistic subexponential linear logic embeds into adjoint logic [42], we expect this embedding to carry over to Sumatra.

7.2 Representing dynamic state

Girard et al.'s bounded linear logic [20] replaces the ordinary exponential with indexed modalities $!_x A$, tracking the *degree* of reuse. This is the ancestor of modern quantitative and graded modal type theories [1, 4, 15], where those indices have been refined using (ordered) commutative monoids, quantales, and semirings. These studies share our use of algebraic annotations but pursue different ends, from defining a linear dependent function space [32] to designing resource-aware functional languages such as Linear Haskell [7], Granule [38], and Idris 2 [9].

Another close relative to our approach is the body of work on separation algebras [11, 25]. Our binary multiplicity algebra derives from the partial commutative monoid of Dockins et al. [18], as implemented in the Mechanized Semantic Library [3]. Separation algebras usually model heaps and other compositional resources; we apply our algebra to assumption annotations, and pair it with an orthogonal mode preorder carrying structural policy. We also generalize the algebra to support semiring-valued multiplicities.

7.3 Mechanized infrastructure

We do not survey the full range of approaches to substructural context management; for a partial overview, we refer the reader to [57].

Wood and Atkey's Agda framework [55], developed further in [54], shares Sumatra's generic aims. It generates intrinsically typed, usage-annotated syntax from descriptions of typing rules: terms are indexed by a typing context, a usage context, and their type. Premises are composed using bunched operators expressing shared usage, additive combination of usage, and scaling. Their notion of environment equips each source variable with a target object together with a linear map accounting for the resources used by these images; generic traversals then yield simultaneous renaming

and substitution. By construction, the framework provides strong generic support for semiring-annotated syntax and its traversals. Sumatra instead leaves syntax and typing judgments to the user and factors resource management into a reusable context interface, which can also be used by judgments and proof techniques beyond typing. For instance, its substitution-allocation judgment is independent of both the representation of terms and the predicate imposed on each substitution image.

Sumatra's more direct precursor is CARVe [57], a Beluga library for annotation-based substructural contexts. Unlike Sumatra, CARVe draws its annotations from a single resource algebra that encodes both structural policy and state. Combining several structural regimes within one context, or extending an existing combination, therefore requires enlarging the algebra, defining the interactions between its elements, and re-establishing its laws (section 2). Although CARVe uses Beluga's simultaneous substitutions to express relations between contexts, it provides no analogue of Sumatra's predicate-parameterized substitution-allocation judgment. The proof assistants also encourage different styles. Beluga's higher-order abstract syntax (HOAS) provides concise representations of binding, while Sumatra works with first-order syntax in Rocq, optionally generated by Autosubst. This makes syntax-directed transformations such as the let-elimination of subsection 6.4 ordinary recursive functions. The corresponding transformation would be less direct in Beluga due to its two-level architecture.

In Rocq, other libraries are available for particular substructural formalisms. Yalla [28] formalizes parameterized propositional linear logic, providing customized instances together with results such as cut admissibility and completeness of focusing. EMTST [13] targets session-type metatheory, using locally nameless binders and finite-map contexts with a partial split operation, undefined whenever an entry would be duplicated. Both are tailored to specific application domains and to essentially linear resource disciplines.

8 Conclusions and future work

We have presented Sumatra, a Rocq library that separates an assumption's static structural policy from its dynamic usage state. These components descend, respectively, from adjoint and subexponential signatures and from algebraic annotation techniques. Sumatra treats them as orthogonal interfaces and develops, over their combination, a reusable theory of contexts and substitution.

The case studies support four main conclusions.

First, retaining variable positions integrates well with Autosubst. Splitting and consumption change annotations, not scopes, so they do not require terms to be reindexed. Modes can mix structural policies without multiplying syntactic substitution operations.

Second, the policy/state separation supports broad reuse of the library. Binary availability supports linear, affine, relevant, and mixed-mode calculi. Semiring-valued multiplicities support quantitative use without changing the context interface.

