Linearity and Uniqueness:

an *Entente Cordiale*

Thursday 7th April

ESOP 2022

Daniel Marshall

Michael Vollmer

Dominic Orchard

University of Kent

gr

Some data is unrestricted.

but...

some data is **resourceful.**

~~infinitely copiable~~

~~arbitrarily discardable~~

~~universally mutable~~

Linear types are like **cakes**.

You can only eat them **once**.

You **have** to eat them.

```
{-# LANGUAGE LinearTypes #-}

desire :: Cake ⊸ (Happy, Cake)
desire cake = (eat cake, have cake)
```

Linearity in Granule!

Unique types are like **coffee.**

A fresh coffee has just been poured.

We can sip our coffee, **but...**

then it is **no longer** fresh!

```
share :: *Coffee → (Awake, *Coffee)
share coffee = (drink coffee, keep coffee)
```

"But what's the difference?"

Clean is a commercially developed, pure functional programming language. It uses *uniqueness types* (Barendsen and Smetsers, 1993), which are a variant of linear types, and strictness annotations (Nöcker and Smetsers, 1993) to help

Linear Haskell

Practical Linearity in a Higher-Order Polymorphic Language

JEAN-PHILIPPE BERNARDY, University of Gothenburg, Sweden
MATHIEU BOESPFLUG, Tweag I/O, France
RYAN R. NEWTON, Indiana University, USA
SIMON PEYTON JONES, Microsoft Research, UK
ARNAUD SPIWACK, Tweag I/O, France

Linear type systems have a long and storied history, but not a clear p
guages such as OCaml or Haskell. In this paper, we study a linea
in mind: backwards-compatibility and code reuse acros
e benefits of linear types permeate conventio
to linear and non-linear counterparts, we inste
ceive inputs from linearly-bound values, but can
rate the efficacy of our linear type system — bo
lementation and how streamlined it makes it t
type system in GHC, the leading Haskell compil
s: mutable data with pure interfaces; and enforc

Linear types and uniqueness types are, at their core, dual: whereas a linear type is a contract that a function uses its argument exactly once even if the call's context can share a linear argument as many times as it pleases, a uniqueness type ensures that the argument of a function is not used anywhere else in the expression's context even if the callee can work with the argument as it pleases.

Unique types **guarantee** that a value has never been duplicated in the **past.**

Linear types **restrict** a value from ever being duplicated (or discarded) in the **future.**

A history of **linearity** and **uniqueness** (abridged)

Linear logic
Girard (1987)

Linear types can change the world!
Wadler (1990)

Linear Haskell: Practical linearity in a higher-order polymorphic language
Bernardy, Boespflug, Newton, Peyton Jones, Spiwack (2018)

Uniqueness logic
Harrington (2006)

Uniqueness typing simplified
de Vries, Plasmeijer, Abrahamson (2008)

Guaranteeing safe destructive updates through a type system with uniqueness information for graphs
Smetsers, Barendsen, van Eekelen, Plasmeijer (1994)

Uniqueness logic

Structural rules are restricted as in linear logic.

- modality allows for contraction and weakening, as with ! from linear logic

Unique **Cartesian** **Linear**

"How can we use both?"

Unique *a

borrowing

Cartesian !a

dereliction

Linear a

Unique values under an **additional** * modality.

Cartesian values under a **comonadic** ! modality.

Linear values as the base.

$$\frac{\Gamma \vdash t : {*}A}{\Gamma \vdash \&t : !A}$$ **borrow**

The ! modality acts as a **relative monad** over *.

$$\frac{\Gamma_1 \vdash t_1 : !A \quad \Gamma_2, x : {*}A \vdash t_2 : !B}{\Gamma_1 + \Gamma_2 \vdash \text{copy } t_1 \text{ as } x \text{ in } t_2 : !B}$$ **copy**

Unit laws and associativity hold!

$$\frac{\Gamma \vdash {*}A}{\Gamma \vdash !A}$$ (return)

$$\frac{\Gamma_1 \vdash !A \quad \Gamma_2, x : {*}A \vdash !B}{\Gamma_1 + \Gamma_2 \vdash !B}$$ (bind)

Metatheory

Operational model:

- Based on heaps
- Call-by-name semantics
- Details in the paper!

Key properties:

Resource conservation
or: if a term is linear then it will not be duplicated/discarded

Uniqueness preservation
or: if a term is unique then it does not have multiple references

"And is there an implementation?"

