# Superstructural Reversible Logic

Zachary Sparks Amr Sabry
Indiana University
zasparks/sabry@indiana.edu

## Abstract

In some substructural logics, the memory used by proofs is treated as a first-class multiplicative resource, but the choices made by those proofs are not. Since we can convert between space and time complexity, these "resource conscious" logics are therefore not actually guaranteed to preserve memory—for example, linear logic allows the erasure and duplication of natural numbers with time complexity proportional to their size. In order to fully account for space-time tradeoffs, we augment contexts to track *all* information, not just multiplicative resources. This creates a reversible, fully resource-preserving logic which allows us to examine the hidden information effects in linear logic and study reversible computation from a proof-theoretic perspective.

***Categories and Subject Descriptors*** Theory of computation [*Models of computation*]; Theory of computation [*Semantics and reasoning*]; Mathematics of computing [*Information theory*]

***General Terms*** Languages, Theory

***Keywords*** bunched logic, isomorphisms, linear logic

## 1. Introduction

In the original article introducing linear logic [11], Girard describes it as "a resource-conscious logic." Wadler [30] explains this idea as follows:

> Truth is free. Having proved a theorem, you may use this proof as many times as you wish, at no extra cost. Food, on the other hand, has a cost. Having baked a cake, you may eat it only once. If traditional logic is about truth, then linear logic is about food.

Perhaps more seriously, Wadler continues to explain that linear logic can be used to reason about *memory resources* which, in general, cannot be "freely copied" or "foolishly discarded."

But whether it is about food or memory, linear logic does *not* view truth as a resource in the general case. For example, take the $\oplus$ connective, which represents disjunctive choice: a proof of $A \oplus B$ is either a proof of $A$ or a proof of $B$, similar to $\vee$ in traditional logic. Consider the following derivation:

$$\dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathbf{1}}\,\mathbf{1}R}{\mathbf{1} \vdash \mathbf{1}}\,\mathbf{1}L \quad \dfrac{\dfrac{}{\cdot \vdash \mathbf{1}}\,\mathbf{1}R}{\mathbf{1} \vdash \mathbf{1}}\,\mathbf{1}L}{\mathbf{1} \oplus \mathbf{1} \vdash \mathbf{1}}\,\oplus L$$

The assumption $\mathbf{1} \oplus \mathbf{1}$ represents a boolean value and hence contains one bit of information. The conclusion represents the unit value, which has no information. Where did the information contained in the *bool* value go? One can answer this question more clearly by looking at the following function of type $bool \multimap \mathbf{1}$, which is the computational counterpart of the above proof:

```
λ x : bool. if x then () else ()
```

The information in the input is represented by the *choice* of which branch is selected. Linear logic does not recognize choice as a resource, so it allows us to erase this information. We can even do the same with natural numbers [24]:

```
eraseNat zero = ()
eraseNat (succ n) = eraseNat n
```

Note that this function takes an amount of time linear in its argument, unlike the earlier function, which ran in constant time. `eraseNat` takes a resource consisting of a memory location holding a natural number and, by a process taking time proportional to the number, produces a trivial value $()$. This erases information, not space, but applying it to both elements of a pair of natural numbers would have the type $\mathbf{1} \otimes \mathbf{1}$, which is equivalent to the type $\mathbf{1}$. Thus, it allows us to take two (or more) memory cells containing nontrivial information and collapse them into one trivial memory cell, erasing both information *and* space.

We can also copy a natural number, with a similar result:

```
copyNat zero = (zero, zero)
copyNat (succ n) =
  let (n1, n2) = copyNat n in
    (succ n1, succ n2)
```

This function duplicates information in a way that bypasses the resource tracking infrastructure of linear logic. Just as `eraseNat` produced a trivial value in addition to some extra time, `copyNat` consumes both its input and some implicit input time as an extra argument and produces two natural numbers.

Linear logic may work to preserve resources when their structure is unknown, but it is not a truly resource-preserving logic. In order to preserve *all* resources, we must examine our notions of "truth" and "resource" more closely.

Martin-Löf [20] argues that "truth" is knowledge or information, i.e., that truth requires evidence and that evidence must be constructed using some resources and finitely communicated using physical means. More directly, Landauer [17] argues that "information is physical" and hence that information is a resource which must be conserved. Recently, James and Sabry [13] expand on this

idea and argue that information is a computational resource that ought to be exposed as such, and furthermore that our computational models should be founded on the principle of conservation of information.

As we explain in more detail in the next section and formalize in the remainder of the paper, linear logic does not recognize choice as a resource, as it allows choices to be freely created and discarded. Yet, as the examples above illustrate, these choices encode information, which *is* a resource. Technically, linear logic tracks *multiplicative* resources (that is, physical objects combined with $\otimes$) but does not consider *additive* resources (that is, choices represented by either $\oplus$ or $\&$). As we explain in Sec. 6, multiplicative resources can be viewed as representing *space* (i.e., memory), while additive resources can be viewed as representing *time* (i.e., choices). Because it is possible in computations to trade space for time and vice versa, this separation of resources is artificial. Instead, we wish to track *spacetime* resources, which allow arbitrary mixture of space and time connectives.

The above considerations lead us to investigate a logic that maintains information as spacetime resources. By the Landauer principle [3, 16], we can do this by making the logic *reversible* in the sense that every proof of conclusions $B$ from assumptions $A$ can be reversed to produce $A$ from $B$. The technical development of our logic is inspired by the logic of bunched implications (BI) [22]. Briefly speaking, we generalize linear logic contexts (which are built using $\otimes$ to account for multiplicative resources) to contexts built using two connectives $\otimes$ and $\oplus$, keeping track of both space and time. At the core of our approach is an explicit rule in our sequent calculus that provides access to the algebraic structural rules (hence the name "superstructural logic"):

$$\frac{\Gamma \sim \Gamma' \quad \Gamma' \vdash C}{\Gamma \vdash C}\ struct$$

To summarize, our major contributions are as follows:

- We define in Sec. 3 Reversible Logic (RL), a sequent calculus in which all entailments are isomorphisms. The methodology used to define this system exemplifies what we call *superstructural logic*, a way of writing sequent calculi that places emphasis on the structural properties of contexts. This is a generalization of BI [22] that allows added expressiveness and flexibility when defining a logic.
- We demonstrate the flexibility of superstructural logic by extending RL so that it is equivalent to linear logic (Sec. 4). This presentation exposes the hidden information effects in linear logic that cause it to preserve only spatial resources.
- We introduce a novel notion of resources, *spacetime* resources, which account for both the space and time needed by a computation (Secs. 3 and 6). Our logic can be used to reason about such resources allowing computations the flexibility of trading space for time and vice-versa.
- We give a computational interpretation for RL via Curry-Howard that is equivalent to the reversible $\Pi$ combinator calculus (Sec. 5). This equivalence demonstrates that each proof in Reversible Logic is in fact an isomoprhism.

## 2. Resource-Conscious Systems

The two resource-conscious systems of computation that have been most influential on this work are $\Pi$ [13], a reversible combinator calculus, and linear logic [11], a logic in which assumptions must be used exactly once. This contrasts with traditional logic, in which assumptions are *persistent* and may be used as many or as few times as needed; instead, linear assumptions are *ephemeral* and cannot be copied or erased inside contexts. We discuss these now to provide important background for the rest of the paper and motivate the need for reversible logic.

### 2.1 Information-preserving Isomorphisms

We review the language $\Pi$, as it has heavily influenced the rest of the paper.[1] The terms of $\Pi$ are not classical values and functions; rather, the terms witness isomorphisms that act on a separate level of values. In other words, the terms of $\Pi$ are proofs that certain shapes of values are isomorphic to one another.

***Data.*** $\Pi$ has two levels of data. The first level is traditional values:

$$v \quad ::= \quad () \mid \mathit{left}\ v \mid \mathit{right}\ v \mid (v, v)$$

which are classified by ordinary types:

$$b \quad ::= \quad 0 \mid 1 \mid b + b \mid b * b$$

Types include the empty type 0, the unit type 1, sum types $b_1 + b_2$, and product types $b_1 * b_2$. Values include $()$ which is the only value of type 1, $\mathit{left}\ v$ and $\mathit{right}\ v$ which inject $v$ into a sum type, and $(v_1, v_2)$ which builds a value of product type.

***Isomorphisms.*** The second level of data in $\Pi$ consists of witnesses of type isomorphisms, classified by the type $b_1 \leftrightarrow b_2$. Fig. 1 defines the base isomorphisms of $\Pi$ which form the core of the language. Each line of Fig. 1 introduces a pair of dual constants[2] that witness the type isomorphism in the middle. Note how these base isomorphisms have two readings: as a set of typing relations for a set of constants, and, if these axioms are seen as universally quantified, orientable statements, as transformations of the values. The (categorical) intuition here is that these axioms have computational content because they witness isomorphisms rather than merely stating an extensional equality.

These base isomorphisms are then extended with a small combinator language in order to form a congruence relation:

$$\frac{}{\mathit{id} : b \leftrightarrow b} \qquad \frac{c : b_1 \leftrightarrow b_2}{\mathit{sym}\ c : b_2 \leftrightarrow b_1} \qquad \frac{c_1 : b_1 \leftrightarrow b_2 \quad c_2 : b_2 \leftrightarrow b_3}{c_1 \fatsemi c_2 : b_1 \leftrightarrow b_3}$$

$$\frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{c_1 \oplus c_2 : b_1 + b_2 \leftrightarrow b_3 + b_4} \qquad \frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{c_1 \otimes c_2 : b_1 * b_2 \leftrightarrow b_3 * b_4}$$

### 2.2 Why not $\Pi$ ?

The language $\Pi$ works well as a model of reversible computation, but it is inconvenient both from computational and logical perspectives. Computationally, languages based entirely on combinators force a "pointfree" style which is often difficult to read and write. From a logical perspective, $\Pi$ gives a Hilbert-style system with a heavy emphasis on axioms, which suffers from the same problems.

