Graded types and Algebraic Effects

Dominic Orchard

University of Kent

University of Cambridge

Institute of Computing for Climate Science

8th March 2024 - SREPLS-14

granule-project.github.io

With thanks to…

Harley Eades III

Michael Vollmer

Daniel Marshall

Jack Hughes

Benjamin Moon

Vilem Liepelt

Tori Vollmer

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

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

Recall the S4 axioms for modal possibility $\Diamond$...

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

`State Int String`

`IO String`

Pure

`String`

Impure

Pure

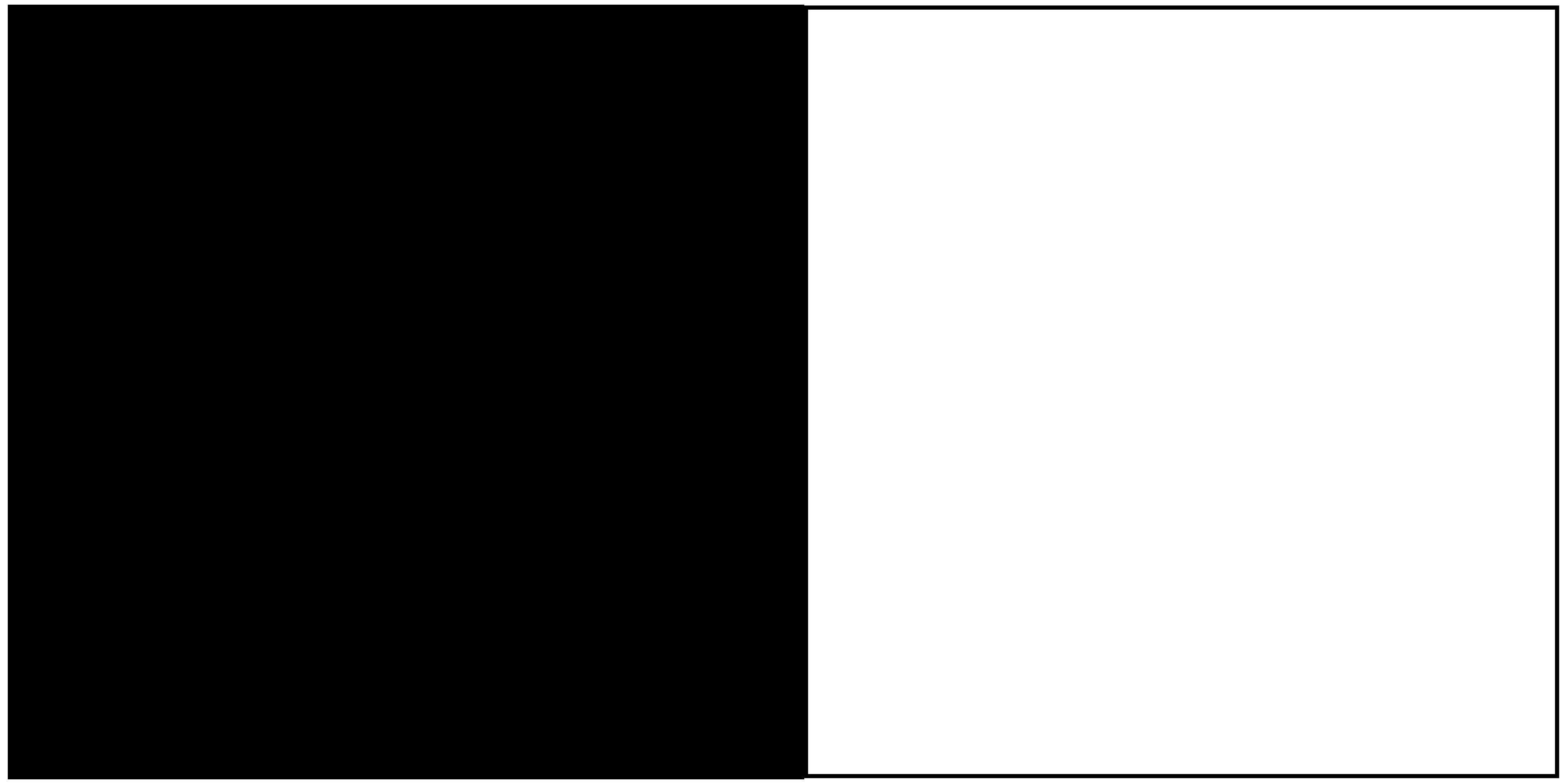
`State Int String`

`String`

Update Write Read Pure

View
1

Modal Type Analysis



Graded Modal Type Analysis

View 1

Modal Type Analysis

A | MA

Pure | Effectful

Graded Modal Type Analysis

$\Diamond_f A$

$f \in \mathcal{M}$

Monoid

Pure | Effectful

View 1

Modal Type Analysis

A	$!A$
linear	non-linear

Graded Modal Type Analysis

$$!_r A$$

linear — non-linear

$r \in \mathcal{R}$

semiring

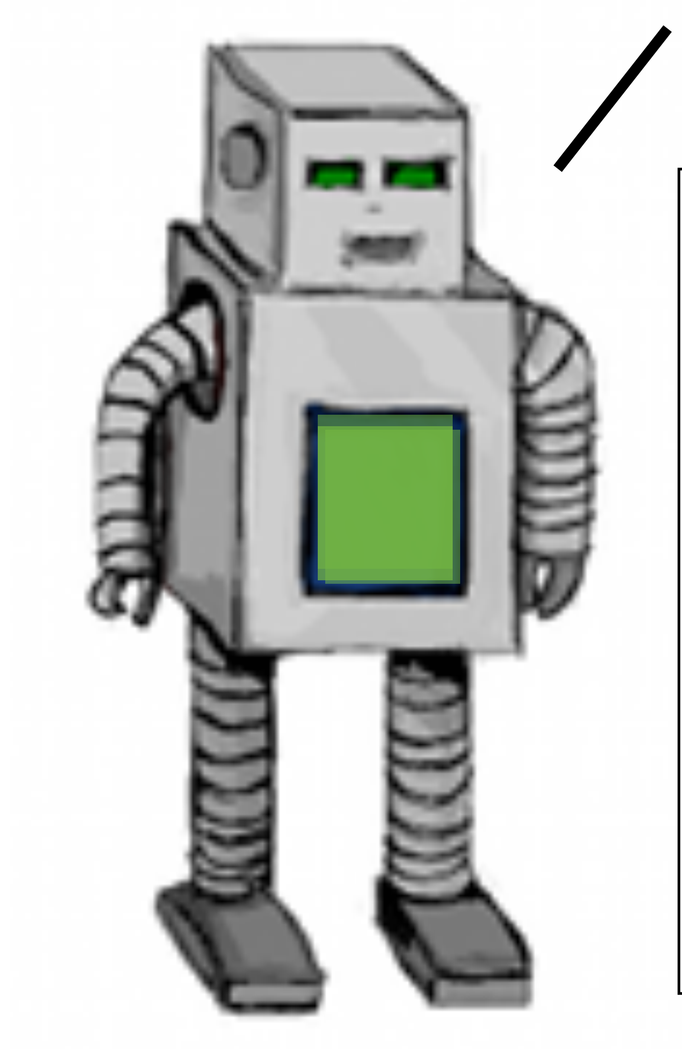
View 2

Intension

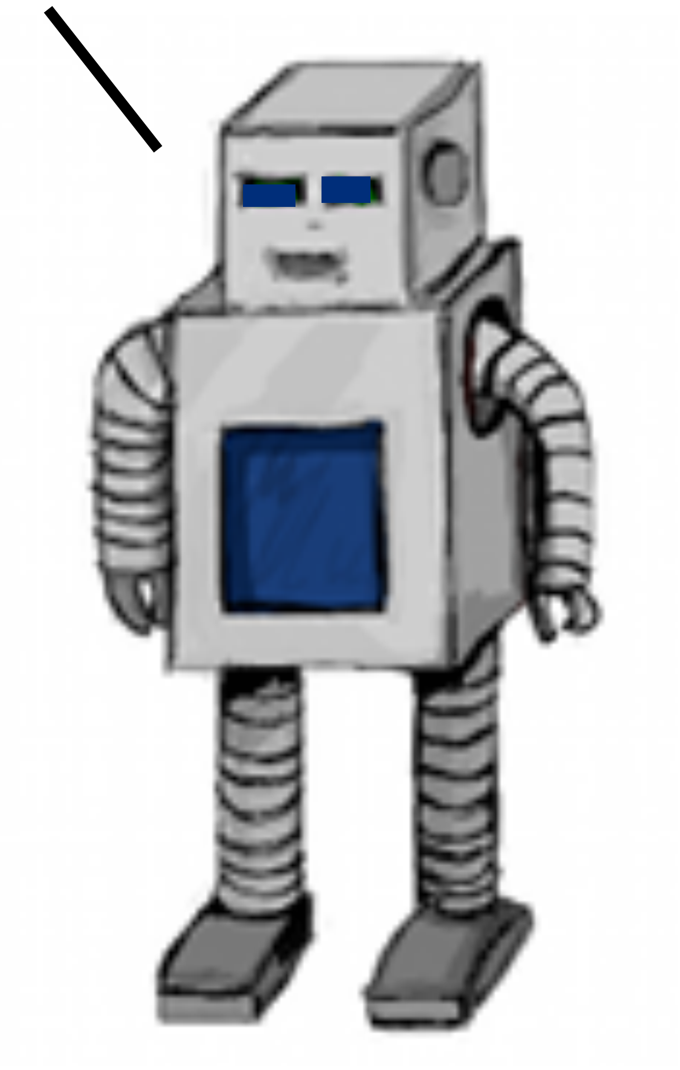
Extension

"how"

"what"