Yes – in **Granule!**

- Already has a **linear** base.
- Already represents **!** as a **coeffect** modality (dual to **effects**).
- Represent ***** as a **third** flavour of modality (called a **guarantee**).

https://granule-project.github.io

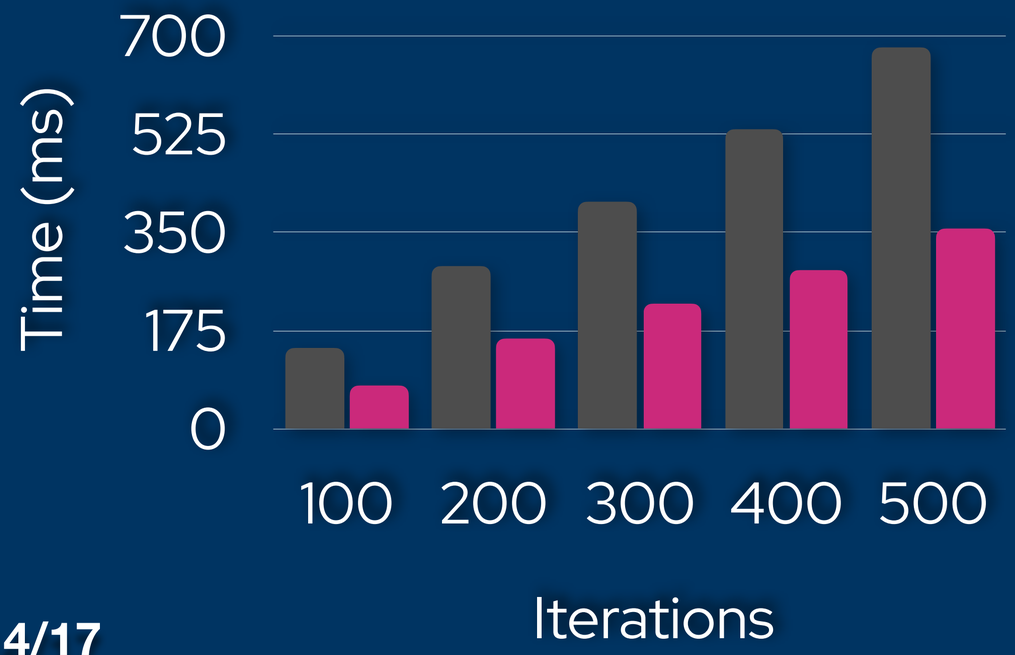
```
sip : *Coffee → (Awake, !Coffee)
sip fresh = let !coffee = &fresh in (drink coffee, [keep coffee])
```

Uniqueness in Granule!

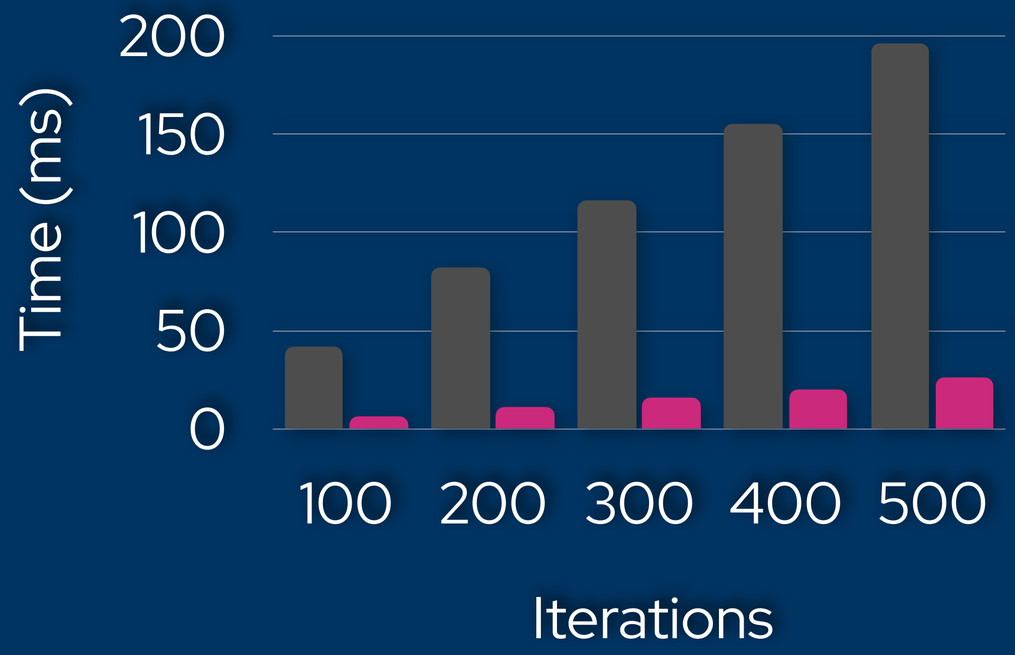
We evaluate our system using **unique arrays.**

It's safe to mutate these in place, so uniqueness gives us some **performance benefits**!

