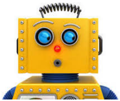

# When programs have to watch paint dry

Danel Ahman

Faculty of Mathematics and Physics
University of Ljubljana

Theory Seminar @ TTÜ, 17.11.2022

# Resources are important in programming!

# **Resources are important in programming!**

- Much of existing work has focussed on **how** resources are used
  - **linear types** to **avoid discarding and dupl.** (of file handles)

$$A, B ::= \ \dots \mid A \otimes B \mid A \multimap B \mid \dots$$

  - **separation logics** for **framing and anti-aliasing** of memory

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}\ \textsc{Frame}$$

  - **session types**, **coeffect systems**, **runners of (alg.) effs.**, ...

# **Resources** are important in programming!

- Much of existing work has focussed on **how** resources are used
  - **linear types** to **avoid discarding and dupl.** (of file handles)

$$A, B ::= \ \ldots \mid A \otimes B \mid A \multimap B \mid \ldots$$

  - **separation logics** for **framing and anti-aliasing** of memory

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}\ \textsc{Frame}$$

  - **session types**, **coeffect systems**, **runners of (alg.) effs.**, ...

- We instead focus on **when** resources are used
  - values might become **usable only after some time**
  - want to **avoid unnecessary blocking and idle waiting**
  - but also **start work as soon as resources become available**

# Temporal resources are also important!

- Consider **controlling robot arms** on a production line:

```
let (body', left-door', right-door') =
  paint (body, left-door, right-door) in

assemble (body', left-door', right-door')
```

# Temporal resources are also important!

- Consider **controlling robot arms** on a production line:

```
let (body', left-door', right-door') =
  paint (body, left-door, right-door) in
                                          ← τ_dry time needs to pass
assemble (body', left-door', right-door')
```

- **Correctness** relies on the **parts given enough time to dry**

# Temporal resources are also important!

- Consider **controlling robot arms** on a production line:

```
let (body', left-door', right-door') =
  paint (body, left-door, right-door) in
                                          ← τ_dry time needs to pass
assemble (body', left-door', right-door')
```

- **Correctness** relies on the **parts given enough time to dry**

  **(a)** a **scheduler** could **dynamically block execution**, or

  **(b)** a **compiler** could **insert enough time delay** between op. calls

# Temporal resources are also important!

- Consider **controlling robot arms** on a production line:

```
let (body', left-door', right-door') =
  paint (body, left-door, right-door) in
                                        ← τ_dry time needs to pass
assemble (body', left-door', right-door')
```

- **Correctness** relies on the **parts given enough time to dry**

  **(a)** a **scheduler** could **dynamically block execution**, or

  **(b)** a **compiler** could **insert enough time delay** between op. calls

- But **how to reason** about the result being **temporally correct**?

  - we focus on the kinds of **code emitted by (b)**, or **written directly** when full control and predictability is important
  - we develop **type-based means** for reasoning about its correctness

# Temporal resources are also important!

- **Not just about assembling (car) parts:**
  - **interrupt-handling** (in low-level embedded IoT code)

    
    

    - handler code should run in predictable time
    - should account for fetching any necessary resources
    - make use of as many of the limited MCU cycles as possible
    - (the receiving end of op. calls and interrupts from sensors)
  - **asynchronous programming** (via async/await, futures, . . . )
    - want time guarantees about when async. comps. come back
    - to know when it is safe to synchronise (for minimal blocking)
  - ...

# Today's plan

- **Temporal resources** via **time-graded modal types**
  - enforcing temporal correctness for the robot arms example
- A **core calculus** for safe programming with temporal resources
  - Fitch-style **time-graded modal types** (for temporal resources)
  - **temporally aware graded algebraic effects** (for time passage)
  - **temporally aware effect handlers** (for user-defined effects)
- A sound **denotational semantics** justifying the proposed design
  - **adjoint strong monoidal functors** (for modalities)
  - **$[-]$-strong time-graded monad** (for effectful computations)
  - a **presheaf example** (for concreteness and intuition)

# Today's plan

- **Temporal resources** via **time-graded modal types**
  - enforcing temporal correctness for the robot arms example
- A **core calculus** for safe programming with temporal resources
  - Fitch-style **time-graded modal types** (for temporal resources)
  - **temporally aware graded algebraic effects** (for time passage)
  - **temporally aware effect handlers** (for user-defined effects)
- A sound **denotational semantics** justifying the proposed design
  - **adjoint strong monoidal functors** (for modalities)
  - $[-]$**-strong time-graded monad** (for effectful computations)
  - a **presheaf example** (for concreteness and intuition)
- **Draft paper:** https://arxiv.org/abs/2210.07738