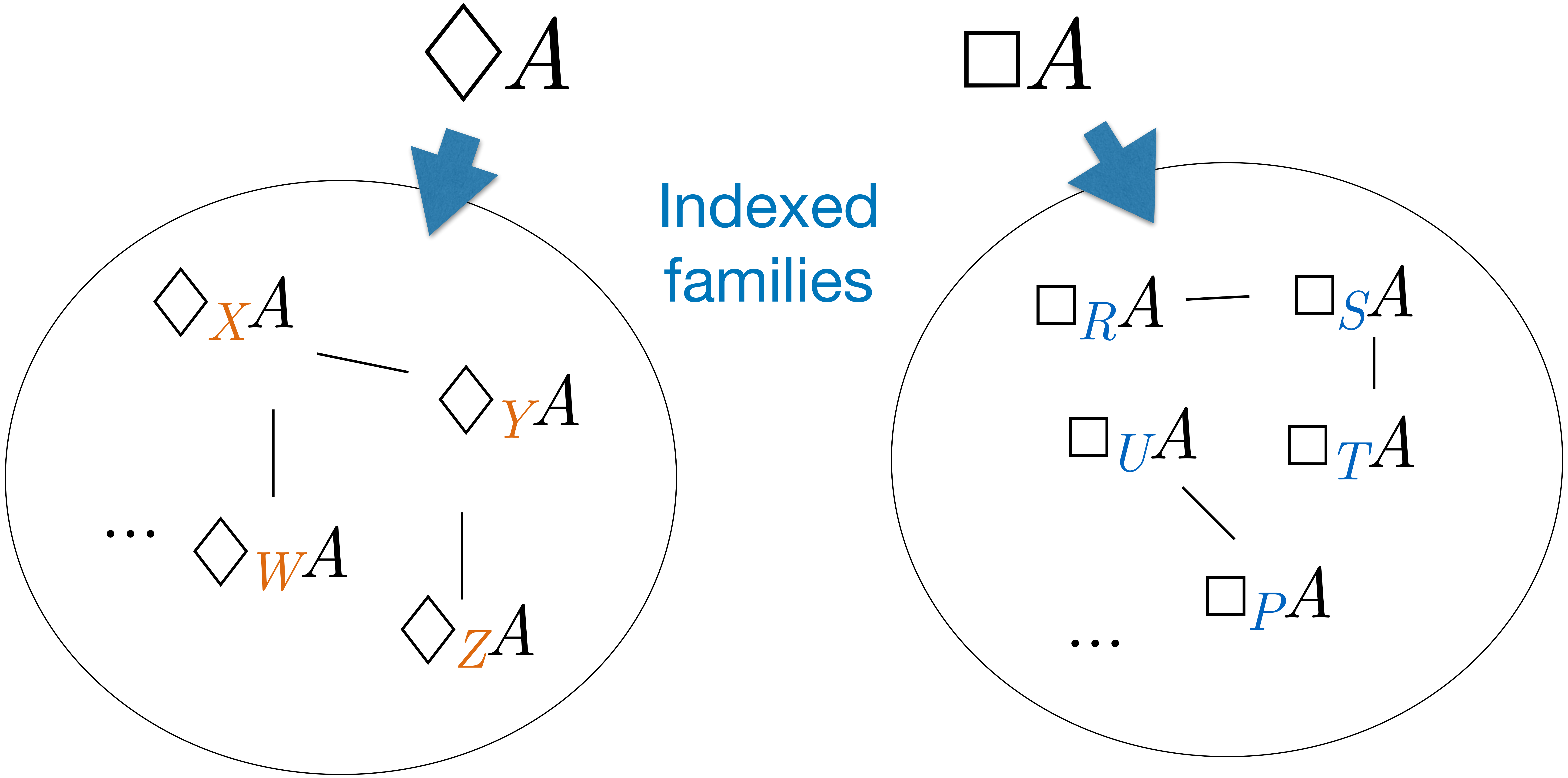
```
data Vec (n : Nat) (a : Type) where
    Nil : Vec 0 a;
    Cons : forall {n : Nat} . a -> Vec n a -> Vec (n+1) a

--- 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

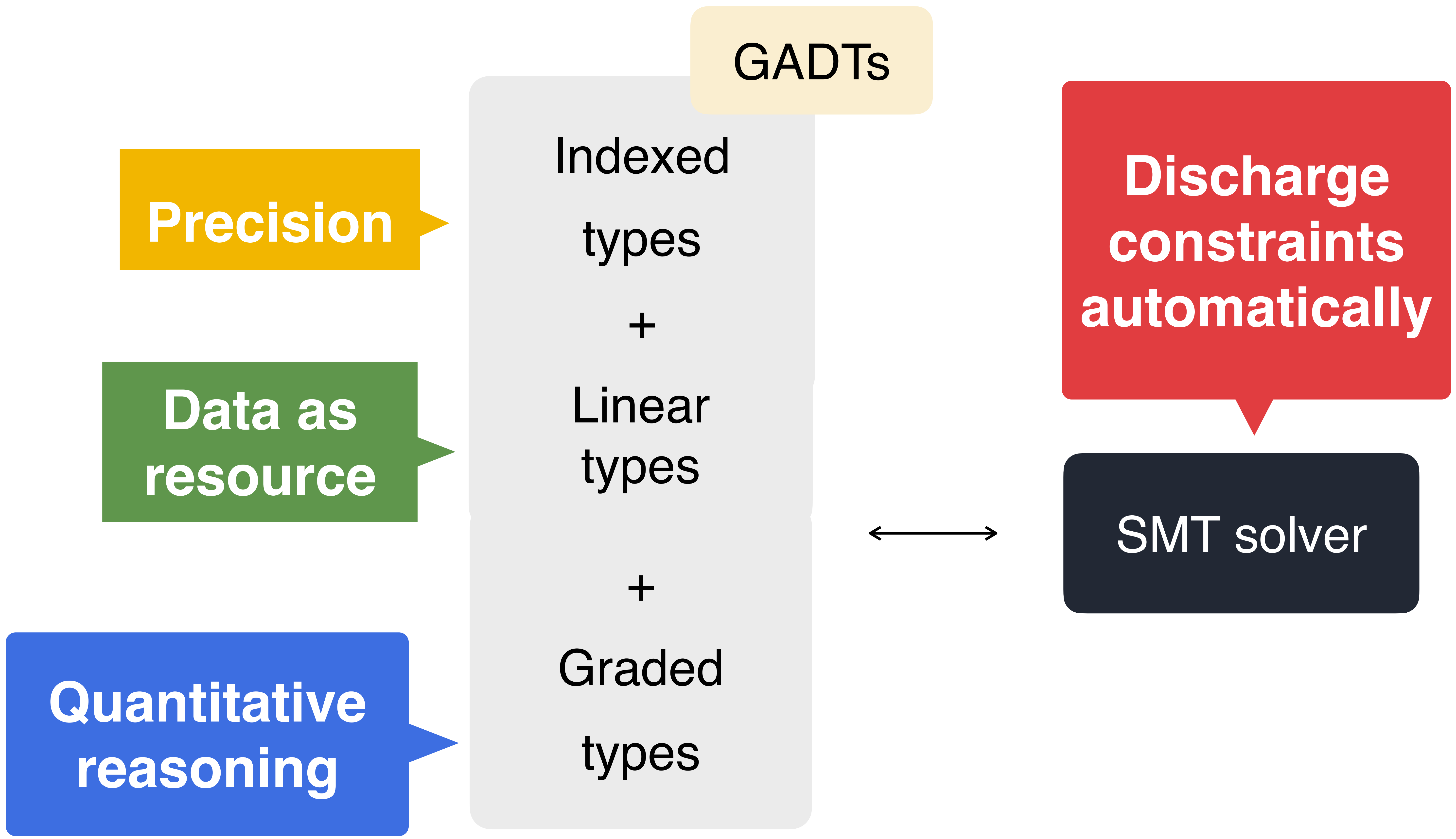
Graded modalities (informally)

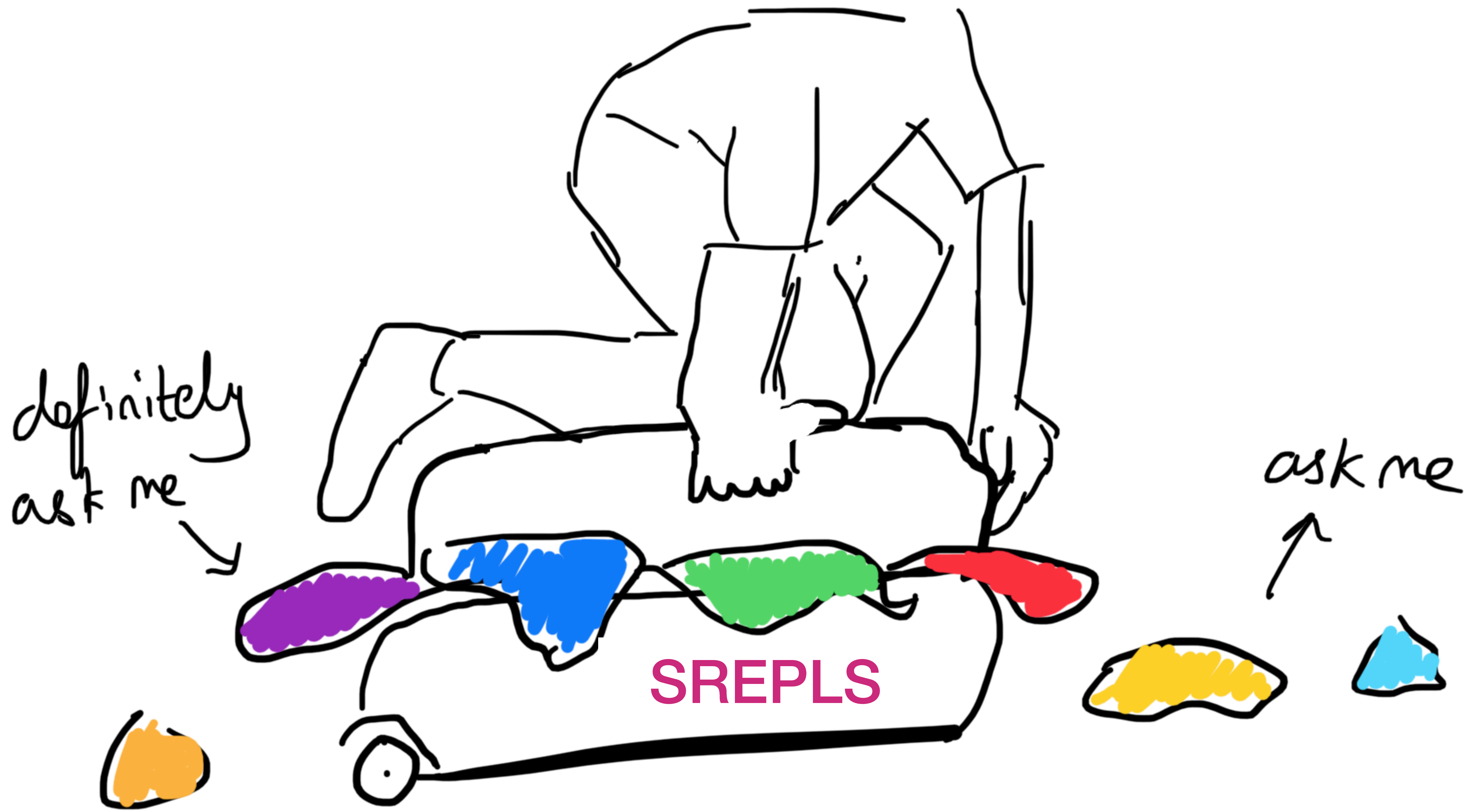


with structure

matching the shape of proofs/programs or a semantics

The granule language




definitely ask me
SREPLS
ask me

Linear types + graded modality

$r \in (\mathcal{R}, *, 1, +, 0)$ **is a semiring**

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

Non-linear value of type A

$$\Gamma ::= \varnothing \mid \Gamma, x : A \mid \Gamma, x : [A]_r$$

Non-linear variable x of type A

e.g.

$$\frac{x : [A]_2 \vdash (x, x) : A \otimes A}{\varnothing \vdash \lambda[x] . (x, x) : \Box_2 A \multimap A \otimes A}$$

(2013) Petricek, O, Mycroft - Coeffects: Unified Static Analysis of Context-Dependence

(2014) Ghica, Smith - Bounded linear types in a resource semiring

(2014) Brunel, Gaboardi, Mazza, Zdancewic - A Core Quantitative Coeffect Calculus

Linear types + graded modality