Third, syntactic substitution and resource allocation are worth separating. Autosubst supplies the syntactic application of a substitution, while `subst_alloc` divides a target context among its images. Typed, reducible, and Howe-related substitutions vary only the condition on each image, and therefore share their structural theory. Quantitative substitutions require a grade-sensitive split lemma but reuse the rest of this infrastructure.

Finally, the object-level obligations left by this design are numerous but individually shallow; targeted solvers and `sumatra_crush` discharge them without obscuring the object-language reasoning.

8.1 Future work

Ordered and heterogeneous resources. Because Sumatra's contexts are functions $\mathsf{fin}\, n \to \mathsf{Entry}$ and its generic operations are pointwise, ordered or non-commutative resources [26, 45] would require a position-sensitive join or another context representation. Likewise, coexisting grade algebras [21] would require an indexed family of multiplicity carriers and changes to join, scaling, and substitution.

Further object-language meta-theory. Future developments include cut elimination for the adjoint sequent calculus and a possibility modality generalizing $\Diamond$ and linear logic's ? [14] for the dual-mode calculus.

All current studies use extrinsic typing. An intrinsic calculus could instead index its terms by both a Sumatra context and a type, with constructors carrying joins, mode bounds, and weakening conditions. This would test the library's generality and permit a more direct comparison with Wood and Atkey's framework.

Our weak-normalization and Howe's-method developments both extend to the substructural setting with few changes. How other logical-relations techniques adapt remains open. A promising benchmark is substructural parametricity [2].

AI usage statement

The authors used a large language model (GPT 5.6) for copy-editing, literature search, and assistance in developing and debugging selected Ltac tactics. References suggested by the model were retrieved, read, and assessed by the authors before being cited.

References

[1] Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. 2023. A graded modal dependent type theory with a universe and erasure, formalized. In *Proc. ICFP '23*. 920–954. https://doi.org/10.1145/3607862

[2] C. B. Aberlé, Karl Crary, Chris Martens, and Frank Pfenning. 2025. Substructural parametricity. In *Proc. FSCD '25 (LIPIcs, Vol. 337)*, Maribel Fernández (Ed.). 4:1–4:21. https://doi.org/10.4230/LIPIcs.FSCD.2025.4

[3] Andrew W. Appel, Robert Dockins, and Aquinas Hobor. 2009. Mechanized Semantic Library. https://vst.cs.princeton.edu/msl/

[4] Robert Atkey. 2018. Syntax and semantics of quantitative type theory. In *Proc. LICS '18*. 56–65. https://doi.org/10.1145/3209108.3209189

[5] Andrew Barber. 1996. *Dual intuitionistic linear logic.* Technical report ECS-LFCS-96-347. University of Edinburgh. https://www.lfcs.inf.ed.ac.uk/reports/96/ECS-LFCS-96-347

[6] P. N. Benton. 1994. A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In *Proc. CSL '94*. 121–135. https://doi.org/10.5555/647844.736565

[7] Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. 2018. Linear Haskell: Practical linearity in a higher-order polymorphic language. *Proc. ACM Program. Lang.* 2, POPL (2018), 5:1–5:29. https://doi.org/10.1145/3158093

[8] Frédéric Besson and Evgeny Makarov. 2026. Micromega. GitHub. https://github.com/rocq-community/micromega-plugin/tree/v1.1.1 (version 1.1.1).

[9] Edwin C. Brady. 2021. Idris 2: Quantitative type theory in practice. In *Proc. ECOOP '21 (LIPIcs, Vol. 194)*, Anders Møller and Manu Sridharan (Eds.). 9:1–9:26. https://doi.org/10.4230/LIPIcs.ECOOP.2021.9

[10] Luís Caires and Frank Pfenning. 2010. Session types as intuitionistic linear propositions. In *Proc. CONCUR '10 (Lect. Notes Comput. Sci., Vol. 6269)*, Paul Gastin and François Laroussinie (Eds.). 222–236. https://doi.org/10.1007/978-3-642-15375-4_16

