Graded Types - Part I

Theory and practice of linear types

Dominic Orchard, 24-28th July 2023, SPLV23

UNIVERSITY OF CAMBRIDGE
University of Kent

Thanks to the team

Harley Eades III

Michael Vollmer

Daniel Marshall

Jack Hughes

Benjamin Moon

Vilem Liepelt

Tori Vollmer

and Declan Barnes, James Dyer, Rowan Smith, Ed Brown

```
{n : N}
-> Fin n
-> Fin (n+1)
```

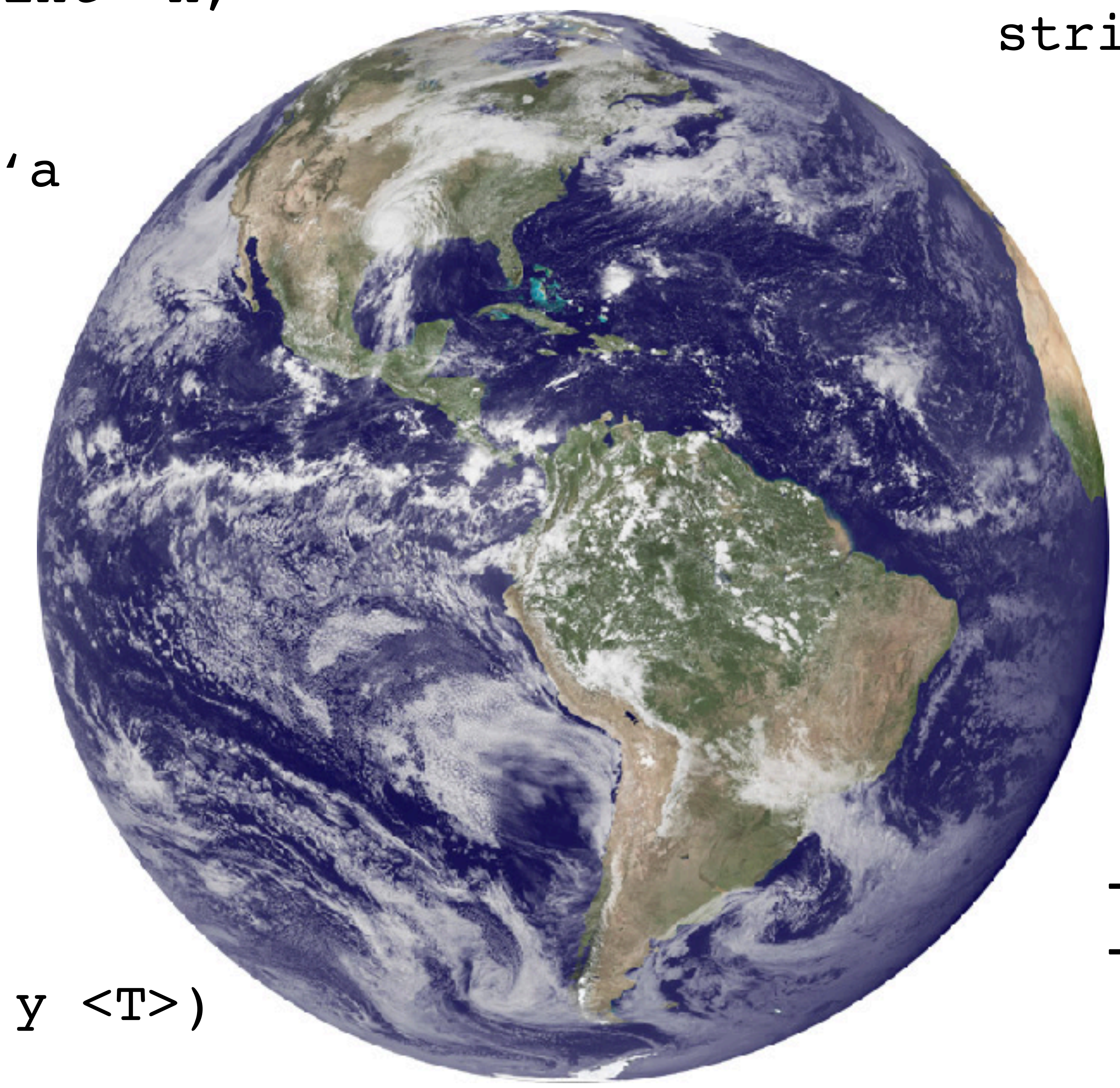
```
factorial(i : u64) -> u64
```

```
int* x;
```

```
string[] args
```

```
‘a * ‘b -> ‘a
```

```
REAL(KIND=8) pi;
```



```
set of 0..10
```

```
Vect n a
-> Vect m a
-> Vect (n+m) a
```

```
   float<‘u>
-> float<‘u ^ 2>
```

```
   [a]
-> (a -> Bool)
-> [a]
```

```
boolean eq(x <T>, y <T>)
```

```
'a => int -> 'a = <handler>
```

```
fib 2/1 :: (integer()) → integer()
```

Types for the “four Rs” of PL design

Reading

- Documentation

‘Riting

- Specification (intention)
- Synthesis

Reasoning

- Guarantee absence of some bugs
- Program properties (see ‘Free Theorems’)

Running

- Optimisations

Dominic A. Orchard:
The four Rs of programming language design. Onward! 2011: 157-162

Impure	Pure
`State Int String`	`String`
`IO String`	

Recall the S4 axioms for modal possibility ◊...