$r \in (\mathscr{R}, *, 1, +, 0)$ **is a semiring**

$$\frac{\Gamma \vdash t : B}{\Gamma, x : [A]_0 \vdash t : B}\ \text{weak} \qquad \frac{\Gamma_1 \vdash t : A \multimap B \quad \Gamma_2 \vdash t' : A}{\Gamma_1 + \Gamma_2 \vdash t\, t' : B}\ \text{app}$$

Use anytime we need to combine contexts

<u>contraction</u>

$$\begin{cases} \Gamma_1 + (\Gamma_2, x : A) = (\Gamma_1 + \Gamma_2), x : A \ \ \textbf{if } x \notin |\Gamma_1| \\ \Gamma_1, x : A + \Gamma_2 = (\Gamma_1 + \Gamma_2), x : A \ \ \textbf{if } x \notin |\Gamma_2| \end{cases}$$

$$(\Gamma_1, x : [A]_r) + (\Gamma_2, x : [A]_s) = (\Gamma_1 + \Gamma_2), x : [A]_{r+s}$$

Modal rule 1 - Dereliction

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma, x : [A]_1 \vdash t : B}\ \text{der}$$

Treat a linear variable as non-linear (dereliction)

Modal rule 2 - Promotion

$$\frac{[\Gamma] \vdash t : B}{r * [\Gamma] \vdash [t] : \Box_r B}\ \text{pr}$$

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

Modal rule 3 - Cut

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

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

A core quantitative coeffect calculus [Brunel et al.'14]

Semirings