# Today's plan

- **Temporal resources** via **time-graded modal types**
  - enforcing temporal correctness for the robot arms example
- A **core calculus** for safe programming with temporal resources
  - Fitch-style **time-graded modal types** (for temporal resources)
  - **temporally aware graded algebraic effects** (for time passage)
  - **temporally aware effect handlers** (for user-defined effects)
- A sound **denotational semantics** justifying the proposed design
  - **adjoint strong monoidal functors** (for modalities)
  - $[-]$**-strong time-graded monad** (for effectful computations)
  - a **presheaf example** (for concreteness and intuition)
- **Agda form.:** https://github.com/danelahman/temporal-resources

**Temporal resources via time-graded modal types**

# General desiderata

- Recall the **production line example**

```
let (body', left-door', right-door') =
  paint (body, left-door, right-door) in
                                          ← τ_dry time needs to pass
assemble (body', left-door', right-door')
```

# General desiderata

- Recall the **production line example**

```
let (body', left-door', right-door') =
  paint (body, left-door, right-door) in
                                              ← τ_dry time needs to pass
assemble (body', left-door', right-door')
```

- In general, **we want a flexible framework** in which
  - **time delay** between paint and assemble
    - could be given by **blocking execution** with delay, but
    - equally well could be given by **doing other useful work**, and
    - want it to be **as much as needed** and **as little as possible**
  - $(\mathsf{body}', \mathsf{left\text{-}door}', \mathsf{right\text{-}door}')$ can have **separate drying times**
  - **executing operations** (e.g., delay) should **make time pass**
  - **ops.** should **be redefinable**, while preserving temporal correctness

# A **naive** solution attempt

- What if **we stay in a simply typed effectful language** and additionally make paint **return the desired drying time**?

```
let (τ_dry, body', left-door', right-door') =
  paint (body, left-door, right-door) in

delay τ_dry;

assemble (body', left-door', right-door')
```

# A **naive** solution attempt

- What if **we stay in a simply typed effectful language** and additionally make paint **return the desired drying time**?

```
let (τ_dry, body', left-door', right-door') =
  paint (body, left-door, right-door) in

delay τ_dry;                                  ← τ_dry time now passes

assemble (body', left-door', right-door')
```

- So, **are we done?**

# A **naive** solution attempt

- What if **we stay in a simply typed effectful language** and additionally make paint **return the desired drying time**?

```
let (τ_dry, body', left-door', right-door') =
  paint (body, left-door, right-door) in

delay τ_dry;                                    ← τ_dry time now passes

assemble (body', left-door', right-door')
```

- So, **are we done?**
- **No**,
  - all the burden for correctness is on the programmer's shoulders
  - typechecker saying yes does not guarantee that `delay` happens, or that it happens where it is supposed to happen

# A **naive** solution attempt

- What if **we stay in a simply typed effectful language** and additionally make paint **return the desired drying time**?

```
let (τ_dry, body', left-door', right-door') =
  paint (body, left-door, right-door) in

delay τ_dry;                                   ← τ_dry time now passes

assemble (body', left-door', right-door')
```

- So, **are we done?**
- **No**,
  - all the burden for correctness is on the programmer's shoulders
  - typechecker saying yes does not guarantee that `delay` happens, or that it happens where it is supposed to happen, e.g., **do not want**

```
assemble (body', left-door', right-door');
delay τ_dry                ← total time of program still τ_dry + τ_assemble
```

# Our solution: temporal resource types

# Our solution: temporal resource types

- We use a **time-graded modal type** to capture temporal resources

$$X, Y, Z \quad ::= \quad \ldots \quad | \quad [\tau]\, X$$

e.g., allowing us to work with **resource values/vars.** such as

$$\mathsf{body}' : [\tau_{\mathsf{dry}}]\, \mathsf{Body} \qquad \mathsf{left\text{-}door}' : [\tau_{\mathsf{dry}}]\, \mathsf{Door} \qquad \ldots$$