[11] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. 2007. Local action and abstract separation logic. In *Proc. LICS '07*. 366–378. https://doi.org/10.1109/LICS.2007.30

[12] Marco Carbone, David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, Frederik Krogsdal Jacobsen, Alberto Momigliano, Luca Padovani, Alceste Scalas, Dawit Tirore, Martin Vassor, Nobuko Yoshida, and Daniel Zackon. 2024. The concurrent calculi formalisation benchmark. In *Proc. COORDINATION '24 (Lect. Notes Comput. Sci., Vol. 14676)*, Ilaria Castellani and Francesco Tiezzi (Eds.). 149–158. https://doi.org/10.1007/978-3-031-62697-5_9

[13] David Castro, Francisco Ferreira, and Nobuko Yoshida. 2020. EMTST: Engineering the meta-theory of session types. In *Proc. TACAS '20 (Lect. Notes Comput. Sci., Vol. 12079)*, Armin Biere and David Parker (Eds.). 278–285. https://doi.org/10.1007/978-3-030-45237-7_17

[14] Bor-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. 2003. *A judgmental analysis of linear logic.* Technical Report CMU-CS-03-131R. Department of Computer Science, Carnegie Mellon University, Pittsburgh, Penn.

[15] Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. 2021. A graded dependent type system with a usage-aware semantics. *Proc. ACM Program. Lang.* 5, POPL (2021), 50:1–50:32. https://doi.org/10.1145/3434331

[16] Roy L. Crole. 1996. How linear is Howe?. In *Advances in Theory and Formal Methods*, G. McCusker, A. Edalat, and S. Jourdan (Eds.). Imperial College Press, 60–72.

[17] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. 1993. The structure of exponentials: Uncovering the dynamics of linear logic proofs. In *Proc. KGC '93 (Lect. Notes Comput. Sci., Vol. 713)*, Georg Gottlob, Alexander Leitsch, and Daniele Mundici (Eds.). 159–171. https://doi.org/10.1007/BFb0022564

[18] Robert Dockins, Aquinas Hobor, and Andrew W. Appel. 2009. A fresh look at separation algebras and share accounting. In *Proc. APLAS '09 (Lect. Notes Comput. Sci., Vol. 5904)*, Zhenjiang Hu (Ed.). 161–177. https://doi.org/10.1007/978-3-642-10672-9_13

[19] Simon J. Gay and Vasco T. Vasconcelos. 2025. *Session Types.* Cambridge University Press, Cambridge. https://doi.org/10.1017/9781009000062

[20] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. 1992. Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.* 97, 1 (1992), 1–66. https://doi.org/10.1016/0304-3975(92)90386-T

[21] Peter Hanukaev and Harley Eades III. 2026. A unification of graded and substructural logics. In *Proc. MFPS '26*. https://ul-fmf.github.io/mfps-sstt-2026/files/pdfs/mfps/MFPS26-4.pdf

[22] Douglas J. Howe. 1996. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.* 124, 2 (1996), 103–112. https://doi.org/10.1006/inco.1996.0008

[23] Andrej Ivašković, Alan Mycroft, and Dominic Orchard. 2020. Data-flow analyses as effects and graded monads. In *Proc. FSCD '20 (LIPIcs, Vol. 167)*, Zena M. Ariola (Ed.). 15:1–15:23. https://doi.org/10.4230/LIPIcs.FSCD.2020.15

[24] Junyoung Jang, Sophia Roshal, Frank Pfenning, and Brigitte Pientka. 2024. Adjoint natural deduction. In *Proc. FSCD '24 (LIPIcs, Vol. 299)*. 15:1–15:23. https://doi.org/10.4230/LIPIcs.FSCD.2024.15

[25] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.* 28 (2018), e20. https://doi.org/10.1017/S0956796818000151

[26] Max I. Kanovich, Stepan L. Kuznetsov, Vivek Nigam, and Andre Scedrov. 2019. Subexponentials in non-commutative linear logic. *Math. Struct. Comput. Sci.* 29, 8 (2019), 1217–1249. https://doi.org/10.1017/S0960129518000117