```
Nat       : Semiring
Level     : Semiring                 {Private, Public} or {Hi, Lo}
Q         : Semiring
LNL       : Semiring                          {Zero, One, Many}
Cartesian : Semiring                                      {Any}
```

(see examples/scale.gr)

```
Set      : Type -> Semiring
SetOp    : Type -> Semiring
Ext      : Semiring -> Semiring
Interval : Semiring -> Semiring
```

(see examples/sets.gr)

(Ext $\mathscr{R}$ = $\mathscr{R} \cup \{\infty\}$)

$\_ \times \_$: Semiring -> Semiring -> Semiring

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

https://granule-project.github.io/docs/modules/Primitives.html

Modules

Top-level

- Primitives

- Bool
- Cake
- Choice
- Coffee
- Either
- Existential
- File
- Fin
- Fix
- Graph
- List
- Maybe
- Nat
- Parallel
- Prelude
- Result
- Stack
- State
- Vec

Built-in primitives

Meta-data

- **Description**: Built-in primitive definitions

Contents

Combining semirings

Two layers of grading…

```
f : (Vec … Patient) [0..1] -> …

f [Cons (Patient [city] [_]) ] = …
```

```
Public                  0..1
```

….generates the context

```
city : .[String]. ([0..1] × Public)
```

Some principles

- ~~No~~ Low magic
- Build things from theoretical elements
- Light syntax
- Interleave type checking and SMT
- CBV as (primary) semantics (but swappable in interpreter)

Graded possibility / monads

$f \in (X, \circledast, I)$ is a monoid

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \textbf{pure}\ e : \Diamond_I A} \qquad \frac{\Gamma_1 \vdash e_1 : \Diamond_f A \qquad \Gamma_2, x : A \vdash e_2 : \Diamond_g B}{\Gamma_1 + \Gamma_1 \vdash \textbf{let}\ x \leftarrow e_1\ \textbf{in}\ e_2 : \Diamond_{f \circledast g} A}$$

$\Diamond x\ A$ written in Granule as `A < x >`

Effect-set-graded possibility $(X, \circledast, I) = (\mathscr{P}(\text{IOlabels}), \cup, \emptyset)$

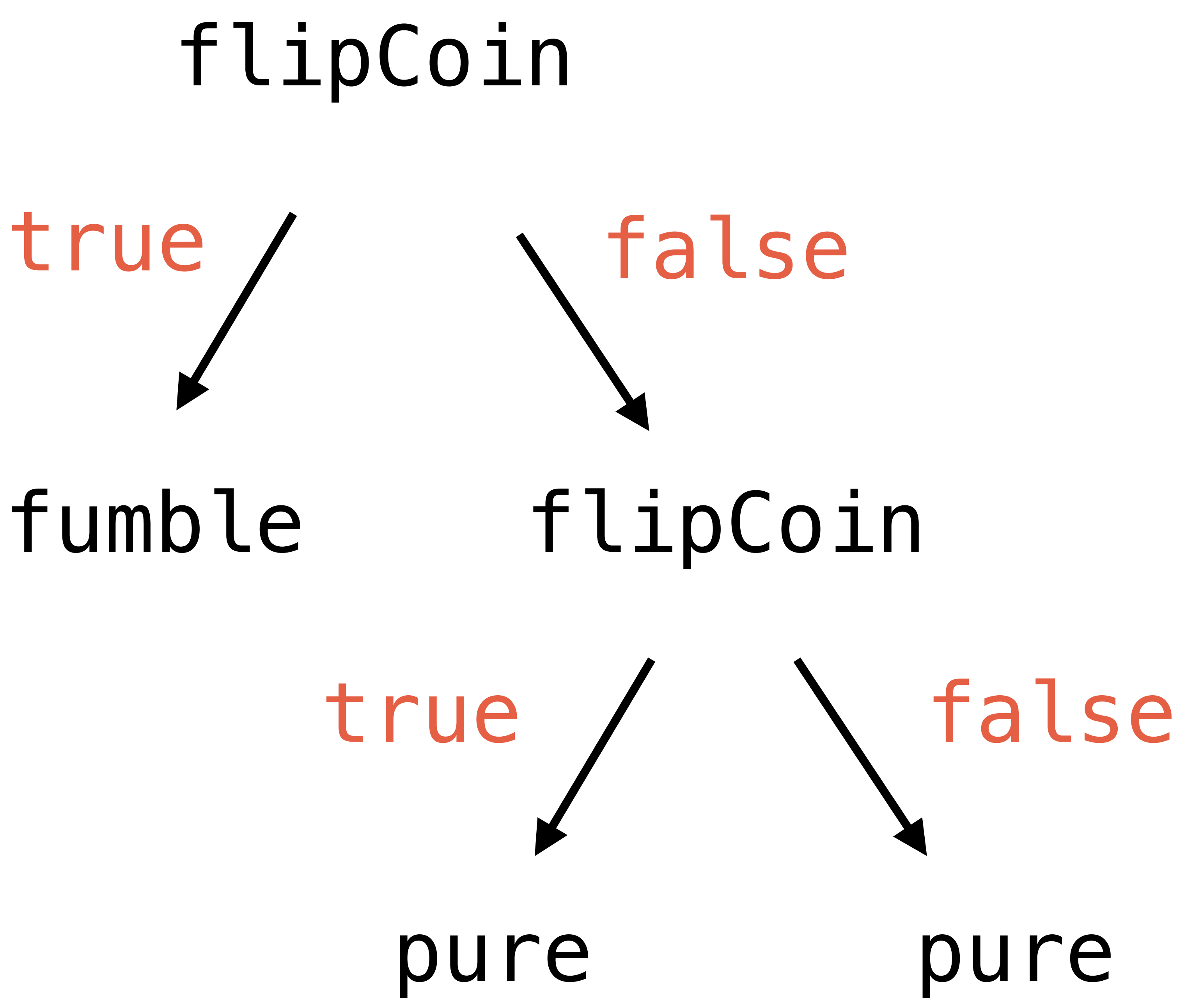
$\mathbb{N}$-graded possibility $(X, \circledast, I) = (\mathbb{N}, +, 0)$

Katsumata - Parametric effect monads and semantics of effect systems (2014)

O, Petricek, Mycroft - The semantic marriage of effects and monads (2014)

Algebraic effects & handlers

Computation tree



Represent via <u>free monad</u> over signature Σ