T $$A \to \Diamond A$$

4 $$\Diamond\Diamond A \to \Diamond A$$

K $$\Diamond(A \to B) \to \Diamond A \to \Diamond B$$

Monads as a possibility modality (Benton, Bierman, de Paiva)

Impure	Pure
`State Int String`	`String`
`IO String`	

Impure

Pure

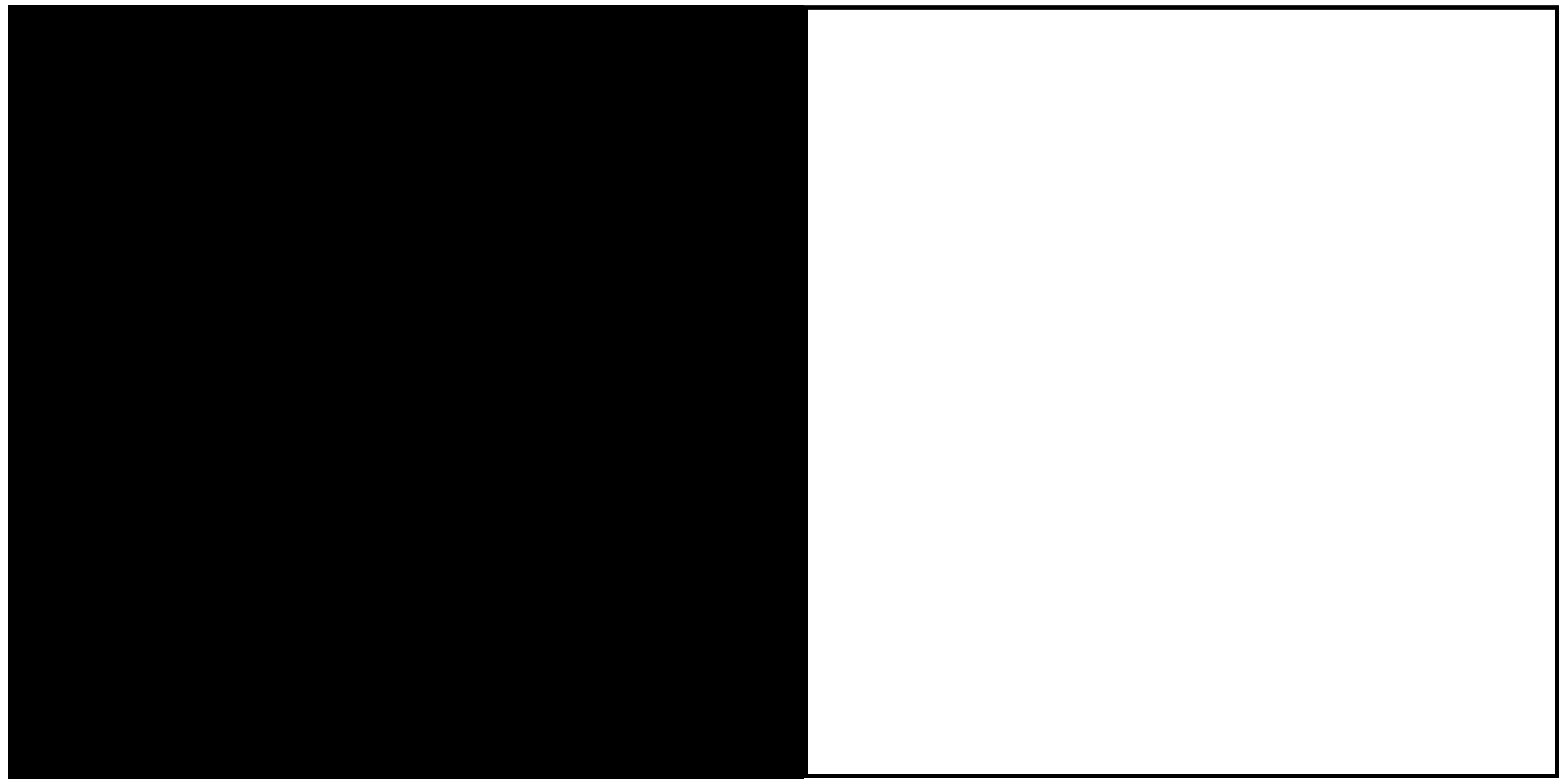
`State Int String`

`String`

Update Write Read Pure

View 1

Modal Type Analysis



$\Box A, \Diamond A$

$!A, \mathrm{M}\, A$

Graded Modal Type Analysis

$!_r A, \mathrm{M}_f\, A$

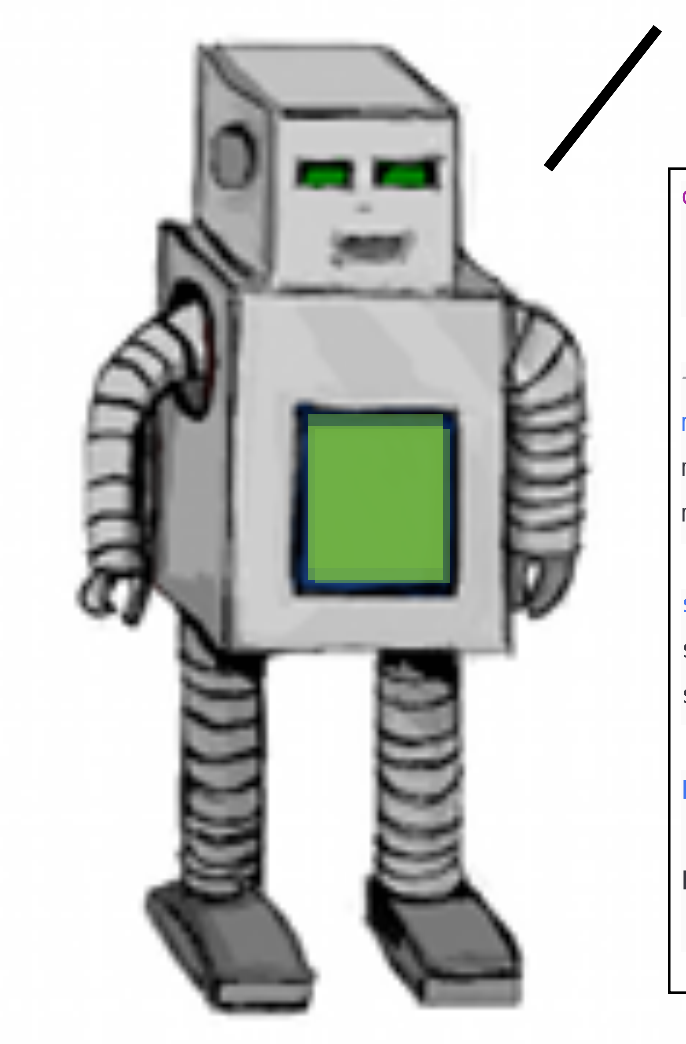
View 2

Intension

Extension

“how”

“what”

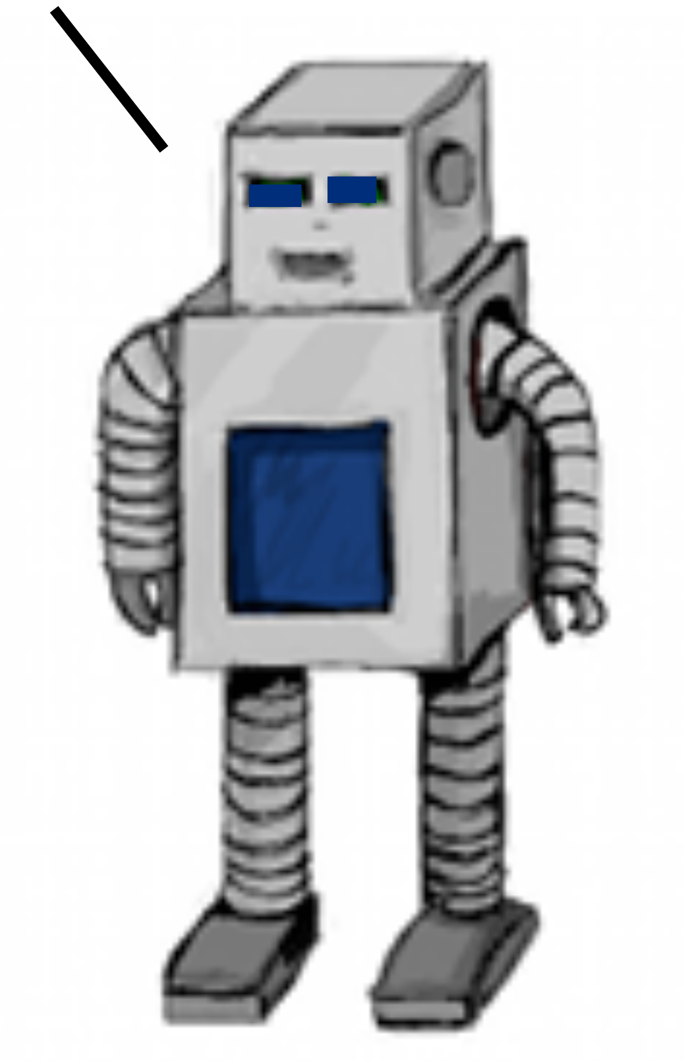


```
data Vec (n : Nat) (a : Type) where
    Nil : Vec 0 a;
    Cons : forall {n : Nat} . a -> Vec n a -> Vec (n+1) a

--- Map function
map : forall {a b : Type, n : Nat} . (a -> b) [n] -> Vec n a -> Vec n b
map [_] Nil = Nil;
map [f] (Cons x xs) = Cons (f x) (map [f] xs)

sequence : forall {n : Nat} . Vec n (() <{Stdout}>) -> () <{Stdout}>
sequence Nil = pure ();
sequence (Cons m xs) = let () <- m in sequence xs

printPerLine : forall {a : Type, n : Nat}
              . Vec n Char -> () <{Stdout}>
printPerLine xs =
 sequence (map [\x -> toStdout (stringAppend (showChar x) ("\n"))] xs)
```