[27] G. A. Kavvos. 2020. Dual-context calculi for modal logic. *Log. Methods Comput. Sci.* 16, 3 (2020). https://doi.org/10.23638/LMCS-16(3:10)2020

[28] Olivier Laurent. 2026. Yalla: Yet another deep embedding of linear logic in Rocq. *J. Autom. Reason.* 70, 1 (2026), 9. https://doi.org/10.1007/s10817-026-09755-y

[29] Daniel R. Licata, Michael Shulman, and Mitchell Riley. 2017. A fibrational framework for substructural and modal logics. In *Proc. FSCD '17 (LIPIcs, Vol. 84)*, Dale Miller (Ed.). 25:1–25:22. https://doi.org/10.4230/LIPIcs.FSCD.2017.25

[30] Mohamed Yousri Mahmoud and Amy P. Felty. 2019. Formalization of metatheory of the Quipper quantum programming language in a linear logic. *J. Autom. Reason.* 63, 4 (2019), 967–1002. https://doi.org/10.1007/s10817-019-09527-x

[31] Danielle Marshall, Michael Vollmer, and Dominic Orchard. 2022. Linearity and uniqueness: An entente cordiale. In *Proc. ESOP '22 (Lect. Notes Comput. Sci., Vol. 13240)*, Ilya Sergey (Ed.). 346–375. https://doi.org/10.1007/978-3-030-99336-8_13

[32] Conor McBride. 2016. I Got Plenty o' Nuttin'. In *A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, Sam Lindley, Conor McBride, Phil Trinder, and Don Sannella (Eds.). Springer, 207–233. https://doi.org/10.1007/978-3-319-30936-1_12

[33] Alberto Momigliano and Frank Pfenning. 2003. Higher-order pattern complement and the strict lambda-calculus. *ACM Trans. Comput. Log.* 4, 4 (2003), 493–529. https://doi.org/10.1145/937555.937559

[34] Alberto Momigliano, Brigitte Pientka, and David Thibodeau. 2019. A case study in programming coinductive proofs: Howe's method. *Math. Struct. Comput. Sci.* 29, 8 (2019), 1309–1343. https://doi.org/10.1017/S0960129518000415

[35] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. 2008. Contextual modal type theory. *ACM Trans. Comput. Log.* 9, 3 (2008), 23:1–23:49. https://doi.org/10.1145/1352582.1352591

[36] Vivek Nigam and Dale Miller. 2009. Algorithmic specifications in linear logic with subexponentials. In *Proc. PPDP '09*, António Porto and Francisco Javier López-Fraguas (Eds.). 129–140. https://doi.org/10.1145/1599410.1599427

[37] Vivek Nigam, Elaine Pimentel, and Giselle Reis. 2016. An extended framework for specifying and reasoning about proof systems. *J. Log. Comput.* 26, 2 (2016), 539–576. https://doi.org/10.1093/logcom/exu029

[38] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative program reasoning with graded modal types. *Proc. ACM Program. Lang.* 3, ICFP (2019), 110:1–110:30. https://doi.org/10.1145/3341714

[39] Frank Pfenning. 2024. Snax. https://bitbucket.org/fpfenning/snax/src/main/. (Version 1.8).

[40] Frank Pfenning and Rowan Davies. 2001. A judgmental reconstruction of modal logic. *Math. Struct. Comput. Sci.* 11, 4 (2001), 511–540. https://doi.org/10.1017/S0960129501003322

[41] Jeff Polakow. 2001. *Ordered Linear Logic and Applications.* Ph.D. thesis. Carnegie Mellon University, Pittsburgh, Penn. http://reports-archive.adm.cs.cmu.edu/anon/2001/CMU-CS-01-152.pdf

[42] Klaas Pruiksma, William Chargin, Frank Pfenning, and Jason Reed. 2018. Adjoint logic. (April 2018). https://www.cs.cmu.edu/~fp/papers/adjoint18b.pdf (unpublished manuscript).