# Our solution: temporal resource types

- We use a **time-graded modal type** to capture temporal resources

$$X, Y, Z \quad ::= \quad \ldots \quad | \quad [\tau]\, X$$

e.g., allowing us to work with **resource values/vars.** such as

$$\mathsf{body}' : [\tau_{\mathsf{dry}}]\, \mathsf{Body} \qquad \mathsf{left\text{-}door}' : [\tau_{\mathsf{dry}}]\, \mathsf{Door} \qquad \ldots$$

- **Intuition 1:** $[\tau]\, X$ denotes that an $X$-typed resource **becomes usable in at most $\tau$ time units** (and remains so afterwards)

- **Intuition 2: at least $\tau$ time units need to pass** before a program is allowed to access the underlying $X$-typed resource

# Our solution: temporal resource type

- Presented as **time-graded variant** of **Fitch-style modal types**

# Our solution: temporal resource type

- Presented as **time-graded variant** of **Fitch-style modal types**
  - Contexts are extended with **context modalities**

$$\Gamma \quad ::= \quad \cdot \quad | \quad \Gamma, x : X \quad | \quad \Gamma, \langle \tau \rangle$$

# Our solution: temporal resource type

- Presented as **time-graded variant** of **Fitch-style modal types**
  - Contexts are extended with **context modalities**

$$\Gamma \quad ::= \quad \cdot \quad | \quad \Gamma, x : X \quad | \quad \Gamma, \langle \tau \rangle$$

  - **Introduction form** is given by **boxing up a temp. resource**

$$\frac{\Gamma, \langle \tau \rangle \vdash V : X}{\Gamma \vdash \mathsf{box}_\tau \; V : [\, \tau \,]\, X}$$

# Our solution: temporal resource type

- Presented as **time-graded variant** of **Fitch-style modal types**
  - Contexts are extended with **context modalities**

$$\Gamma \quad ::= \quad \cdot \quad | \quad \Gamma, x : X \quad | \quad \Gamma, \langle \tau \rangle$$

  - **Introduction form** is given by **boxing up a temp. resource**

$$\frac{\Gamma, \langle \tau \rangle \vdash V : X}{\Gamma \vdash \mathsf{box}_\tau \; V : [\tau] X}$$

  - **Elimination rule** is given by **unboxing a temp. resource**

$$\frac{\tau \leqslant \mathsf{time}\,\Gamma \qquad \Gamma \mid_\tau \vdash V : [\tau] X \qquad \Gamma, x : X \vdash N : Y \,!\, \tau'}{\Gamma \vdash \mathsf{unbox}_\tau \; V \; \mathsf{as} \; x \; \mathsf{in} \; N : Y \,!\, \tau'}$$

# Our solution: temporal resource type

- Presented as **time-graded variant** of **Fitch-style modal types**
  - Contexts are extended with **context modalities**

$$\Gamma \quad ::= \quad \cdot \quad | \quad \Gamma, x : X \quad | \quad \Gamma, \langle \tau \rangle$$

  - **Introduction form** is given by **boxing up a temp. resource**

$$\frac{\Gamma, \langle \tau \rangle \vdash V : X}{\Gamma \vdash \mathsf{box}_\tau \; V : [\tau] X}$$

  - **Elimination rule** is given by **unboxing a temp. resource**

$$\frac{\tau \leqslant \mathrm{time}\,\Gamma \qquad \lfloor \Gamma \rfloor_\tau \vdash V : [\tau] X \qquad \Gamma, x : X \vdash N : Y \,!\, \tau'}{\Gamma \vdash \mathsf{unbox}_\tau \; V \; \mathsf{as} \; x \; \mathsf{in} \; N : Y \,!\, \tau'}$$

  where $\lfloor \Gamma \rfloor_\tau$ takes $\Gamma$ to a $\tau$ **time units earlier state**[1], e.g., as in

$$\lfloor \Gamma, x : X, \langle 4 \rangle, y : Y, \langle 1 \rangle, z : Z \rfloor_3 \; \equiv \; \Gamma, x : X, \langle 2 \rangle$$

[1] We have $\lfloor - \rfloor_\tau \dashv \langle \tau \rangle$ for contexts $\Gamma$ with $\tau \leqslant \mathrm{time}\,\Gamma$ and rens. between them.