Historically, natural deduction and sequent calculi were developed as more "natural" alternatives to such systems. In order to understand $\Pi$ as a deductive system, using proof-theoretic tools, we explore a sequent calculus formulation of reversible logic and computation. This formulation has several advantages: it may suggest a more intuitive syntax than a combinator calculus, it would allow us to reason about reversible logic and computation relative to other well-understood calculi, and it would make the language more extensible.

[1] The presentation in this section focuses on the simplest version of $\Pi$. Other versions include recursive types and trace operators but these extensions are orthogonal to the work emphasized in this paper.

[2] where $swap_*$ and $swap_+$ are self-dual.

$$
\begin{array}{rrcll}
identl_+ : & 0 + b & \leftrightarrow & b & : identr_+ \\
swap_+ : & b_1 + b_2 & \leftrightarrow & b_2 + b_1 & : swap_+ \\
assocl_+ : & b_1 + (b_2 + b_3) & \leftrightarrow & (b_1 + b_2) + b_3 & : assocr_+ \\
identl_* : & 1 * b & \leftrightarrow & b & : identr* \\
swap_* : & b_1 * b_2 & \leftrightarrow & b_2 * b_1 & : swap_* \\
assocl_* : & b_1 * (b_2 * b_3) & \leftrightarrow & (b_1 * b_2) * b_3 & : assocr_* \\
dist_0 : & 0 * b & \leftrightarrow & 0 & : factor_0 \\
dist : & (b_1 + b_2) * b_3 & \leftrightarrow & (b_1 * b_3) + (b_2 * b_3) & : factor
\end{array}
$$

**Figure 1.** $\Pi$ base isomorphisms

### 2.3 Linear logic

There do exist *substructural* logics [7, 23, 26] that preserve some resources by restricting the operations that are allowed on contexts. Linear logic [11] in particular requires that all assumptions be used exactly once, a property enforced by removing weakening and contraction. This rules out a generic constant function and restricts the multiplicative unit $\mathbf{1}$ to the reversible rule $\cdot \vdash \mathbf{1}$. We now investigate linear logic to explain how it comes close to, but does not reach, reversibility.

***Multiplicative connectives.*** One of the early claims about linear logic is that it is "a resource-conscious logic" [11]. In that common interpretation, usually exemplified with a vending machine model, the multiplicative propositions represent the simultaneous availability of distinct physical resources (e.g., candy *and* gum). Consider the left and right rules for $\otimes$ in intuitionistic linear logic:

$$\frac{\Delta_1 \vdash A \quad \Delta_2 \vdash B}{\Delta_1, \Delta_2 \vdash A \otimes B}\ \otimes R \qquad \frac{\Delta, A, B \vdash C}{\Delta, A \otimes B \vdash C}\ \otimes L$$

In these rules, resources may be shuffled around, but not thrown away. The right rule splits both the context and the conclusion, anj the left rule decomposes the connective into the context, allowing it to be examined more closely. In a classical setting, $\otimes$'s dual $\parr$ behaves similarly.

The rules for $\mathbf{1}$, the unit of $\otimes$, are also well-behaved:

$$\frac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma \vdash C}{\Gamma, \mathbf{1} \vdash C}\ \mathbf{1}L$$

The right rule allows only the empty context to prove $\mathbf{1}$. Since the empty context is the unit of the contextual "," operation, no information is lost here. In the left rule, the $\mathbf{1}$ is thrown away. In principle we can rewrite this as

$$\frac{\Gamma, \cdot \vdash C}{\Gamma, \mathbf{1} \vdash C}\ \mathbf{1}L'$$

but this would be identical to the original rule, since linear contexts may be implicitly shuffled and combined.

If we were to restrict ourselves to the multiplicative fragment, we would have a reversible logic, but not an interesting one. Note that all of the types we can denote using only $\mathbf{1}$ and $\otimes$ have exactly one inhabitant. In order to be able to express nontrivial types (such as booleans) we must also have choice in our logic. The additive connectives in linear logic fulfill this need.

***Additive connectives.*** Unfortunately, the additive connectives $\oplus$ and $\&$ are not as well-behaved as $\otimes$. Indeed, the misbehavior begins with their units: the unit $\top$ of $\&$ is commonly interpreted as a "wastebasket for irrelevant alternatives", as it can be proven from any assumptions, and the unit $\mathbf{0}$ of $\oplus$ can be used to prove anything. Clearly these do not preserve information, but what about the connectives themselves? Consider one of the right rules for $\oplus$:

$$\frac{\Delta \vdash A}{\Delta \vdash A \oplus B}\ \oplus R_1$$

Reading the rule from the bottom up, $B$ disappears. This is our first indication that linear logic does not preserve choices as resources, since anything can be injected into a larger choice from which the original information cannot be recovered. This behavior is similar to the contraction rule from standard propositional logic:

$$\frac{\Gamma, A \vdash C}{\Gamma, A, A \vdash C}\ \textsc{Cont}$$

If we want a logic that preserves information, we need to ensure that we are not allowed to apply rules like weakening or contraction anywhere. Even though linear logic removes these rules from the context, forbidding their use on $\otimes$, $\oplus$ and $\&$ still admit weakening- and contraction-like properties.

Another interesting property of the additive rules is that they are nonlinear at a meta level: not every metavariable is used exactly once in both the premise and the conclusion. In $\oplus R_1$, $B$ is erased in the premise. In $\&R$, the context itself is duplicated in the premises:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\ \&R$$

These rules do have legitimate uses, of course. In particular, we *should* be allowed to introduce the units of $\oplus$ and $\&$ whenever we want. Consider the following proof of $A \vdash A \oplus \mathbf{0}$:

$$\frac{\dfrac{}{A \vdash A}\ id_A}{A \vdash A \oplus \mathbf{0}}\ \oplus R_1$$

It is also possible to write an irreversible proof even between two equivalent types. Consider the following proof of $\mathbf{1} \oplus \mathbf{1} \vdash \mathbf{1} \oplus \mathbf{1}$:

$$\frac{\dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}R}{\mathbf{1} \vdash \mathbf{1}}\ \mathbf{1}L}{\mathbf{1} \vdash \mathbf{1} \oplus \mathbf{1}}\ \oplus R_1 \qquad \dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}R}{\mathbf{1} \vdash \mathbf{1}}\ \mathbf{1}L}{\mathbf{1} \vdash \mathbf{1} \oplus \mathbf{1}}\ \oplus R_1}{\mathbf{1} \oplus \mathbf{1} \vdash \mathbf{1} \oplus \mathbf{1}}\ \oplus L$$

This proof corresponds to the lambda calculus function $\lambda x.true$, albeit with more intermediate steps done to erase the variable from the context. The problem is subtle: we use $\oplus R_1$ twice, when really we should have to use each right rule exactly once in order for the proof to be an isomorphism. This is actually an instance of a general fact about linear logic where any proposition $A$ composed of only the base type operators is equivalent to $!A$ and therefore able to be duplicated and erased at will [6]. In particular, this affects booleans and natural numbers.

Recall that $\otimes$ preserves information by explicitly interacting with the context. On the right, it splits the context and passes part to each premise, and on the left it decomposes into the context. This works in part because the behavior of contexts preserves that one variety of information—contexts have $\cdot$ as their unit and can have

$$
\begin{array}{rcl}
\Gamma \oplus \mathbb{0} & \sim & \Gamma \\
\Gamma_1 \oplus \Gamma_2 & \sim & \Gamma_2 \oplus \Gamma_1 \\
\Gamma_1 \oplus (\Gamma_2 \oplus \Gamma_3) & \sim & (\Gamma_1 \oplus \Gamma_2) \oplus \Gamma_3 \\
\Gamma \otimes \mathbb{1} & \sim & \Gamma \\
\Gamma_1 \otimes \Gamma_2 & \sim & \Gamma_2 \otimes \Gamma_1 \\
\Gamma_1 \otimes (\Gamma_2 \otimes \Gamma_3) & \sim & (\Gamma_1 \otimes \Gamma_2) \otimes \Gamma_3 \\
\Gamma \otimes \mathbb{0} & \sim & \mathbb{0} \\
\Gamma_1 \otimes (\Gamma_2 \oplus \Gamma_3) & \sim & (\Gamma_1 \otimes \Gamma_2) \oplus (\Gamma_1 \otimes \Gamma_3)
\end{array}
$$

$$
\frac{}{\Gamma \sim \Gamma} \qquad \frac{\Gamma_1 \sim \Gamma_2}{\Gamma_2 \sim \Gamma_1} \qquad \frac{\Gamma_1 \sim \Gamma_2 \qquad \Gamma_2 \sim \Gamma_3}{\Gamma_1 \sim \Gamma_3}
$$

$$
\frac{\Gamma_1 \sim \Gamma_3 \qquad \Gamma_2 \sim \Gamma_4}{\Gamma_1 \oplus \Gamma_2 \sim \Gamma_3 \oplus \Gamma_4} \qquad \frac{\Gamma_1 \sim \Gamma_3 \qquad \Gamma_2 \sim \Gamma_4}{\Gamma_1 \otimes \Gamma_2 \sim \Gamma_3 \otimes \Gamma_4}
$$

**Figure 2.** Contextual equivalences in RL

their elements shuffled around at will, but nothing may be created or deleted.

The additive connectives exist outside the context and do not interact with it. It stands to reason that adding mechanisms for tracking choices in a context whose structural rules are information-preserving would lead to an information-preserving logic. We investigate this in the next section.

## 3. A Reversible Sequent Calculus

The technical development of our fully reversible logic is based on ideas from both Π and BI. We start from a computational model of information-preserving fully-reversible isomorphisms [13], which has two connectives $\oplus$ and $\otimes$ denoting choice and spatial conjunction, and use it to develop a variant of BI [22] with contexts built from these two connectives. We call the resulting system Reversible Logic, or RL, though it is just one instance of a logic defined using techniques that we call *superstructural logic*.