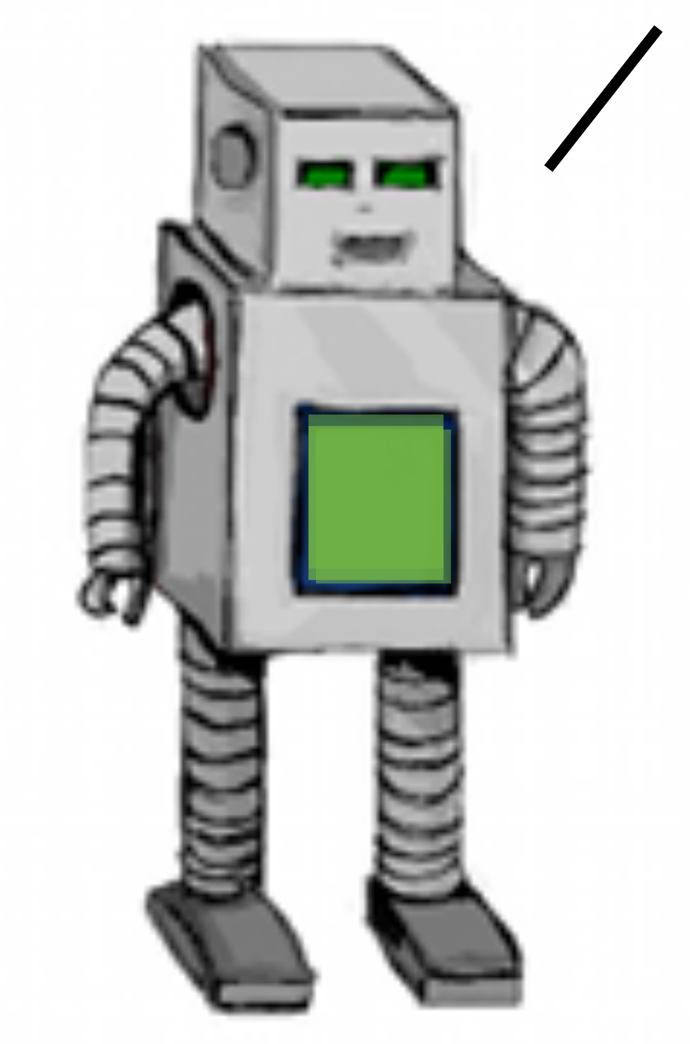
modalities
& grades

types

View 2

Intension

"*how*"