# Our solution: how we make time pass

# Our solution: how we make time pass

- We propose **temporally aware** **graded algebraic effects**, e.g.,

$$\mathsf{paint} \quad : \quad \overrightarrow{\mathsf{Part}} \quad \rightsquigarrow \quad \overrightarrow{[\tau_{\mathsf{dry}_i}]\,\mathsf{Part}} \;\; ! \;\; \tau_{\mathsf{paint}}$$

# Our solution: how we make time pass

- We propose **temporally aware graded algebraic effects**, e.g.,

$$\mathsf{paint} \quad : \quad \overrightarrow{\mathsf{Part}} \quad \rightsquigarrow \quad \overrightarrow{[\tau_{\mathsf{dry}_i}]\,\mathsf{Part}} \ ! \ \tau_{\mathsf{paint}}$$

giving rise to **operation calls** with **temporal awareness**

$$\frac{\Gamma \vdash V : \mathsf{Body} \times \mathsf{Door} \times \mathsf{Door} \qquad \Gamma\,,\,\langle\,\tau_{\mathsf{paint}}\,\rangle\,,\; y : [\tau_{\mathsf{dry}}]\,\mathsf{Body} \times [\tau_{\mathsf{dry}}]\,\mathsf{Door} \times [\tau_{\mathsf{dry}}]\,\mathsf{Door} \vdash M : X \ ! \ \tau}{\Gamma \vdash \mathsf{paint}\ V\ (y\,.\,M) : X \ ! \ \tau_{\mathsf{paint}} + \tau}$$

where the **cont.** $M$ can assume that $\tau_{\mathsf{paint}}$ **additional time has passed** before it starts executing (compared to $\mathsf{paint}\ V\ (y\,.\,M)$)

# Our solution: how we make time pass

- We propose **temporally aware** **graded algebraic effects**, e.g.,

$$\mathsf{paint} \quad : \quad \overrightarrow{\mathsf{Part}} \quad \rightsquigarrow \quad \overrightarrow{[\tau_{\mathsf{dry}_i}]\,\mathsf{Part}} \;\; ! \;\; \tau_{\mathsf{paint}}$$

  giving rise to **operation calls** with **temporal awareness**

$$\frac{\Gamma \vdash V : \mathsf{Body} \times \mathsf{Door} \times \mathsf{Door} \qquad \Gamma\,,\,\langle\,\tau_{\mathsf{paint}}\,\rangle\,,\; y : [\tau_{\mathsf{dry}}]\,\mathsf{Body} \times [\tau_{\mathsf{dry}}]\,\mathsf{Door} \times [\tau_{\mathsf{dry}}]\,\mathsf{Door} \vdash M : X \;!\; \tau}{\Gamma \vdash \mathsf{paint}\; V\; (y\,.\,M) : X \;!\; \tau_{\mathsf{paint}} + \tau}$$

  where the **cont.** $M$ can assume that $\tau_{\mathsf{paint}}$ **additional time has passed** before it starts executing (compared to paint $V\ (y\,.\,M)$)

- This **“temporal action”** also happens in **seq. composition**

$$\frac{\Gamma \vdash M : X \;!\; \tau \qquad \Gamma\,,\,\langle\,\tau\,\rangle\,,\; x : X \vdash N : Y \;!\; \tau'}{\Gamma \vdash \mathsf{let}\; x = M\; \mathsf{in}\; N : Y \;!\; \tau + \tau'}$$

# Our solution: back to controlling robot arm

# Our solution: back to controlling robot arm

- Using the above, we can now **rewrite our example** as

```
let (body', left-door', right-door') =          ← resource-typed variables
  paint (body, left-door, right-door) in

delay τ_dry;                                     ← forces τ_dry time to pass

unbox body' as body'' in   ← context: Γ , body′ : [τ_dry] Body , … , ⟨ τ_dry ⟩
unbox left-door' as left-door'' in
unbox right-door' as right-door'' in

assemble (body'', left-door'', right-door'')    ← non-resource-typed variables
```

# Our solution: back to controlling robot arm

- Using the above, we can now **rewrite our example** as

```
let (body', left-door', right-door') =           ← resource-typed variables
  paint (body, left-door, right-door) in

delay τ_dry;                                     ← forces τ_dry time to pass

unbox body' as body'' in  ← context: Γ , body′ : [τ_dry] Body , … , ⟨ τ_dry ⟩
unbox left-door' as left-door'' in
unbox right-door' as right-door'' in

assemble (body'', left-door'', right-door'')     ← non-resource-typed variables
```

This looks remarkably similar to the naive attempt from earlier!