Throughout this section, we will use $\vdash$ as the sequent symbol for RL when it is unambiguously a RL proof. Otherwise, we will distinguish between linear logic sequents, $\vdash_{LL}$, and RL sequents, $\vdash_{RL}$.

### 3.1 Spacetime contexts

The logic of bunched implication treat contexts as trees with two forms of binary connective and their units. In our case, one connective serves as the internalization of *spatial* conjunction, which represents having multiple resources in different places at the same time, and another connective serves as the internalization of *temporal* conjunction, which represents having one resource that could be in any of several different states at any particular time. Treated as a whole, our contexts track *spacetime* resources.

To make things easier to read, we use $\oplus$ and $\otimes$ instead of commas and semicolons for the contextual connectives. The propositional versions of these connectives are $+$ and $\times$, since our logic can also be interpreted as a deductive system for mathematical equality on these operators:

$$
\begin{array}{rrcl}
\text{Propositions} & A, B & ::= & \mathbf{0} \mid \mathbf{1} \mid A + B \mid A \times B \\
\text{Contexts} & \Gamma, \Delta & ::= & \mathbb{0} \mid \mathbb{1} \mid A \mid \Gamma \oplus \Gamma \mid \Gamma \otimes \Gamma
\end{array}
$$

### 3.2 Contextual equivalence

Contexts are subject to the same structural rules as Π combinators, but lifted to act on the contextual operators, leaving propositional leaves alone. These modified rules are shown in Fig. 2.

If contexts do not include $\oplus$, we recover linear logic contexts. The new rules encode the (monoidal) behavior of additive resources and, crucially, that multiplicative resources distribute over additive ones:

$$
\Gamma_1 \otimes (\Gamma_2 \oplus \Gamma_3) \sim (\Gamma_1 \otimes \Gamma_2) \oplus (\Gamma_1 \otimes \Gamma_3)
$$

This rule allows multiplicative resources to be traded for additive ones and vice-versa, reifying the space-time tradeoff that is usually left implicit in programs. As we will see later, this ability is vital, both in terms of emulating linear logic's behavior (Sec. 4) and trading between space and time complexity (Sec. 6).

At first glance, distributivity appears to violate meta-linearity, as it duplicates or erases a copy of $\Gamma_1$. Looking more closely, though, we find that although it violates *syntactic* linearity, it does not violate *semantic* linearity—since $\oplus$ represents a choice between two alternatives, only one $\Gamma_1$ can exist at any time.

Distributivity and factoring provide a mechanism for choosing to delay or force choices, at will. This mechanism already exists implicitly in linear logic: consider an attempt to prove $A \oplus B, C \vdash (A \oplus B) \otimes C$. Without a predetermined proof search strategy or a cut rule, there are two choices: first, we could attempt to apply $\oplus L$:

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\, id_A}{A \vdash A \oplus B}\, \oplus R_1 \quad \dfrac{}{C \vdash C}\, id_C}{A, C \vdash (A \oplus B) \otimes C}\, \otimes_R \quad \dfrac{\dfrac{\dfrac{}{B \vdash B}\, id_B}{B \vdash A \oplus B}\, \oplus R_2 \quad \dfrac{}{C \vdash C}\, id_C}{B, C \vdash (A \oplus B) \otimes C}\, \otimes_R}{A \oplus B, C \vdash (A \oplus B) \otimes C}\, \oplus L
$$

However, note that this duplicates one of the branches of the proof. We might instead choose to apply $\otimes R$ first:

$$
\dfrac{\dfrac{}{A \oplus B \vdash A \oplus B}\, id_{A \oplus B} \quad \dfrac{}{C \vdash C}\, id_C}{A \oplus B, C \vdash (A \oplus B) \otimes C}\, \otimes R
$$

The former proof corresponds to deconstructing the choice first and using the common resource $C$ later (that is, $(\Gamma_1 \otimes \Gamma_2) \oplus (\Gamma_1 \otimes \Gamma_3)$); the latter proof uses the common resource first, then proceeds to deconstruct the choice (that is, $\Gamma_1 \otimes (\Gamma_2 \oplus \Gamma_3)$). Both proofs are, in some sense, the same. In reversible logic, the structure of the context dictates which choice can be made, but they can be switched between at will.

### 3.3 Inference rules

We now define the left and right rules for our reversible sequent calculus, RL. As in BI, we want the left rules to be able to act on a proposition nested arbitrarily deeply within the context, so instead of writing $\Gamma, A$ as in linear logic, we write $\Gamma[A]$ to mean a context $\Gamma$ with a single hole somewhere in it that is filled by $A$. More generally, we can fill a hole with an arbitrary context, $\Gamma[\Delta]$. We can also have a context with multiple holes in it, which we write with curly braces instead of square braces, $\Gamma\{\Delta\}$.

The rules for $\times$ work the same as in linear logic and BI, matching on the context on the right and decomposing the connective into its contextual version on the left:

$$
\frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma \otimes \Gamma' \vdash A \times B}\, \times R \qquad \frac{\Gamma[A \otimes B] \vdash C}{\Gamma[A \times B] \vdash C}\, \times L
$$

Now that we have a contextual version of $+$, its rules do the same thing:

$$
\frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma \oplus \Gamma' \vdash A + B}\, +R \qquad \frac{\Gamma[A \oplus B] \vdash C}{\Gamma[A + B] \vdash C}\, +L
$$

The units have similar rules, decomposing them into their contextual forms:

$$\frac{}{\mathbb{O} \vdash \mathbf{0}}\ \mathbf{0}R \qquad \frac{\Gamma[\mathbb{O}] \vdash C}{\Gamma[\mathbf{0}] \vdash C}\ \mathbf{0}L \qquad \frac{}{\mathbb{1} \vdash \mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma[\mathbb{1}] \vdash C}{\Gamma[\mathbf{1}] \vdash C}\ \mathbf{1}L$$

This is a small departure from linear logic, especially with the treatment of $\mathbf{0}$. The decomposition of $\mathbf{1}$ is also important, however, since unlike in linear logic, this unit cannot always be eliminated. Recall the linear $\mathbf{1}L$ rule:

$$\frac{\Gamma \vdash_{LL} C}{\Gamma, \mathbf{1} \vdash_{LL} C}\ \mathbf{1}L$$

This only works because there is only one contextual connective, and $\cdot$, the contextual version of $\mathbf{1}$, is its unit. In RL, we have no such guarantee that in $\Gamma[\mathbb{1}]$, the hole is appearing as an immediate child of a $\otimes$ node, so we must rely on the contextual rules to eliminate it when possible.

So far, the rules appear tautological, just allowing us to shift focus from the proposition to the context and vice-versa. This is also the case for multiplicative linear logic, but in addition to this shift of focus, the linear logic context can be silently re-ordered. In our case, we add a rule that explicitly shuffles the resources in the context using the transformations in Fig. 2. The rule is similar to one in BI, with the exception that we treat the contextual equivalence as another derivation rather than a side condition:

$$\frac{\Gamma \sim \Gamma' \quad \Gamma' \vdash C}{\Gamma \vdash C}\ struct$$

For example, a proof that $A \times B \vdash B \times A$ would look as follows:

$$\dfrac{\dfrac{\dfrac{}{A \otimes B \sim B \otimes A} \quad \dfrac{\dfrac{}{B \vdash B}\ id \quad \dfrac{}{A \vdash A}\ id}{B \otimes A \vdash B \times A}\ \times R}{A \otimes B \vdash B \times A}\ struct}{A \times B \vdash B \times A}\ \times L$$

The left and right rules in RL act as wrappers around the combinators that act on contexts. We believe there may be ways to relax these restrictions and make writing proofs in RL more user-friendly, but for now it is intended just as a basic sequent calculus for reversible logic.

### 3.4 Implication

It is also possible to naïvely add implication to RL, using an approach similar to the one in BI:

$$\frac{\Gamma \otimes A \vdash B}{\Gamma \vdash A \multimap B}\ \multimap R \qquad \frac{\Gamma' \vdash A \quad \Gamma[B] \vdash C}{\Gamma[(A \multimap B) \otimes \Gamma'] \vdash C}\ \multimap L$$

While there is nothing immediately wrong with these rules, they are, unfortunately, not reversible. For example, while we can easily prove $(A \multimap B) \times A \vdash B$, we cannot prove $B \vdash (A \multimap B) \times A$. There has been work done on adding implication to reversible languages [15], specifically with fractional types. However, any of the known possibilities would add sufficient complexity to the logic that we defer their investigation to a later paper in order to focus on the foundations here.

### 3.5 Soundness and Completeness

To demonstrate that RL is sound and complete, we show that our sequent calculus as presented has admissible identity and cut rules.

**Theorem 3.1** (Admissibility of identity)**.** *For all propositions $A$, $A \vdash A$.*

*Proof.* By induction on the proposition $A$. □

The proof of a usual statement of cut, unfortunately, does not proceed smoothly. We must first generalize the cut statement to multicut, as in the BI metatheory [25], but that is not yet strong enough for RL. The eventual result hinges on *contextual parametericity*, as follows:

**Lemma 3.2** (Contextual parametericity)**.** *For all multi-hole contexts $\Gamma$ and $\Gamma'$ with no copies of $A$ in them, if $\Gamma\{A\} \sim \Gamma'\{A\}$, then for all contexts $\Delta$, $\Gamma\{\Delta\} \sim \Gamma'\{\Delta\}$.*

Due to the factoring rule, this lemma only holds with the condition that $A$ does not occur in the contexts other than as a replacement in a hole. Otherwise, substituting in $\Delta$ would potentially prevent factoring from happening, even though it could happen before. For example, in the context:

$$(A \otimes B) \oplus (A \otimes C)$$

we can factor out $A$, but if we replace one copy with $\Delta$, we get:

$$(\Delta \otimes B) \oplus (A \otimes C)$$

and the rule can no longer immediately be applied.

