# A Quite Brief Introduction to Linear Types

I don't think we should talk about this anymore. – Jesse

## 1 Context

Weakening and contraction are two fundamental pieces of constructivist logic. Weakening talks about the following situation (in the sequent calculus form that you're used to):

$$\frac{\Gamma \vdash \Sigma}{\Gamma, A \vdash \Sigma}$$

In other words, if some environment $\Gamma$ implies some $\Sigma$, then extending that environment with an $A$ still implies the same $\Sigma$.

Contraction talks about the following situation:

$$\frac{\Gamma, A, A, \vdash \Sigma}{\Gamma, A \vdash \Sigma}$$

Here, if some environment with an element $A$ implies some $\Sigma$, then removing an $A$ from that environment (leaving just one) still allows us to conclude $\Sigma$.

When we remove these two structural rules (at least for some types) we get linear logic, where all variables must be used *exactly* once.

In the wild (e.g., in programming languages), we don't tend to remove these structural rules for every variable. Fully linear programs don't tend towards the interesting side of the spectrum, since things like loops, etc. are hard to do. We therefore allow some non-linear types, called "unlimited," and we denote them with a $!\sigma$ (read "of course sigma").

## 2 Kludging the STLC into Something Like ILL

Languages like Rust which have linear types (or maybe affine types, depending on which forum posts you read) are quite full-featured and complicated. It would be well beyond the scope of this one lecture to talk about the features of Rust's type system.

Instead, let's consider how we might massage something like the simply-typed lambda calculus into having linear types. We'll go typing rule by typing rule, in order of least to most ~~confusing~~ interesting.

## 2.1 Functions

### 2.1.1 Introduction (Lollipop)

Functions in ILL are trivially different from the STLC (here, we use $\multimap$ instead of $\to$).

$\multimap$ Introduction

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.e : \sigma \multimap \tau}$$

### 2.1.2 Elimination (Application)

Function elimination is in the same boat. Note, however, that a separate type environment $\Delta$ provides the information that proves that $f$ has type $\sigma$.

$\multimap$ Elimination

$$\frac{\Gamma \vdash e : \sigma \multimap \tau \quad \Delta \vdash f : \sigma}{\Gamma, \Delta \vdash e\ f : \tau}$$

**Food for thought:** Why can't we just use one type environment $\Gamma$ to prove things about both $e$ and $f$?

## 2.2 Identity

At first glance, the identity rule looks the same as it did in the STLC. Take care, though, to note the absence of a $\Gamma$ in the type environment – it's *only* $x$.

Identity

$$\frac{}{x : \sigma \vdash x : \sigma}$$

**Food for thought:** to better grok why the type environment is just $x$, note that we could also have written it as $!\Gamma, x$, where $!\Gamma$ is a type environment made up of only unlimited types.

## 2.3 Manipulating Unlimited Variables

### 2.3.1 Weakening

If an unlimited type variable $x$ isn't needed to prove that an expression $e$ has some type (i.e., that the expression still types even with the smaller environment), then we can safely discard it from the type environment.

Weakening

$$\frac{\Gamma \vdash e : \tau}{\Gamma, x :!\sigma \vdash \text{discard } x \text{ in } e : \tau}$$

**Food for thought:** think back to the identity rule and what it is about weakening that implies that the typing environment can safely be *just* $x$ (with no unlimited types).

### 2.3.2 Contraction

If we have some unlimited type variable, we can freely make copies of that variable and give these copies different names (so long as doing so yields a program that still types).

Contraction

$$\frac{\Gamma, y :!\sigma, z :!\sigma \vdash e : \tau}{\Gamma, x :!\sigma \vdash \text{copy } x \text{ as } y, z \text{ in } e : \tau}$$

## 2.4 Switching Between Limited and Unlimited

### 2.4.1 Dereliciton

We can turn an expression $e$ of unlimited type $!\sigma$ into the same expression with the linear type $\sigma$.