modalities
& grades

This course

how programs use data

how programs depend on context

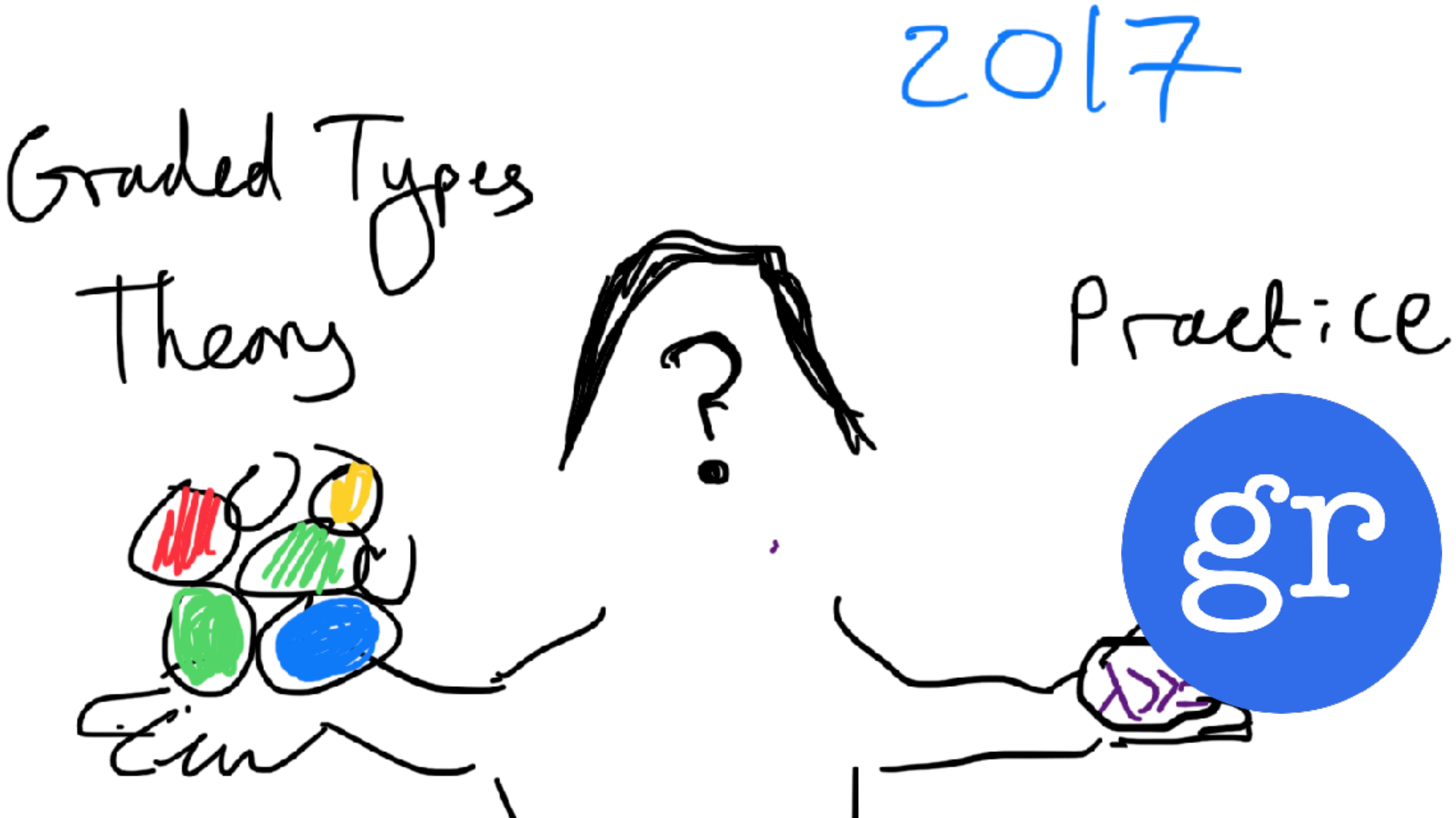
how programs change their context

Our route…

Theory & Practice

Our route…

Theory & Practice




definitely
ask me
ask me
SPLV '23

Session 1 - 2

Learning plan

Learn about linear types

Learn how a type system is formally specified

Specifically: linear types for the *lambda calculus*

See examples of linear programs in Granule

Learn about (a particular flavour of) modalities and graded modalities

Advice: externalise ‘known unknowns’

Things I don’t understand yet	Things I need to get better at	Things I’m getting the hang of

Materials and instructions

https://granule-project.github.io/splv23

Data

Infinitely copiable

Arbitrarily discardable

Universally unconstrained

Data

as a resource

Motivating Example

Problem

(Some) data acts as a resource

Ignoring this leads to bugs!

Solution

Capture resource constraints in types

Do this in a general, extensible way

Linear Logic and the non-linearity (necessity) modality (Girard, 1987)

"Resourceful" interpretation to logic: use exactly once

$$A \vdash A$$

$$A, B \nvdash A$$

$$A \nvdash A \wedge A$$

certainly / always / arbitrarily

but $A, !B \vdash A$

$$!A \vdash A \wedge A$$

Girard, Jean-Yves. "Linear logic." *Theoretical computer science* 50.1 (1987)

Linear types in Granule

(and solving the unsafe files problem)

File handling interface in Granule

like IO a in Haskell

```
openHandle     : IOMode -> String -> Handle <IO>

readChar       : Handle -> (Handle, Char) <IO>

writeChar      : Handle -> Char -> Handle <IO>

isEOF          : Handle -> (Handle, Bool) <IO>

closeHandle    : Handle -> () <IO>
```

Linear lambda calculus

Typing syntax and relation

Church syntax adds a type "signature"

$$t ::= x \mid \lambda(x : A) \,.\, t \mid t_1 \; t_2$$

Type syntax

$$A, B ::= A \multimap B$$

cf Haskell: `t -> t'`

$\mid$ `Int` $\mid$ `Bool` $\mid$ ....