*Proof.* By induction on the context $\Gamma$. □

With this lemma out of the way, we can finally state and prove cut:

**Theorem 3.3** (Admissibility of cut)**.** *If $\Delta \vdash A$, $\Gamma\{A\} \vdash C$, and $A$ does not occur outside a hole in $\Gamma$, then $\Gamma\{\Delta\} \vdash C$.*

*Proof.* As in BI [25], using the contextual parametericity lemma in the right commutative case of the *struct* rule. □

### 3.6 Reversibility

Since we have been writing this logic with the goal of reversibility in mind, it is important that we show that it is actually reversible. We will show in Sec. 5 that all proofs are isomorphisms by way of a translation between the computational interpretation of RL and Π. For now, we show that every proof is an equivalence.

This proof relies on the $\ulcorner - \urcorner$ operator from contexts to propositions, which turns each contextual connective and unit into its propositional counterpart.

**Theorem 3.4** (Logical reversibility)**.** *If $\Gamma \vdash A$, then $A \vdash \ulcorner \Gamma \urcorner$.*

*Proof.* Another simple induction on the structure of the derivation of $\Gamma \vdash A$. □

## 4. Back to Linear Logic

Now that we have a reversible logic, what does it take to get back to linear logic? Intuitively, we would like to be able to derive rules of inference for RL that are similar to those for linear logic. In this section, we show that this can be done with the addition of a single connective and a few structural rules, then prove that our embedding (which we call Linear Contextual Logic, or LCL) is sound and complete with respect to the appropriate fragment of judgmental linear logic [6].

### 4.1 Units: 0 and ⊤

We begin by adding the units, $\mathbf{0}$ and $\top$, of the linear connectives. Since we only have one additive unit in RL, we add another propositional unit, $\top$, and its contextual form, $\mathbb{T}$, to LCL, with the following right and left rules:

$$\frac{}{\mathbb{T} \vdash \top}\ \top R \qquad \frac{\Gamma[\mathbb{T}] \vdash C}{\Gamma[\top] \vdash C}\ \top L$$

We will add structural rules for $\mathbb{T}$ as we discover a need for them. Now, recall the rules[3] for $\mathbf{0}$ and $\top$ from linear logic:

$$\frac{}{\Gamma, \mathbf{0} \vdash_{LL} C}\ \mathbf{0}L \qquad \frac{}{\Gamma \vdash_{LL} \top}\ \top R$$

We would like to be able to derive similar rules in LCL:

$$\frac{}{\Gamma \otimes \mathbf{0} \vdash C}\ \mathbf{0}L' \qquad \frac{}{\Gamma \vdash \top}\ \top R'$$

Deriving the $\mathbf{0}L'$ rule proceeds smoothly at first:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \otimes \mathbb{O} \rightsquigarrow \mathbb{O}} \quad \begin{array}{c}\vdots\\ \mathbb{O} \vdash C\end{array}}{\Gamma \otimes \mathbb{O} \vdash C}\ struct}{\Gamma \otimes \mathbf{0} \vdash C}\ \mathbf{0}L$$

but we cannot yet prove $\mathbb{O} \vdash C$. In linear logic, the $\mathbf{0}L$ rule allows us to prove anything from an assumption $\mathbb{O}$; this is an implicit information effect that is not captured by standard linear contexts. We can try to add a structural rule to LCL that encapsulates this effect:

$$\mathbb{O} \sim \Gamma$$

However, this rule makes our logic unsound, as we can trivially prove equivalence of any two contexts with two applications of it, one in each direction: $\Gamma \sim \mathbb{O} \sim \Gamma'$.

The problem here is that RL context equivalences form an equivalence relation, but in linear logic, information effects make the effects that occur on temporal resources irreversible. We can fix this by replacing $\sim$ with an asymmetric relation $\rightsquigarrow$ and changing the structural rule to

$$\frac{\Gamma \rightsquigarrow \Gamma' \quad \Gamma' \vdash C}{\Gamma \vdash C}\ struct$$

We can still keep all the rules from RL in LCL, so long as we treat $\Gamma \sim \Gamma'$ as syntactic sugar for $\Gamma \rightsquigarrow \Gamma'$ and $\Gamma' \rightsquigarrow \Gamma$. This new expressive power allows us to define the correct structural rule for $\mathbb{O}$:

$$\mathbb{O} \rightsquigarrow \Gamma$$

which allows us to prove $\mathbb{O} \vdash C$ with a derived rule we will call $\mathbb{O}L$:

$$\dfrac{\dfrac{}{\mathbb{O} \rightsquigarrow C} \quad \dfrac{}{C \vdash C}\ id_C}{\mathbb{O} \vdash C}\ struct$$

thus completing our derivation of the rule $\mathbf{0}L'$.

With $\top R'$, we get stuck right away, but we can apply a similar intuition to get a new structural rule for $\mathbb{T}$: since linear logic allows any context to become $\top$ in $\top R$, we need to add a structural rule that explicitly captures that implicit information effect:

$$\Gamma \rightsquigarrow \mathbb{T}$$

and complete the derivation of $\top R'$:

$$\dfrac{\dfrac{}{\Gamma \rightsquigarrow \mathbb{T}} \quad \dfrac{}{\mathbb{T} \vdash \top}\ \top R}{\Gamma \vdash \top}\ struct$$

### 4.2 Linear $\oplus$

Recall the linear logic right[4] and left rules for $\oplus$:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\ \oplus R_1 \qquad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C}\ \oplus L$$

We would like to be able to derive similar rules for the $+$ connective in $RL$:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A + B}\ +R_1' \qquad \frac{\Gamma \otimes A \vdash C \quad \Gamma \otimes B \vdash C}{\Gamma \otimes (A + B) \vdash C}\ +L'$$

The rules we have already added for $\mathbf{0}$ are enough for the right rule:

$$\dfrac{\dfrac{}{\Gamma \rightsquigarrow \Gamma \oplus \mathbb{O}} \quad \dfrac{\Gamma \vdash A \quad \dfrac{}{\mathbb{O} \vdash B}\ \mathbb{O}L}{\Gamma \oplus \mathbb{O} \vdash A + B}\ +R}{\Gamma \vdash A + B}\ struct$$

but the left rule will require more work. We can begin a derivation as follows:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ (\Gamma \otimes A) \oplus (\Gamma \otimes B) \vdash C\end{array}}{\Gamma \otimes (A \oplus B) \vdash C}\ distrib_{\oplus}}{\Gamma \otimes (A + B) \vdash C}\ +L$$

Intuitively, we need to be able to apply $+R$ to be able to proceed, since the primary conjunction on the left is $\oplus$. We can do this with an application of $cut$, but we soon get stuck again:

$$\dfrac{\dfrac{\Gamma \otimes A \vdash C \quad \Gamma \otimes B \vdash C}{(\Gamma \otimes A) \oplus (\Gamma \otimes B) \vdash C + C}\ +R \quad \dfrac{\begin{array}{c}\vdots\\ C \oplus C \vdash C\end{array}}{C + C \vdash C}\ +L}{(\Gamma \otimes A) \oplus (\Gamma \otimes B) \vdash C}\ cut$$

Recall from section 2 that we made a comparison between linear logic's $\oplus R$ rules and the structural rule contraction:

$$\frac{\Gamma, A \vdash C}{\Gamma, A, A \vdash C}\ cont$$

This is exactly the kind of property we need to hold here on the $\oplus$ connective: the ability to contract $\Gamma \oplus \Gamma$ into $\Gamma$. We can add this context transition to LCL:

$$\Gamma \oplus \Gamma \rightsquigarrow \Gamma$$

and finish the derivation of $+L'$:

$$\dfrac{\dfrac{}{C \oplus C \rightsquigarrow C} \quad \dfrac{}{C \vdash C}\ id_C}{C \oplus C \vdash C}\ struct$$

### 4.3 Linear $\&$

Fortunately, adding rules for $\&$ is much more straightforward. In order to avoid confusion with $+$ and $\oplus$, we add the propositional connective $\&$ and the contextual connective ⓡ to LCL, with the obvious right and left rules for $\&$:

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1 \,ⓡ\, \Gamma_2 \vdash A \,\&\, B}\ \&R \qquad \frac{\Gamma[A \,ⓡ\, B] \vdash C}{\Gamma[A \,\&\, B] \vdash C}\ \&L$$

We need ⓡ to be a monoid with $\mathbb{T}$ as its unit, as well as have a congruence rule like the ones for $\otimes$ and $\oplus$. As with $\mathbb{T}$, we will add additional structural rules for ⓡ as we need them.

Recall the rules[5] for $\&$ from linear logic:

[3] In this section, we will use $\vdash_{LL}$ for linear logic entailment and $\vdash$ or $\vdash_{LCL}$ for LCL entailment.

[4] We will only present one of the right rules here; the other is symmetric and therefore redundant.