[43] Giselle Reis, Zan Naeem, and Mohammed Hashim. 2020. Sequoia: A playground for logicians (system description). In *Proc. IJCAR '20*, Nicolas Peltier and Viorica Sofronie-Stokkermans (Eds.). 480–488. https://doi.org/10.1007/978-3-030-51054-1_32

[44] Greg Restall. 2024. Substructural Logics. In *The Stanford Encyclopedia of Philosophy* (fall 2024 ed.), Edward N. Zalta and Uri Nodelman (Eds.). Metaphysics Research Lab, Stanford University.

[45] Sophia Roshal and Frank Pfenning. 2026. Ordered adjoint logic. In *Proc. IJCAR '26.* https://www.cs.cmu.edu/~fp/papers/ijcar26.pdf (to appear).

[46] Anders Schack-Nielsen and Carsten Schürmann. 2010. Curry-style explicit substitutions for the linear and affine lambda calculus. In *Proc. IJCAR '10 (Lect. Notes Comput. Sci., Vol. 6173)*, Jürgen Giesl and Reiner Hähnle (Eds.). 1–14. https://doi.org/10.1007/978-3-642-14203-1_1

[47] Peter Selinger and Benoît Valiron. 2006. A lambda calculus for quantum computation with classical control. *Math. Struct. Comput. Sci.* 16, 3 (2006), 527–552. https://doi.org/10.1017/S0960129506005238

[48] Kathrin Stark. 2019. *Mechanising syntax with binders in Coq.* Ph.D. thesis. Saarland University, Saarbrücken, Germany. https://www.ps.uni-saarland.de/~kstark/thesis/

[49] Kathrin Stark, Steven Schäfer, and Jonas Kaiser. 2019. Autosubst 2: Reasoning with multi-sorted de Bruijn terms and vector substitutions. In *Proc. CPP '19*, Assia Mahboubi and Magnus O. Myreen (Eds.). ACM, 166–180. https://doi.org/10.1145/3293880.3294101

[50] Thomas Streicher. 1993. *Investigations into intensional type theory.* Habilitation thesis. Ludwig-Maximilians-Universität München, Munich, Germany. https://www2.mathematik.tu-darmstadt.de/~streicher/HabilStreicher.pdf

[51] Victoria Vollmer, Danielle Marshall, Harley Eades III, and Dominic Orchard. 2025. A mixed linear and graded logic: Proofs, terms, and models. In *Proc. CSL '25 (LIPIcs, Vol. 326)*, Jörg Endrullis and Sylvain Schmitz (Eds.). 32:1–32:21. https://doi.org/10.4230/LIPIcs.CSL.2025.32

[52] Philip Wadler. 2012. Propositions as sessions. In *Proc. ICFP '12.* 273–286. https://doi.org/10.1145/2364527.2364568

[53] Aaron Weiss, Daniel Patterson, Nicholas D. Matsakis, and Amal Ahmed. 2019. Oxide: The essence of Rust. *CoRR* abs/1903.00982 (2019). arXiv:1903.00982

[54] James Wood. 2024. *A framework for semiring-annotated type systems.* Ph.D. thesis. University of Strathclyde, Glasgow. https://doi.org/10.48730/pa0y-ct71

[55] James Wood and Robert Atkey. 2022. A framework for substructural type systems. In *Proc. ESOP '22 (Lect. Notes Comput. Sci., Vol. 13240)*, Ilya Sergey (Ed.). 376–402. https://doi.org/10.1007/978-3-030-99336-8_14

[56] Bruno Xavier, Carlos Olarte, and Elaine Pimentel. 2022. A linear logic framework for multimodal logics. *Math. Struct. Comput. Sci.* 32, 9 (2022), 1176–1204. https://doi.org/10.1017/S0960129522000366

[57] Daniel Zackon, Chuta Sano, Alberto Momigliano, and Brigitte Pientka. 2025. Split decisions: Explicit contexts for substructural languages. In *Proc. CPP '25.* 257–271. https://doi.org/10.1145/3703595.3705888

Received 20yy-mm-dd