# Our solution: back to controlling robot arm

- Using the above, we can now **rewrite our example** as

```
let (body', left-door', right-door') =            ← resource-typed variables
  paint (body, left-door, right-door) in

delay τ_dry;                                      ← forces τ_dry time to pass

unbox body' as body'' in   ← context: Γ , body' : [τ_dry] Body , ... , ⟨τ_dry⟩
unbox left-door' as left-door'' in
unbox right-door' as right-door'' in

assemble (body'', left-door'', right-door'')      ← non-resource-typed variables
```

This looks remarkably similar to the naive attempt from earlier!

- Alternatively, instead of **blocking execution** with

```
delay τ_dry;
```

we could have equally well called enough **other useful operations**

```
op_1; op_2; ...; op_n;          ← as long as they collectively take ⩾ τ_dry time
```

# Making it formal: core calculus $\lambda_{[\tau]}$

# Core calculus: types

- Based on Levy et al's fine-grain call-by-value (FGCBV) calculus

- **Ground types** (for base types $\mathsf{b} \in \mathcal{B}$, and where $\tau \in \mathbb{N}$)

$$A, B \quad ::= \quad \mathsf{b} \;\mid\; 1 \;\mid\; A \times B \;\mid\; [\tau]\, A$$

- **Operation signatures** (for operations $\mathsf{op} \in \mathcal{O}$)

$$\mathsf{op} \;:\; A_{\mathsf{op}} \;\rightsquigarrow\; B_{\mathsf{op}} \,!\, \tau_{\mathsf{op}}$$

- **Value types** (extend ground types)

$$X, Y, Z \quad ::= \quad A \;\mid\; X \times Y \;\mid\; X \to Y \,!\, \tau \;\mid\; [\tau]\, X$$

- **Computation types**

$$X \,!\, \tau$$

# Core calculus: terms

- Terms are split into **values** and **computations**
- **Values**

$$V, W \quad ::= \quad x \mid \mathsf{f}(V_1, \ldots, V_n) \mid () \mid \ldots \mid \mathsf{box}_\tau\ V$$

- **Computations**

$$\begin{array}{rcll} M, N & ::= & \mathsf{return}\ V & \\ & \mid & \mathsf{let}\ x = M\ \mathsf{in}\ N & \\ & \mid & \ldots & \\ & \mid & \mathsf{op}\ V\ (y\,.\,M) & \leftarrow \text{user-redefinable via handling} \\ & \mid & \mathsf{delay}\ \tau\ M & \leftarrow \text{primitive, not user-definable} \\ & \mid & \mathsf{handle}\ M\ \mathsf{with}\ (x.k.M_{\mathsf{op}})_{\mathsf{op} \in \mathcal{O}}\ \mathsf{to}\ y\ \mathsf{in}\ N & \\ & \mid & \mathsf{unbox}_\tau\ V\ \mathsf{as}\ x\ \mathsf{in}\ N & \end{array}$$

# Core calculus: type system

- Well-typed values and computations typed using **judgements**

$$\Gamma \vdash V : X \qquad\qquad \Gamma \vdash M : X \,!\, \tau$$

- For example, **typing rules** for **variables**[2] and **returning values**

$$\frac{}{\Gamma, x : X, \Gamma' \vdash x : X} \qquad\qquad \frac{\Gamma \vdash V : X}{\Gamma \vdash \mathsf{return}\ V : X \,!\, 0}$$

and for **effect handling**

$$\frac{\Gamma \vdash M : X \,!\, \tau \qquad \Gamma\,,\, \langle \tau \rangle\,,\, y : X \vdash N : Y \,!\, \tau' \qquad \Big(\forall \tau''.\ \Gamma\,,\ x : A_{\mathsf{op}}\,,\ k : [\tau_{\mathsf{op}}](B_{\mathsf{op}} \to Y \,!\, \tau'') \vdash M_{\mathsf{op}} : Y \,!\, \tau_{\mathsf{op}} + \tau''\Big)_{\mathsf{op} \in \mathcal{O}}}{\Gamma \vdash \mathsf{handle}\ M\ \mathsf{with}\ (x.k.M_{\mathsf{op}})_{\mathsf{op} \in \mathcal{O}}\ \mathsf{to}\ y\ \mathsf{in}\ N : Y \,!\, \tau + \tau'}$$