[5] Again, we only look at one of the left rules, since the other is symmetric.

$$\frac{\Gamma \vdash_{LL} A \quad \Gamma \vdash_{LL} B}{\Gamma \vdash_{LL} A \,\&\, B}\ \&R \qquad \frac{\Gamma, A \vdash_{LL} C}{\Gamma, A \,\&\, B \vdash_{LL} C}\ \&L_1$$

Again, we would like to be able to derive similar rules in LCL:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash_{LL} B}{\Gamma \vdash A \,\&\, B}\ \&R' \qquad \frac{\Gamma \otimes A \vdash C}{\Gamma \otimes (A \,\&\, B) \vdash C}\ \&L'_1$$

As with $+L'$, we get stuck early on when trying to derive $\&R'$: we would like to apply $\&R$, since $\&$ appears on the right, but we are unable to at first. Recall, however, that in linear logic, $\&R$ duplicates the context to be able to pass it to both premises. Therefore, we must add a structural rule that allows us to do the same thing:

$$\Gamma \rightsquigarrow \Gamma \mathbin{⅋} \Gamma$$

The derivation now proceeds smoothly:

$$\frac{\dfrac{}{\Gamma \rightsquigarrow \Gamma \mathbin{⅋} \Gamma} \quad \dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \mathbin{⅋} \Gamma \vdash A \,\&\, B}\ \&R}{\Gamma \vdash A \,\&\, B}\ struct$$

The derivation for $\&L'_1$ depends only on the rules we added for the unit of ⅋, just as in $+R_1$:

$$\frac{\dfrac{\dfrac{\dfrac{}{A \rightsquigarrow A} \quad \dfrac{}{B \rightsquigarrow \mathbb{T}}}{A \mathbin{⅋} B \rightsquigarrow A \mathbin{⅋} \mathbb{T}} \quad \dfrac{}{A \mathbin{⅋} \mathbb{T} \rightsquigarrow A}}{A \mathbin{⅋} B \rightsquigarrow A} \quad \Gamma \otimes A \vdash C}{\dfrac{\Gamma \otimes (A \mathbin{⅋} B) \vdash C}{\Gamma \otimes (A \,\&\, B) \vdash C}\ \&L}\ struct$$

Of course, since ⅋ is a new connective, we must make sure we have added the structural rules that make $\mathbb{T}$ its unit:

$$\Gamma \mathbin{⅋} \mathbb{T} \sim \Gamma$$

Also, while the case for $\&L'_2$ is symmetric, it requires that the ⅋ connective also be symmetric, which we can ensure with the following rule:

$$\Gamma_1 \mathbin{⅋} \Gamma_2 \sim \Gamma_2 \mathbin{⅋} \Gamma_1$$

### 4.4 Linear information effects

To summarize, the unidirectional rules we have added are:

$$\mathbb{O} \rightsquigarrow \Gamma \qquad \Gamma \rightsquigarrow \Gamma \mathbin{⅋} \Gamma$$

$$\Gamma \rightsquigarrow \mathbb{T} \qquad \Gamma \oplus \Gamma \rightsquigarrow \Gamma$$

These capture the implicit information effects inherent in linear logic in an explicit way, by allowing contraction for $\oplus$ and weakening for ⅋, as well as appropriate rules for their units.

### 4.5 Correctness

We now prove that superstructural linear logic is equivalent to intuitionistic linear logic. In order to do this, we need two operators: $[\![-]\!]$ , which translates linear contexts and propositions into their counterparts in RL, and $[-]$, which translates RL contexts and propositions into linear propositions. Both are defined in Fig. 3.

**Theorem 4.1** (Completeness of the embedding)**.** *If* $\Gamma \vdash_{LL} A$*, then* $[\![\Gamma]\!] \vdash_{LCL} [\![A]\!]$

*Proof.* All of the heavy lifting of the proof is done by the derived rules from the previous subsection: we replace rules for the additive connectives $\&$ and $\oplus$ with their derived versions, rules for $\otimes$ with their corresponding rules in LCL, and applications of exchange with a structural application of $swap_{\otimes}$. □

The proof of the embedding's soundness is more complicated. Unlike the completeness proof, we cannot define a simple translation on contexts, since there is no equivalent of $\oplus$ or ⅋ in a linear context. Instead, we turn LCL contexts directly into LL propositions.

**Theorem 4.2** (Soundness of the embedding)**.** *If* $\Gamma \vdash_{LCL} A$*, then* $[\Gamma] \vdash_{LL} [A]$

*Proof.* Most cases proceed straightforwardly, but the case of $struct$ bears closer investigation:

$$\frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma \equiv \Gamma' & \Gamma' \vdash_{LCL} C\end{array}}{\Gamma \vdash_{LCL} C}\ struct$$

We can apply the inductive hypothesis to $\mathcal{D}_2$ to get $[\Gamma'] \vdash_{LL} [C]$, but we still need to be able to replace $[\Gamma']$ with $[\Gamma]$. To proceed, we must prove a lemma relating context transitions and linear logic proofs:

**Lemma 4.3** (Soundness of contextual transitions)**.** *If* $\Gamma \rightsquigarrow \Delta$*, then* $[\Gamma] \vdash_{LL} [\Delta]$

Since our context transition rules have been chosen carefully, this proof proceeds straightforwardly, and we can apply the new lemma to $\mathcal{D}_1$ to get a proof that $[\Gamma] \vdash_{LL} [\Gamma']$. This means we can apply the cut rule of linear logic to get the following derivation:

$$\frac{[\Gamma] \vdash_{LL} [\Gamma'] \quad [\Gamma'] \vdash_{LL} [C]}{[\Gamma] \vdash_{LL} C}\ cut$$

□

This completes the proof of the embedding's soundness, but more importantly, it suggests a general technique: when embedding a language into CL, we should be able to use any deductive system for that language in place of a contextual combinator calculus. This enables us to keep the expressive power of CL's contexts without having to use a combinator calculus to act on contexts. In particular, the structural rule would change to:

$$\frac{\Gamma \vdash' \Delta \quad \Delta \vdash C}{\Gamma \vdash C}\ struct'$$

where $\vdash'$ is the contextual notion of entailment (in our example here, this would be linear logic).

## 5. Computational Interpretation

We assign a computational interpretation to RL's proofs via the Curry-Howard correspondence [12], creating a programming language we call $\lambda^R$. As usual, we will assume that all variable names are unique, achieving this property by tacitly renaming variables when needed [2].

To update contexts, we add variable annotations to propositions, leaving the other forms alone:

$$\Gamma \quad ::= \quad \mathbb{1} \mid \mathbb{O} \mid \Gamma_1 \otimes \Gamma_2 \mid \Gamma_1 \oplus \Gamma_2 \mid x : A$$

For proof terms, we need multiplicative and additive pairs, unit expressions, and let-notation for each of the left rules:

$$
\begin{array}{rcl}
[\![\cdot]\!] & = & \mathbb{1} \\
[\![\Gamma, A]\!] & = & [\![\Gamma]\!] \otimes [\![A]\!]
\end{array}
\qquad
\begin{array}{rcl}
[\![\mathbf{1}]\!] & = & \mathbf{1} \\
[\![\top]\!] & = & \top \\
[\![\mathbf{0}]\!] & = & \mathbf{0} \\
[\![A \otimes B]\!] & = & [\![A]\!] \times [\![B]\!] \\
[\![A \,\&\, B]\!] & = & [\![A]\!] \,\&\, [\![B]\!] \\
[\![A \oplus B]\!] & = & [\![A]\!] + [\![B]\!]
\end{array}
\qquad
\begin{array}{rcl}
[\mathbb{1}] & = & \mathbf{1} \\
[\mathbb{T}] & = & \top \\
[\mathbb{O}] & = & \mathbf{0} \\
[\Gamma_1 \otimes \Gamma_2] & = & [\Gamma_1] \otimes [\Gamma_2] \\
[\Gamma_1 \mathbin{⅋} \Gamma_2] & = & [\Gamma_1] \,\&\, [\Gamma_2] \\
[\Gamma_1 \oplus \Gamma_2] & = & [\Gamma_1] \oplus [\Gamma_2]
\end{array}
\qquad
\begin{array}{rcl}
[\mathbf{1}] & = & \mathbf{1} \\
[\top] & = & \top \\
[\mathbf{0}] & = & \mathbf{0} \\
[A \times B] & = & [A] \otimes [B] \\
[A \,\&\, B] & = & [A] \,\&\, [B] \\
[A + B] & = & [A] \oplus [B]
\end{array}
$$

**Figure 3.** Converting back and forth between LL and LCL contexts and propositions

$$\frac{}{x : A \vdash x : A}\ id$$

$$\frac{\Gamma \vdash M : A \quad \Gamma' \vdash N : B}{\Gamma \otimes \Gamma' \vdash \langle M, N \rangle : A \times B}\ \times R \qquad \frac{\Gamma \vdash M : A \quad \Gamma' \vdash N : B}{\Gamma \oplus \Gamma' \vdash [M, N] : A + B}\ +R$$

$$\frac{\Gamma[(y : A) \otimes (z : B)] \vdash M : C}{\Gamma[x : A \times B] \vdash \mathsf{let}\ \langle y, z \rangle = x\ \mathsf{in}\ M : C}\ \times L$$

$$\frac{\Gamma[(y : A) \oplus (z : B)] \vdash M : C}{\Gamma[x : A + B] \vdash \mathsf{let}\ [y, z] = x\ \mathsf{in}\ M : C}\ +L$$

$$\frac{}{\mathbb{O} \vdash 0 : \mathbf{0}}\ \mathbf{0}R \qquad \frac{\Gamma[\mathbb{O}] \vdash M : C}{\Gamma[x : \mathbf{0}] \vdash \mathsf{let}\ 0 = x\ \mathsf{in}\ M : C}\ \mathbf{0}L$$

$$\frac{}{\mathbb{1} \vdash 1 : \mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma[\mathbb{1}] \vdash M : C}{\Gamma[x : \mathbf{1}] \vdash \mathsf{let}\ 1 = x\ \mathsf{in}\ M : C}\ \mathbf{1}L$$

$$\frac{c : \Gamma \sim \Gamma' \quad \Gamma' \vdash M : C}{\Gamma \vdash \mathsf{do}\ c\ \mathsf{in}\ M : C}\ struct$$

**Figure 4.** Proof term assignments

$$
\begin{array}{rcl}
M & ::= & x \mid 1 \mid 0 \\
& \mid & \langle M_1, M_2 \rangle \mid [M_1, M_2] \\
& \mid & \mathsf{let}\ 1 = x\ \mathsf{in}\ M \mid \mathsf{let}\ 0 = x\ \mathsf{in}\ M \\
& \mid & \mathsf{let}\ \langle y, z \rangle = x\ \mathsf{in}\ M \mid \mathsf{let}\ [y, z] = x\ \mathsf{in}\ M \\
& \mid & \mathsf{do}\ c\ \mathsf{in}\ M
\end{array}
$$

The $c$ in "do $c$ in $M$" is a combinator, with syntax as defined in Sec. 2 and typing rules as defined in Fig. 2. As before, the combinator's job is to shuffle the context so it has the right shape to be passed in to $M$.