Overall runtime

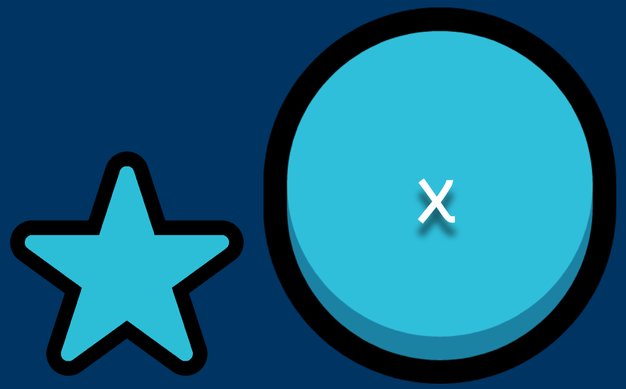


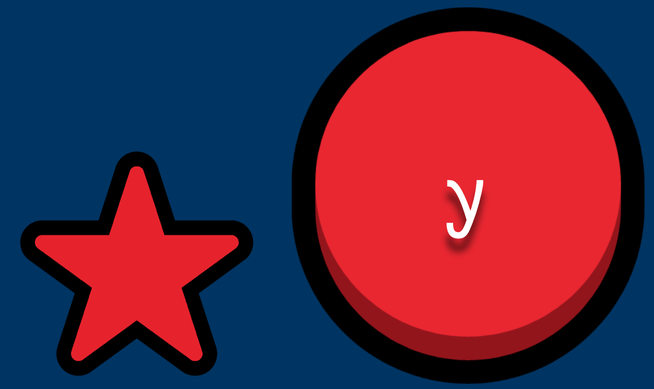
Garbage collection overhead



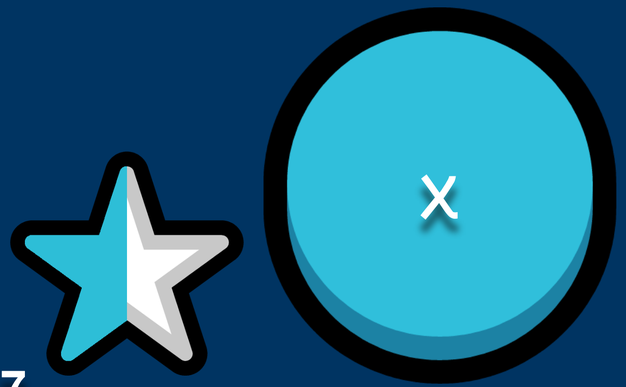
Future work: **ownership?**

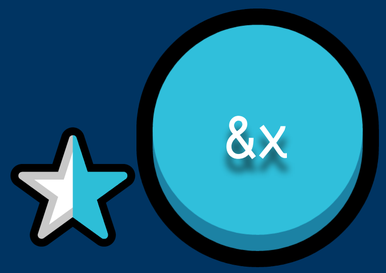
Ownership can be modelled via **capabilities.**

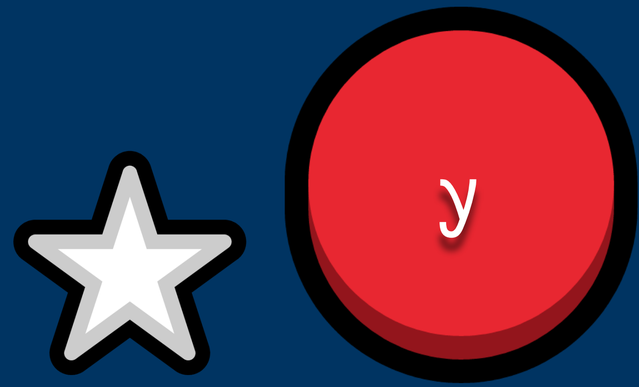




Borrowing splits capabilities into **fractions.**







Future work: **ownership?**

Linear types can be generalised to allow for **quantitative** restrictions.

$!_2\ a \to a \times a$

*(similar to **bounded linear logic**)*

Unique types can be generalised to allow for **fractional** guarantees.

$*_{1/2}\ a \times *_{1/2}\ a \leftrightarrow *a$

*(similar to **fractional permissions**)*

Thanks for listening!

In summary:

Formalised the relationship between linearity and uniqueness

Developed a unified calculus and proved key properties

Implemented unique arrays and demonstrated efficiency gains

Next, understand ownership through fractional permissions

https://granule-project.github.io