[2] No restriction on $\Gamma'$ compared to Clouston's Fitch-style modal lambda-calculi

# Core calculus: type system

- Well-typed values and computations typed using **judgements**

$$\Gamma \vdash V : X \qquad \Gamma \vdash M : X \,!\, \tau$$

- For example, **typing rules** for **variables**[2] and **returning values**

$$\frac{}{\Gamma, x\!:\!X, \Gamma' \vdash x : X} \qquad \frac{\Gamma \vdash V : X}{\Gamma \vdash \mathsf{return}\ V : X \,!\, 0}$$

  and for **effect handling**

$$\frac{\Gamma \vdash M : X \,!\, \tau \qquad \Gamma\, ,\, \langle\, \tau \,\rangle\, ,\, y\!:\!X \vdash N : Y \,!\, \tau' \qquad \Big(\forall \tau''.\ \Gamma\, ,\ x\!:\!A_{\mathsf{op}}\, ,\ k\!:\![\tau_{\mathsf{op}}]\big(B_{\mathsf{op}} \to Y \,!\, \tau''\big) \vdash M_{\mathsf{op}} : Y \,!\, \tau_{\mathsf{op}} + \tau''\Big)_{\mathsf{op} \in \mathcal{O}}}{\Gamma \vdash \mathsf{handle}\ M\ \mathsf{with}\ \big(x.k.M_{\mathsf{op}}\big)_{\mathsf{op} \in \mathcal{O}}\ \mathsf{to}\ y\ \mathsf{in}\ N : Y \,!\, \tau + \tau'}$$

- **Note: No sub-effecting!** Non-trivial due to $\langle\, \tau \,\rangle$. Future work.

[2] No restriction on $\Gamma'$ compared to Clouston's Fitch-style modal lambda-calculi

# Core calculus: admissible typing rules

- **Standard structural rules** (weakening, contraction, exchange)

# Core calculus: admissible typing rules

- **Standard structural rules** (weakening, contraction, exchange)
- **Strong monoidal functor (with co-strength) laws for $\langle - \rangle$**

$$\frac{\Gamma, \langle 0 \rangle \vdash J}{\Gamma \vdash J} \qquad \frac{\Gamma, \langle \tau_1 + \tau_2 \rangle \vdash J}{\Gamma, \langle \tau_1 \rangle, \langle \tau_2 \rangle \vdash J} \qquad \frac{\Gamma, \langle \tau \rangle \vdash J \quad \tau \leqslant \tau'}{\Gamma, \langle \tau' \rangle \vdash J} \qquad \frac{\Gamma, \langle \tau \rangle, x : X \vdash J}{\Gamma, x : X, \langle \tau \rangle \vdash J}$$

  - for 2nd rule it is useful that unbox uses $\lvert \Gamma \rvert_\tau$ and not $\Gamma_1, \Gamma_2$
  - 4th rule shows that all types are **monotone** with respect to time

# Core calculus: admissible typing rules

- **Standard structural rules** (weakening, contraction, exchange)
- **Strong monoidal functor (with co-strength) laws for $\langle - \rangle$**

$$\frac{\Gamma, \langle 0 \rangle \vdash J}{\Gamma \vdash J} \qquad \frac{\Gamma, \langle \tau_1 + \tau_2 \rangle \vdash J}{\Gamma, \langle \tau_1 \rangle, \langle \tau_2 \rangle \vdash J} \qquad \frac{\Gamma, \langle \tau \rangle \vdash J \quad \tau \leqslant \tau'}{\Gamma, \langle \tau' \rangle \vdash J} \qquad \frac{\Gamma, \langle \tau \rangle, x : X \vdash J}{\Gamma, x : X, \langle \tau \rangle \vdash J}$$

  - for 2nd rule it is useful that unbox uses $\lvert \Gamma \rvert_\tau$ and not $\Gamma_1, \Gamma_2$
  - 4th rule shows that all types are **monotone** with respect to time

- **Proof sketch** (for the above two groups of rules):
  - **(a)** inductively define **renaming relation** $\rho : \Gamma \rightsquigarrow \Gamma'$
  - **(b)** **prove** that if $\Gamma \vdash J$ and $\rho : \Gamma \rightsquigarrow \Gamma'$, then $\Gamma' \vdash J[\rho]$