Proof term assignments are shown in Fig. 4. Using the sequent calculus for a proof term assignment means that variables act as locations within the context; in theory we could create an equivalent natural deduction system and assign proof terms to it, but this would add extra complexity to the terms and semantics.

Note that the pair form introduced in the $+R$ rule, $[M_1, M_2]$, is different from the value forms for $+$. This is because it ends up acting as a restricted form of case analysis based on which branch of the context is "really" active. A term $[M_1, M_2]$ can be applied to a left value or a right value, activating either $M_1$ or $M_2$, but not both.

### 5.1 Values and Environments

Since it is not higher-order, $\lambda^R$ maintains a distinction between proof terms and the values on which they act. In order to evaluate an application of a term to a value, we also need to maintain an environment. This will act like a dynamic version of type contexts, storing the same information as values on a separate syntactic level. As with static contexts, values can be bound to variables inside environments, but values cannot refer to variables or contexts.

As an alternative to environments, we may wish to use a substitution-based semantics, but due to the way in which we must treat sums, as well as the lack of first-class terms, this would be difficult, if not outright impossible. The usual presentation of environments just treats them as finite functions, which is also inadequate for our purposes, again due to the extra structure required by our treatment of sums. Just as we have nontrivial type contexts, we must also admit nontrivial environments at runtime.

$$\frac{}{\langle\rangle : \mathbf{1}} \qquad \frac{v_1 : A \quad v_2 : B}{\langle v_1, v_2 \rangle : A \times B}$$

$$\frac{v_1 : A}{\mathsf{inl}(v_1) : A + B} \qquad \frac{v_2 : B}{\mathsf{inr}(v_2) : A + B}$$

$$\frac{v : A}{(x = v) : (x : A)} \qquad \frac{}{\langle\!\langle\rangle\!\rangle : \mathbb{1}} \qquad \frac{E_1 : \Gamma_1 \quad E_2 : \Gamma_2}{\langle\!\langle E_1, E_2 \rangle\!\rangle : \Gamma_1 \otimes \Gamma_2}$$

$$\frac{E_1 : \Gamma_1}{\overline{\mathsf{inl}}(E_1) : \Gamma_1 \oplus \Gamma_2} \qquad \frac{E_2 : \Gamma_2}{\overline{\mathsf{inr}}(E_2) : \Gamma_1 \oplus \Gamma_2}$$

**Figure 5.** Value and environment typing rules

$$
\begin{array}{rcl}
v & ::= & \langle\rangle \mid \langle v_1, v_2 \rangle \mid \mathsf{inl}(v) \mid \mathsf{inr}(v) \\
E & ::= & \langle\!\langle\rangle\!\rangle \mid \langle\!\langle E_1, E_2 \rangle\!\rangle \mid \overline{\mathsf{inl}}(E) \mid \overline{\mathsf{inr}}(E) \mid x = v
\end{array}
$$

The typing rules for values and environments are shown in Fig. 5. The type of a value is a proposition, and the type of an environment is a type context. Note that they are declarative—in particular, the rules for injections into sum types are back to creating and erasing information. We allow this because runtime objects do not represent interesting proofs; information only needs to be preserved in the main proof terms, not the objects on which they act.

### 5.2 Operational Semantics

We now define a small step judgment, $M @ E \mapsto M' @ E'$, for the dynamic semantics, shown in Fig. 6. In the rules, whenever we encounter a let-expression, i.e., a binder, we create the appropriate bindings in the environment. If there was a choice between two possible bindings (as in the rules for let $[y, z] = x$ in $M$) and one of them was chosen based on the value of $x$ then the environment entry is itself tagged with the choice. The evaluation of pairs can in principle be done in parallel since the pair components do not, by construction, share any resources. The rules realize this by providing a non-deterministic choice: when evaluating $\langle M_1, M_2 \rangle$ one can either evaluation $M_1$ or $M_2$. A novel aspect of the semantics is that whenever we encounter an expression do $c$ in $M$ we dynamically use $c$ to restructure the environment. The rules for this action

$$\frac{}{(\mathsf{let}\ \langle y, z\rangle = x\ \mathsf{in}\ M)@E[x = \langle v_1, v_2\rangle] \mapsto M@E[\langle\!\langle y = v_1, z = v_2\rangle\!\rangle]} \qquad \frac{c\ E \Downarrow E'}{(\mathsf{do}\ c\ \mathsf{in}\ M)@E \mapsto M@E'}$$

$$\frac{M_1@E_1 \mapsto M_1'@E_1'}{\langle M_1, M_2\rangle@\langle\!\langle E_1, E_2\rangle\!\rangle \mapsto \langle M_1', M_2\rangle@\langle\!\langle E_1', E_2\rangle\!\rangle} \qquad \frac{M_2@E_2 \mapsto M_2'@E_2'}{\langle M_1, M_2\rangle@\langle\!\langle E_1, E_2\rangle\!\rangle \mapsto \langle M_1, M_2'\rangle@\langle\!\langle E_1, E_2'\rangle\!\rangle}$$

$$\frac{}{(\mathsf{let}\ [y, z] = x\ \mathsf{in}\ M)@E[x = \mathsf{inl}(v_1)] \mapsto M@E[\overline{\mathsf{inl}}(y = v_1)]} \qquad \frac{}{(\mathsf{let}\ [y, z] = x\ \mathsf{in}\ M)@E[x = \mathsf{inr}(v_2)] \mapsto M@E[\overline{\mathsf{inr}}(y = v_2)]}$$

$$\frac{M_1@E_1 \mapsto M_1'@E_1'}{[M_1, M_2]@\overline{\mathsf{inl}}(E_1) \mapsto [M_1', M_2]@\overline{\mathsf{inl}}(E_1')} \qquad \frac{M_2@E_2 \mapsto M_2'@E_2'}{[M_1, M_2]@\overline{\mathsf{inr}}(E_2) \mapsto [M_1, M_2']@\overline{\mathsf{inr}}(E_2')}$$

**Figure 6.** Dynamic semantics

$$\frac{}{id\ E \Downarrow E} \qquad \frac{c_1\ E \Downarrow E' \quad c_2\ E' \Downarrow E''}{(c_1 \fatsemi c_2)\ E \Downarrow E''} \qquad \frac{c_1\ E_1 \Downarrow E_1' \quad c_2\ E_2 \Downarrow E_2'}{(c_1 \otimes c_2)\ \langle\!\langle E_1, E_2\rangle\!\rangle \Downarrow \langle\!\langle E_1', E_2'\rangle\!\rangle}$$

$$\frac{c_1\ E_1 \Downarrow E_1'}{(c_1 \oplus c_2)\ \overline{\mathsf{inl}}(E_1) \Downarrow \overline{\mathsf{inl}}(E_1')} \qquad \frac{c_2\ E_2 \Downarrow E_2'}{(c_1 \oplus c_2)\ \overline{\mathsf{inr}}(E_2) \Downarrow \overline{\mathsf{inr}}(E_2')}$$

$$\frac{}{identl_\oplus\ \overline{\mathsf{inl}}(E) \Downarrow E} \qquad \frac{}{identr_\oplus\ E \Downarrow \overline{\mathsf{inl}}(E)} \qquad \frac{}{identl_\otimes\ \langle\!\langle E, \langle\!\langle\rangle\!\rangle\rangle\!\rangle \Downarrow E} \qquad \frac{}{identr_\otimes\ E \Downarrow \langle\!\langle E, \langle\!\langle\rangle\!\rangle\rangle\!\rangle}$$

$$\frac{}{swap_\oplus\ \overline{\mathsf{inl}}(E) \Downarrow \overline{\mathsf{inr}}(E)} \qquad \frac{}{swap_\oplus\ \overline{\mathsf{inr}}(E) \Downarrow \overline{\mathsf{inl}}(E)} \qquad \frac{}{swap_\otimes\ \langle\!\langle E_1, E_2\rangle\!\rangle \Downarrow \langle\!\langle E_2, E_1\rangle\!\rangle}$$

$$\frac{}{assocl_\oplus\ \overline{\mathsf{inl}}(E) \Downarrow \overline{\mathsf{inl}}(\overline{\mathsf{inl}}(E))} \qquad \frac{}{assocl_\oplus\ \overline{\mathsf{inr}}(\overline{\mathsf{inl}}(E)) \Downarrow \overline{\mathsf{inl}}(\overline{\mathsf{inr}}(E))} \qquad \frac{}{assocl_\oplus\ \overline{\mathsf{inr}}(\overline{\mathsf{inr}}(E)) \Downarrow \overline{\mathsf{inr}}(E)}$$

$$\frac{}{assocr_\oplus\ \overline{\mathsf{inl}}(\overline{\mathsf{inl}}(E)) \Downarrow \overline{\mathsf{inl}}(E)} \qquad \frac{}{assocr_\oplus\ \overline{\mathsf{inl}}(\overline{\mathsf{inr}}(E)) \Downarrow \overline{\mathsf{inr}}(\overline{\mathsf{inl}}(E))} \qquad \frac{}{assocr_\oplus\ \overline{\mathsf{inr}}(E) \Downarrow \overline{\mathsf{inr}}(\overline{\mathsf{inr}}(E))}$$

$$\frac{}{assocl_\otimes\ \langle\!\langle E_1, \langle\!\langle E_2, E_3\rangle\!\rangle\rangle\!\rangle \Downarrow \langle\!\langle\langle\!\langle E_1, E_2\rangle\!\rangle, E_3\rangle\!\rangle} \qquad \frac{}{assocr_\otimes\ \langle\!\langle\langle\!\langle E_1, E_2\rangle\!\rangle, E_3\rangle\!\rangle \Downarrow \langle\!\langle E_1, \langle\!\langle E_2, E_3\rangle\!\rangle\rangle\!\rangle}$$

$$\frac{}{dist\ \langle\!\langle E_1, \overline{\mathsf{inl}}(E_2)\rangle\!\rangle \Downarrow \overline{\mathsf{inl}}(\langle\!\langle E_1, E_2\rangle\!\rangle)} \qquad \frac{}{dist\ \langle\!\langle E_1, \overline{\mathsf{inr}}(E_3)\rangle\!\rangle \Downarrow \overline{\mathsf{inr}}(\langle\!\langle E_1, E_3\rangle\!\rangle)}$$

$$\frac{}{factor\ \overline{\mathsf{inl}}(\langle\!\langle E_1, E_2\rangle\!\rangle) \Downarrow \langle\!\langle E_1, \overline{\mathsf{inl}}(E_2)\rangle\!\rangle} \qquad \frac{}{factor\ \overline{\mathsf{inr}}(\langle\!\langle E_1, E_3\rangle\!\rangle) \Downarrow \langle\!\langle E_1, \overline{\mathsf{inr}}(E_3)\rangle\!\rangle}$$