```
data GameOps r where
  FlipCoin : () -> (Bool -> r) -> GameOps r;
  Fumble   : () -> (Void -> r) -> GameOps r
```

```
comp : Free GameOps (Bool,Bool)
```

Handler to interpret (e.g., into a monad)

```
handle : (a + GameOps b -> b)
       -> Free GameOps a -> b
```

$$\texttt{Free GameOps a} \xrightarrow{\texttt{handle h}} \texttt{b}$$

Graded free monad

(For some signature functor Σ)

$$eff : \mathsf{Effect} \vdash \Sigma : eff \to \mathsf{Type} \to \mathsf{Type}$$

Constructors

$$\mathsf{pure} : A \multimap \Diamond_{\mathsf{Eff}_\Sigma(I)} A$$

$$\mathsf{impure} : \Sigma\, f\, (\Diamond_{\mathsf{Eff}_\Sigma(g)} A) \multimap \Diamond_{\mathsf{Eff}_\Sigma(f \circledast g)} A$$

(where $\mathsf{Eff} : \{eff : \mathsf{Effect}\} \to (\Sigma : eff \to \mathsf{Type} \to \mathsf{Type}) \to (f : eff) \to \mathsf{Type}$)

Generic effect operation

$$eff : \mathsf{Effect} \vdash \Sigma : eff \to \mathsf{Type} \to \mathsf{Type}$$

$$\frac{\Gamma \vdash t : (I \multimap \Box_r (O \multimap R)) \multimap \Sigma\; f\; R}{\Gamma \vdash \mathsf{call}\ \ t : I \multimap \Diamond_{\mathsf{Eff}(\Sigma, f)} O}$$

```
call : forall {eff : Effect, s : Semiring, grd : s, i : Type, o : Type, r : Type, sig : eff -> Type -> Type, e : eff}

     . (i -> (o -> r) [grd] -> sig e r)

    -> i -> o <Eff eff sig e>
```

Graded types ⋈ Algebraic effects and handlers

$$\frac{\Gamma \vdash t : \Box_{0..\omega}(\forall(e : \mathit{eff})\,.\,\Sigma\ e\ B \multimap B) \qquad \Gamma \vdash t' : A \multimap B}{\Gamma \vdash \mathsf{handle}\ t\ t' : \Diamond_{\mathsf{Eff}(\Sigma, f)} A \multimap B}$$

(together t and t' are a family of (Σ + -) algebras, for every e)

Graded types ⋈ Algebraic effects and handlers

$$\frac{\Gamma \vdash t : \Box_{0..\omega}(\forall(e : \mathit{eff}) . \Sigma\ e\ B \multimap B) \qquad \Gamma \vdash t' : A \multimap B}{\Gamma \vdash \mathsf{handle}\ t\ t' : \Diamond_{\mathsf{Eff}(\Sigma, f)} A \multimap B}$$

```
handle : forall {eff : Effect, sig : eff -> Type -> Type
       , a b : Type, e : eff}
```

Functor

```
       . (fmap : (forall {a b : Type} {l : eff}
               . (a -> b) [0..Inf] -> sig l a -> sig l b))) [0..Inf])
       --- ^ functoriality of sig
```

Handler