In a full language we'd want more…

Typing lets us relate expressions to types, e.g.

$$\lambda(x : A) \,.\, x : A \multimap A$$

cf.

```
id :: a -> a
id = \x -> x
```

Quick exercise:

Q: What is the type of this lambda term?

$$\lambda(x : A).\lambda(y : B).x$$

A: $A \multimap B \multimap A$

Cf.:

```
const :: a -> b -> a
const x y = x
```

Q: What is the type of this lambda term?

$$\lambda(x : A).y$$

A: *It depends!*

Typing syntax and relation

Typing *judgement* with *assumptions* about variable types

$$y : B \vdash \lambda(x : A) \,.\, y : A \multimap B$$

Assumptions **Term** **Type**

Syntax of assumptions

$$\Gamma ::= \Gamma, x : A \mid \emptyset$$

Typing *judgement* form: $\Gamma \vdash t : A$

Typing rules

Defined inductively

Base case: $\dfrac{}{\textbf{conclusions}}$

Inductive step: $\dfrac{\textbf{premises (inductive hypotheses)}}{\textbf{conclusions}}$

$$\frac{}{x : A \vdash x : A}\ \text{var}$$

A term which is just one variable, has just one assumption

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x \,.\, t : A \multimap B}\ \text{abs}$$

Binding free variables

$$\frac{\Gamma \vdash t : A \multimap B \qquad \Delta \vdash t' : A}{\Gamma, \Delta \vdash t\, t' : B}\ \text{app}$$

Two sub terms have **different** contexts of assumptions

Example

$$\lambda(x : A).\lambda(y : A \to B).y\ x :$$???

Re-ordering in Γ

(abs) takes the "first" assumption:

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x \,.\, t : A \multimap B} \text{ abs}$$

What if we want to lambda bind *y* in the following?

$$\frac{y : A, x : B \vdash t : C}{x : B \vdash \lambda(y : A) \,.\, t : A \multimap C} \text{ ???}$$

✗

Allow reordering in Γ

Can be made explicit:

$$\frac{\Gamma, x : A, y : B, \Delta \vdash t : C}{\Gamma, y : B, x : A, \Delta \vdash t : C} \text{ exchange}$$

$$\frac{\dfrac{y : A, x : B \vdash t : C}{x : B, y : A \vdash t : C} \text{ exchange}}{x : B \vdash \lambda(y : A) \,.\, t : A \multimap C} \text{ abs}$$

✔

A non-example

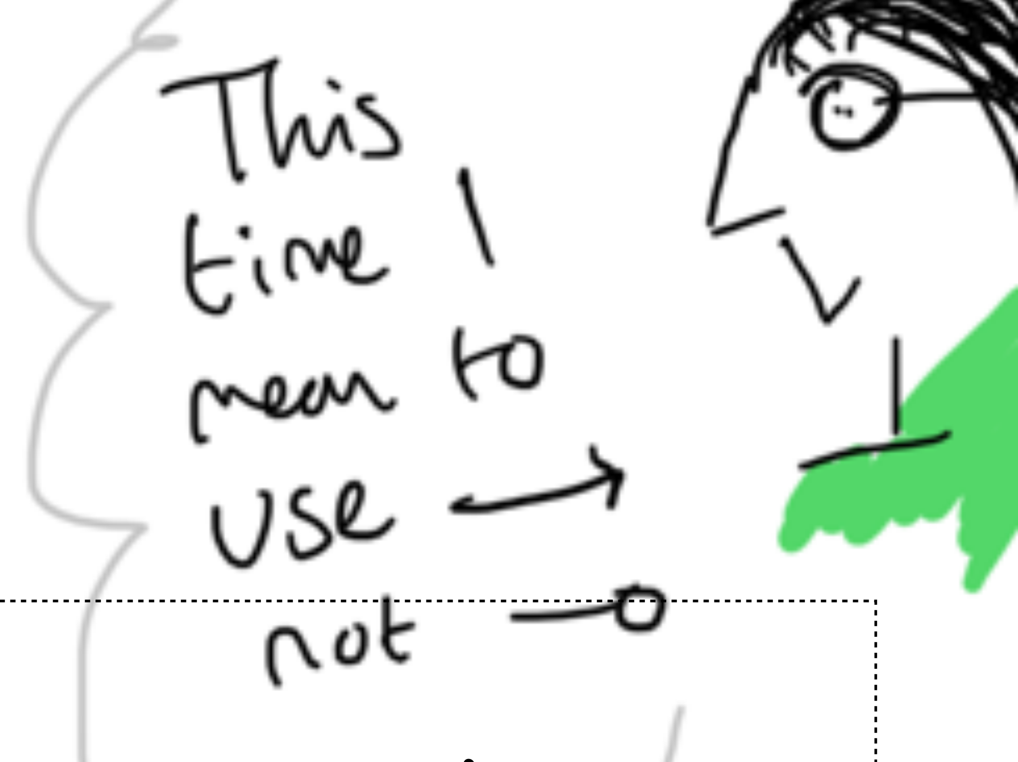
Can't use var rule

$$\dfrac{\dfrac{\dfrac{}{x : A, y : B \vdash x : A}\ ???}{x : A \vdash \lambda(y : B).\, x : B \to A}\ \textbf{abs}}{\emptyset \vdash \lambda(x : A).\, \lambda(y : B).\, x \ : A \to (B \to A)}\ \textbf{abs}$$