(no rule for $dist_{\mathrm{O}}$ or $factor_{\mathrm{O}}$)

**Figure 7.** Environment evaluation

$$\frac{}{x@(x = v)\ \mathsf{final}} \qquad \frac{}{1@\langle\!\langle\rangle\!\rangle\ \mathsf{final}} \qquad \frac{M_1@E_1\ \mathsf{final} \quad M_2@E_2\ \mathsf{final}}{\langle M_1, M_2\rangle@\langle\!\langle E_1, E_2\rangle\!\rangle\ \mathsf{final}} \qquad \frac{M_1@E_1\ \mathsf{final}}{[M_1, M_2]@\overline{\mathsf{inl}}(E_1)\ \mathsf{final}} \qquad \frac{M_2@E_2\ \mathsf{final}}{[M_1, M_2]@\overline{\mathsf{inr}}(E_2)\ \mathsf{final}}$$

**Figure 8.** Final configurations

on environments are essentially the same rules for the operational semantics of Π [13]: they are in Fig. 7.

We also define a judgment $M@E\ \mathsf{final}$ for when $M$ is finished evaluating in the environment $E$, shown in Fig. 8. Note that the additive pair $[M_1, M_2]$ might be final in one environment but not in another; only one branch will be evaluated, but we still keep both around. We can now define the full evaluation of an application of an expression to a value, written $M@E \mapsto^* M'@E'$, in the usual way:

$$\frac{M@E\ \mathsf{final}}{M@E \mapsto^* M@E} \qquad \frac{M@E \mapsto M'@E' \quad M'@E' \mapsto^* M''@E''}{M@E \mapsto^* M''@E''}$$

### 5.3 Safety

We prove progress and preservation [32] lemmas to demonstrate that $\lambda^R$ is typesafe. Both proofs proceed straightforwardly by induction, so we omit them here.

**Theorem 5.1** (Progress)**.** *If $E : \Gamma$ and $\Gamma \vdash M : A$, then either $M@E$ final or there exist $M'$ and $E'$ such that $M@E \mapsto M'@E'$.*

**Theorem 5.2** (Preservation)**.** *If $E : \Gamma$, $\Gamma \vdash M : A$, and $M@E \mapsto M'@E'$, then $E' : \Gamma'$ and $\Gamma' \vdash M' : A$.*

### 5.4 Equivalence to $\Pi$

We now show that $\lambda^R$ is equivalent to $\Pi$. Proving that any $\Pi$ combinator has an equivalent $\lambda^R$ term is simple, since it involves only decomposing the proposition in the context and ending with an application of the input combinator to the structural rule. The other direction requires slightly more effort. First, we must define a translation $\pi$ from $\lambda^R$ terms into $\Pi$ combinators. The translation removes all the names and the associated let-expressions:

$$\begin{array}{rcl}
\pi(x) & = & id \\
\pi(1) & = & id \\
\pi(0) & = & id \\
\pi(\langle M_1, M_2 \rangle) & = & \pi(M_1) \otimes \pi(M_2) \\
\pi([M_1, M_2]) & = & \pi(M_1) \oplus \pi(M_2) \\
\pi(\mathsf{let}\ _\ \mathsf{in}\ M) & = & \pi(M) \\
\pi(\mathsf{do}\ c\ \mathsf{in}\ M) & = & c \fatsemi \pi(M)
\end{array}$$

**Theorem 5.3** (Soundness of types over translation)**.** *If $\Gamma \vdash M : A$, then $\pi(M) : \ulcorner \Gamma \urcorner \leftrightarrow A$, where $\ulcorner \Gamma \urcorner$ views a context $\Gamma$ as a type.*

*Proof.* The proof proceeds by induction on the derivation $\Gamma \vdash M : A$. Note that this justifies our collapsing of all the let forms (i.e., the left rules) into one case, since the context in the premise and conclusion both reify to the same type. □

**Theorem 5.4** (Equivalence to $\Pi$)**.** *If $E : \Gamma$, $\Gamma \vdash M : A$, $\pi(M) = c$, and $M@E \mapsto^* M'@E'$, then $c \ulcorner E \urcorner \Downarrow \ulcorner E' \urcorner$, where $\ulcorner E \urcorner$ views an environment as a value.*

*Proof.* This also proceeds by induction on $M$. □

Now that we know $\lambda^R$ is equivalent to $\Pi$, we also know that it is a reversible language. To get the reversal of a proof $M$, we just have to convert the inverse of $\pi(M)$ back into $\lambda^R$, in the manner described above.

## 6. Resource Interpretation

Categorically, RL models a *bimonoidal category*, i.e., a category with two symmetric monoidal structures $(0, +)$ and $(1, *)$ and with multiplication distributing over addition. In this section, we investigate a model based on *consistent timelines* and create a visual notation for them in order to explore the space-time tradeoffs present in RL more concretely.

### 6.1 Consistent timelines

Intuitively, we want to think of the types in our system as possible events, and values as consistent timelines of events. From this perspective, $A * B$ denotes spatial composition, where both $A$ and $B$ must happen in any order, and $A + B$ denotes temporal composition, where exactly one of $A$ and $B$ must happen at a given time. The event 1 is then trivial, and 0 is the impossible event. Spatial composition can be thought of creating two locations in which events must happen, whereas temporal composition creates an event that is a choice between two sub-events.

Take the type $2 = 1 + 1$, representing a choice between two trivial events. There are two possible timelines here, just as we would hope. What happens in $2 + 2$ and $2 * 2$? In the former case, we are given an immediate choice between two lesser choices, resulting in a total of four possible timelines; in the latter case, we must make two different choices simultaneously, which also results in four possible timelines.

Both encodings represent the same number of possibilities, but our denotation distinguishes between them. $2 + 2$ only takes up one "cell" of space, but contains two temporally-sequenced events, whereas $2 * 2$ creates two cells whose events may happen concurrently. This suggests that each type has a *depth*—an upper bound on the number of events that need to occur (or alternatively, checks that need to be made) in order to reach the end of a timeline—and a *size*—an upper bound on the number of memory cells (created by spatial conjunction) that will be created over the course of a timeline. We can define these metrics as follows:

```
depth 1 = 0
depth 0 = 0
depth (A * B) = max (depth A, depth B)
depth (A + B) = 1 + max (depth A, depth B)



size 1 = 1
size 0 = 0
size (A * B) = size A + size B
size (A + B) = max (size A, size B)
```

Concretely, the depth and size of $2+2$ are 2 and 1, respectively, whereas the depth and size of $2*2$ are 1 and 2. Thus, we can choose between otherwise equivalent representations of a type with four alternatives in order to minimize either space or time complexity, at the expense of the other.

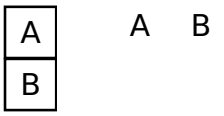
As presented earlier, the isomorphisms provide the mechanism for making this tradeoff. If we wish to convert $2 * 2$ into $2 + 2$ in order to save some space, we must first apply distributivity, resulting in the type $(2 * 1) + (2 * 1)$. This intermediate step is actually much worse than either alternative—its depth and size are both two—but there are also two superfluous multiplications by 1 being done. Removing these converts the type to $2 + 2$. Distributivity was the first step towards making the representation more space-efficient; conversely, factoring out a common number is important when making something more time-efficient.

Informally, we note that the depth and size operations correspond nicely with the operational semantics: the depth of a type is an upper bound on the number of times the $+L$ rule must do a case analysis on a value of that type and its subterms, and the size of a type is an upper bound on the number of pairs $\langle v_1, v_2 \rangle$ that occur in a value and its subterms. Put differently, given a value $v : A$, `size A` represents the numbers of cells used by $v$ in the environment, and `depth A` represents the maximum number of times the dynamic semantics could pattern match on $v$ and its subterms using $\mathsf{let}\ [y, z] = x\ \mathsf{in}\ M$.

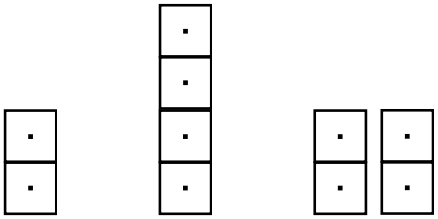
### 6.2 Modeling timelines

We formalize the notion of resources using a graphical notation for events and timelines.