# Core calculus: admissible typing rules

- **Standard structural rules** (weakening, contraction, exchange)
- **Strong monoidal functor (with co-strength) laws for $\langle - \rangle$**

$$\frac{\Gamma, \langle 0 \rangle \vdash J}{\Gamma \vdash J} \qquad \frac{\Gamma, \langle \tau_1 + \tau_2 \rangle \vdash J}{\Gamma, \langle \tau_1 \rangle, \langle \tau_2 \rangle \vdash J} \qquad \frac{\Gamma, \langle \tau \rangle \vdash J \quad \tau \leqslant \tau'}{\Gamma, \langle \tau' \rangle \vdash J} \qquad \frac{\Gamma, \langle \tau \rangle, x : X \vdash J}{\Gamma, x : X, \langle \tau \rangle \vdash J}$$

  - for 2nd rule it is useful that unbox uses $\lvert \Gamma \rvert_\tau$ and not $\Gamma_1, \Gamma_2$
  - 4th rule shows that all types are **monotone** with respect to time

- **Proof sketch** (for the above two groups of rules):
  **(a)** inductively define **renaming relation** $\rho : \Gamma \rightsquigarrow \Gamma'$
  **(b)** **prove** that if $\Gamma \vdash J$ and $\rho : \Gamma \rightsquigarrow \Gamma'$, then $\Gamma' \vdash J[\rho]$

- **Substitution rules**

$$\frac{\Gamma, x : X, \Gamma' \vdash J \qquad \Gamma \vdash V : X}{\Gamma, \Gamma' \vdash J[V/x]}$$

# Core calculus: equational theory

- Given by **equations** between well-typed values and computations

$$\Gamma \vdash V \equiv W : X \qquad\qquad \Gamma \vdash M \equiv N : X \; ! \; \tau$$

# Core calculus: equational theory

- Given by **equations** between well-typed values and computations

$$\Gamma \vdash V \equiv W : X \qquad\qquad \Gamma \vdash M \equiv N : X \mathrel{!} \tau$$

- **Standard $\beta$-/$\eta$-equations** of FGCBV-based calculi, e.g.,

$$\Gamma \vdash (\mathsf{let}\ x = \mathsf{return}\ V\ \mathsf{in}\ N) \equiv N[V/x] : Y \mathrel{!} \tau$$

# Core calculus: equational theory

- Given by **equations** between well-typed values and computations

$$\Gamma \vdash V \equiv W : X \qquad\qquad \Gamma \vdash M \equiv N : X \mathbin{!} \tau$$

- **Standard $\beta$-/$\eta$-equations** of FGCBV-based calculi, e.g.,

$$\Gamma \vdash (\mathsf{let}\ x = \mathsf{return}\ V\ \mathsf{in}\ N) \equiv N[V/x] : Y \mathbin{!} \tau$$

- **Algebraicity equations for algebraic operations**

- **Homomorphism equations for effect handling**

# Core calculus: equational theory

- Given by **equations** between well-typed values and computations

$$\Gamma \vdash V \equiv W : X \qquad\qquad \Gamma \vdash M \equiv N : X \,!\, \tau$$

- **Standard $\beta$-/$\eta$-equations** of FGCBV-based calculi, e.g.,

$$\Gamma \vdash (\mathsf{let}\ x = \mathsf{return}\ V\ \mathsf{in}\ N) \equiv N[V/x] : Y \,!\, \tau$$

- **Algebraicity equations for algebraic operations**

- **Homomorphism equations for effect handling**

- **$\beta$-/$\eta$-equations for temporal resources**

$$\Gamma \vdash \mathsf{unbox}_\tau\ (\mathsf{box}_\tau\ V)\ \mathsf{as}\ x\ \mathsf{in}\ N \equiv N[V/x] : Y \,!\, \tau'$$

$$\Gamma \vdash \mathsf{unbox}_\tau\ W\ \mathsf{as}\ x\ \mathsf{in}\ N[\mathsf{box}_\tau\ x/y] \equiv N[W/y] : Y \,!\, \tau'$$

# Core calculus: equational theory

- Given by **equations** between well-typed values and computations

$$\Gamma \vdash V \equiv W : X \qquad\qquad \Gamma \vdash M \equiv N : X \; ! \; \tau$$

- **Standard $\beta$-/$\eta$-equations** of FGCBV-based calculi, e.g.,

$$\Gamma \vdash (\mathsf{let}\ x = \mathsf{return}\ V\ \mathsf{in}\ N) \equiv N[V/x] : Y \; ! \; \tau$$