Ignoring variable y is dissallowed

Simple typing
(the usual state of affairs…)

Simple typing = Linear typing + weakening + contraction



$$\text{var}\ \frac{}{x : A \vdash x : A} \qquad \text{abs}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \text{app}\ \frac{\Gamma \vdash t : A \to B \quad \Delta \vdash t' : A}{\Gamma, \Delta \vdash t\, t' : B}$$

Linear λ-calculus typing

$$\text{exchange}\ \frac{\Gamma, x : A, y : B, \Gamma' \vdash t : A}{\Gamma, y : B, x : A, \Gamma' \vdash t : A}$$

$$\text{weaken}\ \frac{\Gamma \vdash t : A}{\Gamma, x : A' \vdash t : A} \qquad \text{contract}\ \frac{\Gamma, y : A', z : A' \vdash t : A}{\Gamma, x : A' \vdash t[x/z][x/y] : A}$$

Ignore variables

Reuse variables

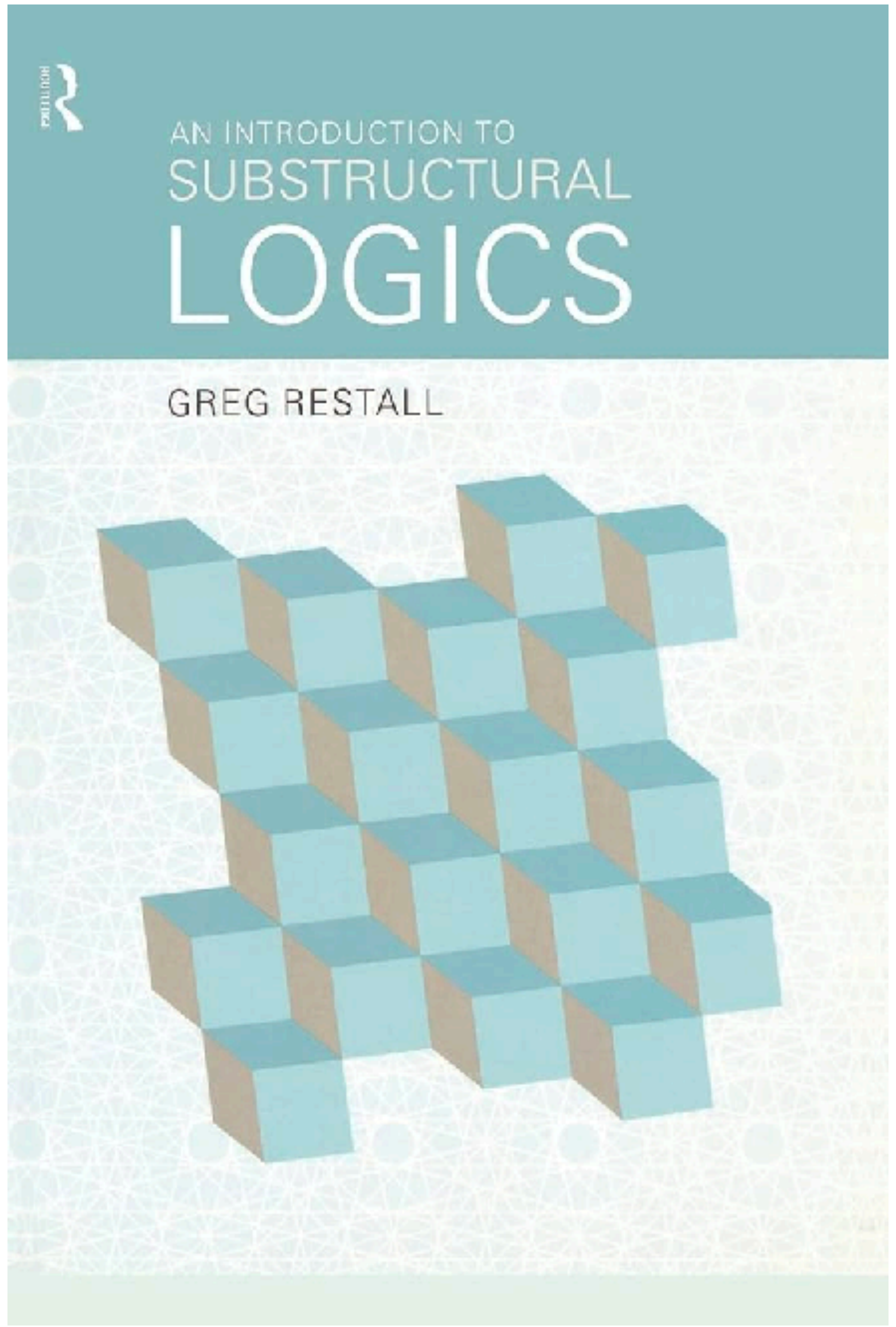
Structural rules

exchange
$$\frac{\Gamma, x : A, y : B, \Gamma' \vdash t : A}{\Gamma, y : B, x : A, \Gamma' \vdash t : A}$$

contract
$$\frac{\Gamma, y : A', z : A' \vdash t : A}{\Gamma, x : A' \vdash t[x/z][x/y] : A}$$

weaken
$$\frac{\Gamma \vdash t : A}{\Gamma, x : A' \vdash t : A}$$

Logics without one or more of these are called "substructural logics"



Weakening example

Couldn't do this in the linear system

$$\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A}\;\text{var}}{x : A, y : B \vdash x : A}\;\textbf{weaken}}{\dfrac{x : A \vdash \lambda(y : B).\, x : B \to A}{\emptyset \vdash \lambda(x : A).\, \lambda(y : B).\, x : A \to (B \to A)}\;\textbf{abs}}\;\textbf{abs}$$

Ignoring variable y

(4) $!A \multimap !!A$

(T) $!A \multimap A$

(K) $!(A \multimap B) \multimap !A \multimap !B$

Behaves like an (S4) modal $\Box$ + some more axioms

! modality — use any number of times

linear logic — use exactly once

Non-linearity modality
in
Granule

(Written postfix `a []` in Granule)

Typing syntax and relation

Extend syntax of types and typing assumptions

$$A, B ::= A \multimap B \mid \Box A$$

Non-linear value of type A

$$\Gamma ::= \emptyset \mid \Gamma, x : A \mid \Gamma, x : [A]$$