```
       -> (forall {l : eff} . sig l b -> b) [0..Inf]
       -> (a -> b)
       --- ^ (a + sig) - algebra
```

```
       -> a <Eff eff sig e>
       -> b
```

Graded algebras… (wip)

$$\frac{\Gamma \vdash t : \Box_{0..\omega}(\forall (e, f : \mathit{eff}) \,.\, \Sigma\ e\ (Bf) \multimap B(e \circledast f)) \qquad \Gamma \vdash t' : A \multimap BI}{\Gamma \vdash \mathsf{handleGr}\ t\ t' : \Diamond_{\mathsf{Eff}(\Sigma, f)} A \multimap Bf}$$

```
handleGr : forall {… b : Set labels -> Type}
        . (fmap  :…)

        -> (forall {l j : Set labels} . sig (b j) l -> b (j * l)) [0..Inf])
        -> (a -> b {})
        --- ^ (a + sig) - graded algebra

        -> a <Eff labels sig e>
        -> b e
```

Take home messages re effects

- **A.E.H. + graded linear types to control continuation use**
    - Fine-grained single-shot vs multi-shot control
- **Next steps:**
  - More implementation to enable *graded-algebras*
  - Layering

Graded types in Haskell (GHC 9)

```
{-# LANGUAGE LinearTypes #-}
```

cf. linear-base:

```
a %r -> b
```

Graded arrow

Linear
```
a %One -> b
```
```
a        -> b
a [Lin]  -> b
```

Unrestricted
```
a %Many -> b
```
```
a [Many] -> b
```

Graded modality

```
data Box r a where { Box :: a %r-> Box r a }
```

(2017) - Bernardy, Boespflug, Newton, Peyton Jones, Spiwak - Linear Haskell: practical linearity in a higher-order polymorphic language.

Graded-base coeffects

$$A ::= A \xrightarrow{r} B$$

$$\Delta ::= \emptyset \mid \Delta, x :_r A$$

2013 - Petricek, O, **Mycroft**
Coeffects: Unified Static Analysis of Context-Dependence

2014 - Petricek, O, **Mycroft**
Coeffects: a calculus of context-dependent computation.

2016 - McBride
I Got Plenty o' Nuttin'

2017 - Bernardy, Boespflug, Newton, Peyton Jones, Spiwack
Linear Haskell: practical linearity in a higher-order polymorphic language

2018 - Atkey
Syntax and Semantics of Quantitative Type Theory.

2021 - Abel, Bernardy
A unified view of modalities in type systems

Linear-base coeffects

$$A ::= A \multimap B \mid \Box_r A$$

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

2014 - Ghica, Smith
Bounded linear types in a resource semiring

2014 - Brunel, Gaboardi, Mazza, Zdancewic
A Core Quantitative Coeffect Calculus

2016 - Gaboardi, Katsumata, O, Breuvart, Uustalu
Combining effects & coeffects via grading

2019 - O, Liepelt, Eades
Quantitative program reasoning with graded modal types

....

+ a lot of work from the Granule project

language GradedBase