Dereliction

$$\frac{\Gamma \vdash e :!\sigma}{\Gamma \vdash \text{derelict } e : \sigma}$$

### 2.4.2 Promotion

We can also promote an expression $e$ of linear type $\sigma$ into the same expression of unlimited type $!\sigma$. There is a caveat, though – we can only do this if the type environment $\Gamma$ is unlimited (i.e., that it does not contain any linear type variables).

Promotion

$$\frac{!\Gamma \vdash e : \sigma}{!\Gamma \vdash \text{promote } e :!\sigma}$$

**Food for thought:** Why does $\Gamma$ have to be unlimited? Hint: think about what would happen to a linear variable if you used it to prove that $e$ is the unlimited type $!\sigma$.

## 2.5 Times – $\otimes$

### 2.5.1 Introduction

The multiplicative conjunction is written $\otimes$. It allows us to prove something about two different things (with two separate type environments).

$\otimes$ Introduction

$$\frac{\Gamma \vdash e : \sigma \quad \Delta \vdash f : \tau}{\Gamma, \Delta \vdash \langle e, f \rangle : \sigma \otimes \tau}$$

### 2.5.2 Elimination

We can use the $\langle\rangle$ pair that we got from introducing a $\otimes$ type to provide types to two things at once.

Note that the $\Delta$ and $\Gamma$ are necessarily different from what they were in the introduction rule.

$\otimes$ Elimination

$$\frac{\Gamma \vdash e : \sigma \otimes \tau \quad \Delta, x : \sigma, y : \tau \vdash f : \rho}{\Gamma, \Delta \vdash \text{let } \langle x, y \rangle \ = e \text{ in } f : \rho}$$

## 2.6 With – &

### 2.6.1 Introduction

Whereas the multiplicative conjunction ($\otimes$) uses two different type environments to type check its two parts (and then subsequently uses both in one go), the additive conjunction & uses *one* type environment to prove something about two different things, then uses exactly one of them.

& Introduction

$$\frac{\Gamma \vdash e : \sigma \quad \Gamma \vdash f : \tau}{\Gamma \vdash [e, f] : \sigma \ \& \ \tau}$$

### 2.6.2 Elimination

We can only get one term out of eliminating an additive conjunction, since both of the terms about which we learned something in & introduction used the *same* type environment $\Gamma$. The nice part is that we get to choose.

& Elimination$_1$

$$\frac{\Gamma \vdash e : \sigma \ \& \ \tau}{\Gamma \vdash \text{fst } e : \sigma}$$

& Elimination$_2$

$$\frac{\Gamma \vdash e : \sigma \,\&\, \tau}{\Gamma \vdash \text{snd } e : \tau}$$

## 2.7 Plus – $\oplus$

### 2.7.1 Introduction

The additive disjunction, like the conjunction, necessitates that both of its pieces be proven from the same type environment $\Gamma$. Unlike the conjunction, however, both of its pieces are ultimately used to prove that some further statement has a third type $\rho$. One can draw an analog between this and branching.

$\oplus$ Introduction$_1$

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash \text{inl } e : \sigma \oplus \tau}$$

$\oplus$ Introduction$_2$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{inr } e : \sigma \oplus \tau}$$

### 2.7.2 Elimination

Given two statements of type $\sigma \oplus \tau$, we can eliminate them using "case."

$\oplus$ Elimination

$$\frac{\Gamma \vdash e : \sigma \oplus \tau \quad \Delta, x : \sigma \vdash f : \rho \quad \Delta, y : \tau \vdash g : \rho}{\Gamma, \Delta \vdash \text{case } e \text{ of inl } x \to f;\ \text{inr } y \to g : \rho}$$

## 2.8 Unit-y Stuff

We present the units after everything else, since they are (mostly) uninteresting. The exception is $\top$, which can consume a type environment.