Non-linear variable assumption x of type A

(var), (abs), (app) stay the same…

…but we add weakening for non-linear assumptions

$$\frac{\Gamma \vdash t : B}{\Gamma, x : [A] \vdash t : B}\ \text{weak}$$

Linear types + modality

$$\frac{\Gamma_1 \vdash t : A \multimap B \quad \Gamma_2 \vdash t' : A}{\Gamma_1 + \Gamma_2 \vdash t\,t' : B}\ \text{app}$$

Adds contraction

Use anytime we need to combine contexts

$$(\Gamma, x : [A]) + (\Gamma', x : [A]) = (\Gamma + \Gamma'), x : [A]$$

$$\Gamma + (\Gamma', x : A) = (\Gamma + \Gamma'), x : A \ \ \textbf{if}\ x \notin |\Gamma|$$

$$\Gamma, x : A + \Gamma' = (\Gamma + \Gamma'), x : A \ \ \textbf{if}\ x \notin |\Gamma'|$$

Instead of...

$$\frac{\Gamma, x : [A], y : [A] \vdash t : B}{\Gamma, z : [A] \vdash t[z/x][z/y] : B}\ \text{contract}$$

…and syntax + rules for working with non-linearity

Treat a linear variable as non-linear:
(derelection)

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A] \vdash t : B}\ \text{der}$$

Non-linear results require non-linear variables
(promotion)

$$\frac{[\Gamma] \vdash t : B}{[\Gamma] \vdash [t] : \Box B}\ \Box_i$$

Composition (substitution) of non-linear value into non-linear variable

$$\frac{\Gamma \vdash t_1 : \Box A \qquad \Delta, x : [A] \vdash t_2 : B}{\Gamma + \Delta \vdash \text{let } [x] = t_1 \text{ in } t_2 : B}\ \Box_e$$

Is this logic "good"?

- "*All logics are invented, some are useful.*"

- Standard probes:

Lemma 1. (Admissibility of substitution). *Let* $\Delta \vdash t' : A$, *then:*

– *(Linear)* *If* $\Gamma, x : A, \Gamma' \vdash t : B$ *then* $\Gamma + \Delta + \Gamma' \vdash [t'/x]t : B$

– *(Modal)* *If* $\Gamma, x : [A], \Gamma' \vdash t : B$ *and* $[\Delta]$ *then* $\Gamma + \Delta + \Gamma' \vdash [t'/x]t : B$

Difficulty of getting this for S4 and ! explained in:

Dag Prawitz. 1965. Natural Deduction: A proof-theoretical study.

Philip Wadler. 1992. There's no substitute for linear logic. In 8th International Workshop on MFPS