```
A % r -> B
```

LOPSTR 2020

Resourceful Prog
Graded L

Jack Hughes(✉) a

School of Computing, Univ
{joh6,d.a.orc

Abstract. Linear types provide
fying that some values must be use
modal types augments and refine
quantitative specification of data
vided by graded modal types appe
gram synthesis, where these additi
the search space of candidate pro
implementation challenges of a sy
does the synthesis algorithm effici
are satisfied throughout program g
this *resource management* problen

ESOP 2024

Program Synthesis from Graded Types

Jack Hughes[1] (✉) and Dominic Orchard[1,2]

[1] University of Kent, Canterbury, UK
[2] University of Cambridge, Cambridge, UK

Abstract. Graded type systems are a class of type system for fine-grained quantitative reasoning about data-flow in programs. Through the use of resource annotations (or *grades*), a programmer can express various program properties at the type level, reducing the number of typeable programs. These additional constraints on types lend themselves naturally to type-directed *program synthesis*, where this information can be exploited to constrain the search space of programs. We present a synthesis algorithm for a graded type system, where grades form an arbitrary pre-ordered semiring. Harnessing this grade information in synthesis is non-trivial, and we explore some of the issues involved in designing and implementing a resource-aware program synthesis tool. In our evaluation we show that by harnessing grades in synthesis, the majority of our benchmark programs (many of which involve recursive functions over recursive ADTs) require less exploration of the synthesis search space than a purely type-driven approach and with fewer needed input-output examples. This *type-and-graded-directed* approach is demonstrated for the

ESOP 2022

Check for updates

Linearity and Uniquer

Daniel Marshall[1] (✉) , Michael

[1] University of
{dm635,m.vollmer,
[2] University

Abstract. Substructural type s
cause they allow for a resourcef
used to rule out various softwa
nally taking hold in modern prog
roughly based on Girard's linear
arrows, Clean has uniqueness ty
at most a single reference to the
system for guaranteeing memor
of resourceful type systems, the
of their relative strengths and w
frameworks can be unified. Ther
earity and uniqueness are essent
one another, or somewhere in b
lationship between these two w
building on two distinct bodies o
and advantageous to have both li

OOPSLA 2024

Functional Ownership through Fractional Uniqueness

DANIEL MARSHALL, University of Kent, United Kingdom
DOMINIC ORCHARD, University of Kent, United Kingdom and University of Cambridge, United Kingdom

Ownership and borrowing systems, designed to enforce safe memory management without the need for garbage collection, have been brought to the fore by the Rust programming language. Rust also aims to bring some guarantees offered by functional programming into the realm of performant systems code, but the type system is largely separate from the ownership model, with type and borrow checking happening in separate compilation phases. Recent models such as RustBelt and Oxide aim to formalise Rust in depth, but there is less focus on integrating the basic ideas into more traditional type systems. An approach designed to expose an essential core for ownership and borrowing would open the door for functional languages to borrow concepts found in Rust and other ownership frameworks, so that more programmers can enjoy their benefits.

One strategy for managing memory in a functional setting is through *uniqueness types*, but these offer a coarse-grained view: either a value has exactly one reference, and can be mutated safely, or it cannot, since other references may exist. Recent work demonstrates that *linear* and *uniqueness* types can be combined in a single system to offer restrictions on program behaviour and guarantees about memory usage. We develop this connection further, showing that just as *graded* type systems like those of Granule and Idris generalise linearity, a Rust-like *ownership* model arises as a graded generalisation of uniqueness. We combine fractional permissions with grading to give the first account of ownership and borrowing that smoothly integrates into a standard type system alongside linearity and graded types, and extend Granule accordingly with these ideas.

Uniqueness and Linearity together

Unique $*a$

↓ sharing

Cartesian $!a$

↓ dereliction

Linear a

Unique values have only own “owner”

Cartesian values under **comonadic !** modality (Abitrary use)

Linear values must be used once

Graded uniqueness (*the third flavour…*)

$$*A$$

$$\&_* A \qquad \&_p A \qquad \&_1 A$$

Uniquely owned A

$p \in (0,1) \subset \mathbb{Q}$

Immutable borrow

Mutable borrow

+ primitives for borrowing, mutable borrowing by splitting/joining lifetimes

e.g. split : $\&_p A \multimap \&_{\frac{p}{2}} A \otimes \&_{\frac{p}{2}} A$

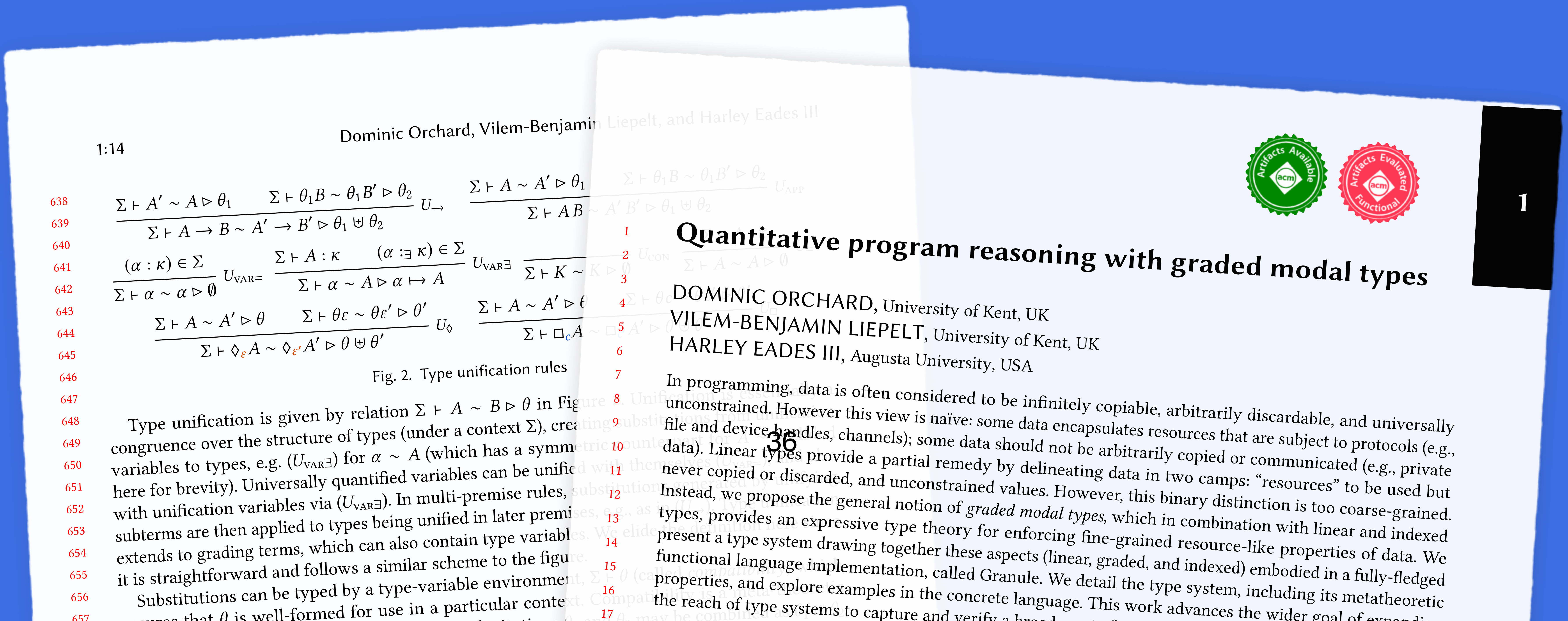
(follow Daniel Marshall's work -> https://starsandspira.ls/)

Download and play!

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

Some more resources here from recent summer school material

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



Shout out to many others working on (/ who have worked) on graded types!

- Andreas Abel
- Jean-Philippe Bernardy
- Shin-ya Katsumata
- Dylan McDermott
- Tarmo Uustalu
- Riccardo Biancinni
- Frank Pfenning
- Stephanie Weirich
- Marco Gaboardi
- Flavien Bruevart
- Francesco Dagnino
- Paola Giannini
- Elena Zucca
- Bob Atkey
- James Wood
- Dan Ghica
- Conor McBride
- AND MORE