The denotation of 1 is an endpoint; the denotation of 0 is empty space. The denotation of $A+B$ is the denotation of $A$ drawn on top of the denotation of $B$ and separated by a border, and the denotation of $A * B$ is just the denotation of $A$ written next to the denotation of $B$:



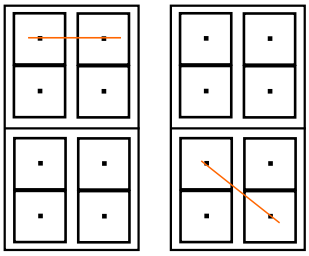

To simplify drawings and emphasize that $+$ is associative, we will write multiple consecutive applications of $+$ as a single stack of cells. For example, the denotations of 2, $2+2$, and $2*2$ would be as follows:



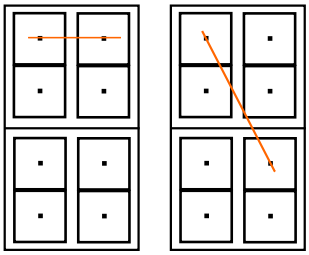
Now we can formally define a consistent timeline recursively on the structure of a diagram:

- An endpoint (the denotation of 1) has exactly one timeline through it. Empty space (the denotation of 0) has no timelines through it.
- A timeline through a stack of alternatives $A_1 + A_2 + ... + A_n$ is a timeline through exactly one of the possible alternatives.
- A timeline through a spatial composition of cells $A_1 * A_2 * ... * A_n$ is a timeline through *all* of the cells, in any order[6].

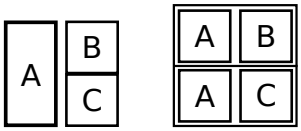
For example, the following would be a consistent timeline through the diagram for $((2+2)*(2+2))+((2+2)*(2+2))$ times itself:



but the following would not, since it attempts to skip between multiple disjoint alternatives at once in the second cell:



From this definition, it is evident that simple rules such as commutativity, associativity, and units hold, but we take a closer look at distributivity, which says that the following two diagrams should admit the same number of paths:




Since a timeline must pass through a column in exactly one point, this must be true, since the lower diagram cannot pass through both copies of $A$. Going from the bottom diagram to the top (via factoring) then acts as a sort of hash-consing by factoring out the common occurrences of $A$.

## 7. Related Work

***Reversibility.*** The history of reversible logic goes back to the papers by Toffoli and Fredkin in the early eighties [9, 29]. These papers and much of the following research focused on designing energy efficient hardware circuits.

Recently, reversibility has taken a more central, even foundational role in the sense that there appears to be a convergence of ideas from several distinct research communities (physics, mathematics, and computer science) towards replacing *equalities* by *isomorphisms*. The combined programme has sparked a significant amount of research that unveiled new and surprising connections between geometry, algebra, logic, and computation (see [1] for an overview of some of the connections). In the physics community, Landauer [16, 18], Feynman [8], and others have interpreted the laws of physics as fundamentally related to computation. The great majority of these laws are formulated as equalities between different physical observables which is unsatisfying: *different* physical observables should not be related by an *equality*. It is more appropriate to relate them by an *isomorphism* that witnesses, explains, and models the process of transforming one observable to the other.

In the mathematics and logic community, Martin-Löf developed an extension of the simply typed $\lambda$-calculus originally intended to provide a rigorous framework for constructive mathematics [19]. This theory has been further extended with *identity types* representing the proposition that two terms are "equal." (See [28, 31] for a survey.) Briefly speaking, given two terms $a$ and $b$ of the same type $A$, one forms the type $\mathtt{Id}_A(a,b)$ representing the proposition that $a$ and $b$ are equal: in other words, a term of type $\mathtt{Id}_A(a,b)$ witnesses, explains, and models the process of transforming $a$ to $b$ and vice-versa. In the computer science community, the theory and practice of type isomorphisms is well-established. Originally, such type isomorphisms were motivated by the pragmatic concern of searching large libraries of functions by providing one of the many possible isomorphic types for the desired function [27].

More recently, type isomorphisms have taken a more central role as *the* fundamental computational mechanism from which more conventional, i.e., irreversible computation, is derived. Work on the $\Pi$ combinator calculus [4, 13–15] started with the notion of type isomorphism and developed from it a family of programming languages in which computation is an isomorphism preserving information-theoretic entropy.

***Contextual logics.*** Reversible logic is not the first logic to take a superstructural approach to logic definition, though we believe ourselves to be the first to name the approach and embrace it fully. Deep inference [5] goes in a similar direction by focusing more on the structure and manipulation of contexts than a traditional logic. Bunched implications [22] also takes contexts more seriously and adds an extra connective, which was the original inspiration for superstructural logic's treatment of contexts. BI, however, explicitly

[6] If we were exploring a noncommutative structure, we would want this to be explicitly sequential, but since spatial composition in RL is commutative we allow this to happen in any order. This is consistent with our intuition that spatial composition creates memory cells that may be accessed in parallel.

eschews distributivity, which is central to RL; its contextual forms are distinguished by whether or not they allow weakening and contraction. Some work in mathematical logic [10] has an approach similar to superstructural logic, replacing propositional connectives with separate connectives on which structural rules are allowed to act.

## 8. Conclusion

We have demonstrated a general approach for writing substructural or otherwise resource-conscious that is places a heavy emphasis on the structural rules of the logic and that subsumes prior approaches. We have used this superstructural approach to develop a sequent calculus for reversible logic. We have given this sequent calculus a computational interpretation, which we used to show that it is computationally equivalent to prior approaches to reversible computing, and a resource interpretation to show that it preserves a combined notion of spatial and temporal resources.

One of the major contributions of this approach to viewing logic is that it provides insight into some of the problems with linear logic. Linear logic's preservation of spatial resources but not temporal resources is implicit in usual presentations, but becomes clear when viewed through superstructural logic. We hope this will aid in the development of linear-like logics that explicitly track information creation and deletion.

Using the machinery developed here, we plan to investigate reversible computation from a proof-theoretic perspective, which we believe will provide a huge advantage for understanding how to add more type-level features to a reversible programming language. In particular, the translation from a language with information effects to $\Pi$ given by James and Sabry [13] was syntax-directed rather than type-directed; we believe that we may be able to use existential types, as in closure conversion [21], to fix this problem and make sure source language terms of the same type translate to target language terms of the same type.

There are also opportunities for creating a more robust, user-friendly programming language out of $\lambda^R$. Right now, it is a combinator calculus with slightly more syntax; we aim to create a more lambda calculus-like language with more intuitive syntax, but the same semantics. There are a few ways in which we feel we could make the combinators act implicitly rather than explicitly, lightening the burden on the programmer. This would also make it more feasible to create a $\lambda^R$ DSL in an existing programming language, such as Haskell or Agda.

One of the major drawbacks of $\lambda^R$ is its lack of higher-order functions. As investigated in a recent paper [15], the basic idea is to add *fractional types* which are duals to product types. The technical details are however non-trivial. James et. al [15] only partially solve the problem by restricting the language (removing the empty type) and achieve reversibility only in a trivial, relational sense. We plan on investigating this idea from a proof-theoretic perspective.

## References


[1] J. Baez and M. Stay. Physics, topology, logic and computation: a rosetta stone. *New Structures for Physics*, 2011.

[2] H. P. Barendregt. *The Lambda Calculus, its Syntax and Semantics, Revised second edition*. North-Holland, 1984.

[3] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17, 1973.

[4] W. J. Bowman, R. P. James, and A. Sabry. Dagger Traced Symmetric Monoidal Categories and Reversible Programming. In *RC*, 2011.

[5] K. Brünnler. *Deep inference and symmetry in classical proofs*. Logos Verlag, 2003.

[6] B.-Y. E. Chang, K. Chaudhuri, and F. Pfenning. A Judgmental Analysis of Linear Logic. Technical Report CMU-CS-03-131R, Carnegie Mellon University, Apr. 2003.

[7] K. Dosen and P. Schroeder-Heister, editors. *Substructural Logics*. Oxford University Press, 1993.

[8] R. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21, 1982.

[9] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3), 1982.

[10] N. Galatos and H. Ono. Cut elimination and strong separation for substructural logics: An algebraic approach. *Annals of Pure and Applied Logic*, 161(9), 2010.

[11] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 1987.

[12] W. A. Howard. The formulas-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980. Reprint of 1969 article.

[13] R. P. James and A. Sabry. Information effects. In *POPL*, 2012.

[14] R. P. James and A. Sabry. Isomorphic interpreters from logically reversible abstract machines. In *RC*, 2012.

[15] R. P. James, Z. Sparks, J. Carette, and A. Sabry. Fractional types. Submitted for publication, 2013.

[16] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5, July 1961.

[17] R. Landauer. Information is physical. *Physics Today*, 44(5), 1991.

[18] R. Landauer. The physical nature of information. *Physics Letters A*, 1996.

[19] P. Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. *The Journal of Symbolic Logic*, 49(1), Mar. 1984.

[20] P. Martin-Löf. On the Meanings of the Logical Constants and the Justifications of the Logical Laws. *Nordic Journal of Philosophical Logic*, 1(1), May 1996.

[21] Y. Minamide, G. Morrisett, and R. Harper. Typed closure conversion. In *POPL*, New York, NY, USA, 1996. ACM.

[22] P. W. O'Hearn and D. J. Pym. The Logic of Bunched Implications. *Bulletin of Symbolic Logic*, 1999.

[23] F. Paoli. *Substructural Logics: A Primer*. Kluwer Academic Publisher, 2002.

[24] F. Pfenning. Recursive types. `http://www.cs.cmu.edu/~fp/courses/98-linear/handouts/rectypes.pdf`, 1998.

[25] D. J. Pym. *The semantics and proof theory of the logic of bunched implications*. Kluwer Academic Pub, 2002.

[26] G. Restall. *An Introduction to Substructural Logics*. Routledge, London, 2000.

[27] M. Rittri. Using types as search keys in function libraries. In *FPCA*, 1989.

[28] T. Streicher. Investigations into intensional type theory. Habilitationsschrift, Universität München, 1993.

[29] T. Toffoli. Reversible computing. In *Automata, Languages and Programming*. Springer-Verlag, 1980.

[30] P. Wadler. A taste of linear logic. In A. M. Borzyszkowski and S. Sokolowski, editors, *Mathematical Foundations of Computer Science*, Gdańsk, Poland, 1993. Springer-Verlag LNCS 781.

[31] M. Warren. *Homotopy theoretic aspects of constructive type theory*. PhD thesis, Carnegie-Mellon University, 2008.

[32] A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1), Nov. 1994.