An empty type environment is of type unit:

1 Introduction

$$\frac{}{\vdash \langle\rangle : 1}$$

And, if you can show that something is of type unit, you can replace it with an empty expression:

1 Elimination

$$\frac{\Gamma \vdash e : 1 \quad \Delta \vdash f : \sigma}{\Gamma, \Delta \vdash \text{let } \langle\rangle = e \text{ in } f : \sigma}$$

True and false work

Consume all of the $\Gamma$s.

$\top$ Introduction

$$\frac{}{\Gamma \vdash \text{true } \top}$$

0 Elimination

$$\frac{\Gamma \vdash e : 0}{\Gamma \vdash \text{false } e : \tau}$$

## 3 Typing Rules

Here are all of the typing rules in one chunk, presented in (approximate) order of interest.

$\multimap$ Introduction

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.e : \sigma \multimap \tau}$$

$\multimap$ Elimination

$$\frac{\Gamma \vdash e : \sigma \multimap \tau \quad \Delta \vdash f : \sigma}{\Gamma, \Delta \vdash e\ f : \tau}$$

Identity

$$\frac{}{x : \sigma \vdash x : \sigma}$$

Weakening

$$\frac{\Gamma \vdash e : \tau}{\Gamma, x :!\sigma \vdash \text{discard } x \text{ in } e : \tau}$$

Contraction

$$\frac{\Gamma, y :!\sigma, z :!\sigma \vdash e : \tau}{\Gamma, x :!\sigma \vdash \text{copy } x \text{ as } y, z \text{ in } e : \tau}$$

Dereliction

$$\frac{\Gamma \vdash e :!\sigma}{\Gamma \vdash \text{derelict } e : \sigma}$$

Promotion

$$\frac{!\Gamma \vdash e : \sigma}{!\Gamma \vdash \text{promote } e :!\sigma}$$

$\otimes$ Introduction

$$\frac{\Gamma \vdash e : \sigma \quad \Delta \vdash f : \tau}{\Gamma, \Delta \vdash \langle e, f \rangle : \sigma \otimes \tau}$$

$\otimes$ Elimination

$$\frac{\Gamma \vdash e : \sigma \otimes \tau \quad \Delta, x : \sigma, y : \tau \vdash f : \rho}{\Gamma, \Delta \vdash \text{let } \langle x, y \rangle \ = e \text{ in } f : \rho}$$

& Introduction

$$\frac{\Gamma \vdash e : \sigma \quad \Gamma \vdash f : \tau}{\Gamma \vdash [e, f] : \sigma \ \& \ \tau}$$

& Elimination$_1$

$$\frac{\Gamma \vdash e : \sigma \ \& \ \tau}{\Gamma \vdash \text{fst } e : \sigma}$$

& Elimination$_2$

$$\frac{\Gamma \vdash e : \sigma \ \& \ \tau}{\Gamma \vdash \text{snd } e : \tau}$$

$\oplus$ Introduction$_1$

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash \text{inl } e : \sigma \oplus \tau}$$

$\oplus$ Introduction$_2$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{inr } e : \sigma \oplus \tau}$$

$\oplus$ Elimination

$$\frac{\Gamma \vdash e : \sigma \oplus \tau \quad \Delta, x : \sigma \vdash f : \rho \quad \Delta, y : \tau \vdash g : \rho}{\Gamma, \Delta \vdash \text{case } e \text{ of inl } x \to f; \text{ inr } y \to g : \rho}$$

$\top$ Introduction

$$\frac{}{\Gamma \vdash \text{true } \top}$$

0 Elimination

$$\frac{\Gamma \vdash e : 0}{\Gamma \vdash \text{false } e : \tau}$$

1 Introduction

$$\frac{}{\vdash \langle\rangle : 1}$$

1 Elimination

$$\frac{\Gamma \vdash e : 1 \quad \Delta \vdash f : \sigma}{\Gamma, \Delta \vdash \text{let } \langle\rangle = e \text{ in } f : \sigma}$$