- **Algebraicity equations for algebraic operations**

- **Homomorphism equations for effect handling**

- **$\beta$-/$\eta$-equations for temporal resources**

$$\Gamma \vdash \mathsf{unbox}_\tau\ (\mathsf{box}_\tau\ V)\ \mathsf{as}\ x\ \mathsf{in}\ N \equiv N[V/x] : Y \; ! \; \tau'$$

$$\Gamma \vdash \mathsf{unbox}_\tau\ W\ \mathsf{as}\ x\ \mathsf{in}\ N[\mathsf{box}_\tau\ x/y] \equiv N[W/y] : Y \; ! \; \tau'$$

- **Optional extension:** 0- **and** $+$**-equations** for delay ops.

# Making it formal: denotational semantics

# Denotational semantics: **big picture**

- Given suitable **category** $\mathbb{C}$ and suitable **structure** (e.g., $T$) on it
- Given objects $[\![\mathrm{b}]\!] \in \mathbb{C}$ for all **base types** $\mathrm{b} \in \mathcal{B}$

- We **interpret types** $X$ as objects $[\![X]\!] \in \mathbb{C}$
- We **interpret contexts** $\Gamma$ as objects $[\![\Gamma]\!] \in \mathbb{C}$
- We **interpret well-typed values** $\Gamma \vdash V : X$ as morphisms

$$[\![\Gamma \vdash V : X]\!] : [\![\Gamma]\!] \longrightarrow [\![X]\!]$$

- We **interpret well-typed computations** $\Gamma \vdash M : X \,!\, \tau$ as

$$[\![\Gamma \vdash M : X \,!\, \tau]\!] : [\![\Gamma]\!] \longrightarrow T\,\tau\,[\![X]\!]$$

- **Such that:** If $\Gamma \vdash I \equiv J$, then $[\![\Gamma \vdash I]\!] \equiv [\![\Gamma \vdash J]\!]$ **(soundness)**

# Denotational semantics: **category** $\mathbb{C}$

- Want $\mathbb{C}$ to have **binary products** $(\mathbb{1}, A \times B)$
- Want $\mathbb{C}$ to have **exponentials** $A \Rightarrow B$
  - for completeness, would need to restrict to Kleisli exponentials

# Denotational semantics: category $\mathbb{C}$

- Want $\mathbb{C}$ to have **binary products** $(\mathbb{1}, A \times B)$
- Want $\mathbb{C}$ to have **exponentials** $A \Rightarrow B$
  - for completeness, would need to restrict to Kleisli exponentials
- **Example: presheaf category** $\mathrm{Set}^{(\mathbb{N},\leqslant)}$
  - objects are **covariant functors** $A : (\mathbb{N}, \leqslant) \longrightarrow \mathrm{Set}$
  - gives Kripke's **possible worlds style semantics**

# Denotational semantics: category $\mathbb{C}$

- Want $\mathbb{C}$ to have **binary products** $(\mathbb{1}, A \times B)$
- Want $\mathbb{C}$ to have **exponentials** $A \Rightarrow B$
  - for completeness, would need to restrict to Kleisli exponentials
- **Example: presheaf category** $\mathrm{Set}^{(\mathbb{N},\leqslant)}$
  - objects are **covariant functors** $A : (\mathbb{N}, \leqslant) \longrightarrow \mathrm{Set}$
  - gives Kripke's **possible worlds style semantics**
  - but with **all types being monotone**: given $A \in \mathrm{Set}^{(\mathbb{N},\leqslant)}$, then

$$t_1 \leqslant t_2 \quad \text{implies} \quad A(t_1 \leqslant t_2) : A(t_1) \longrightarrow A(t_2)$$

# Denotational semantics: category $\mathbb{C}$

- Want $\mathbb{C}$ to have **binary products** $(\mathbb{1}, A \times B)$
- Want $\mathbb{C}$ to have **exponentials** $A \Rightarrow B$
  - for completeness, would need to restrict to Kleisli exponentials

- **Example: presheaf category** $\mathrm{Set}^{(\mathbb{N},\leqslant)}$
  - objects are **covariant functors** $A : (\mathbb{N}, \leqslant) \longrightarrow \mathrm{Set}$
  - gives Kripke's **possible worlds style semantics**
  - but with **all types being monotone**: given $A \in \mathrm{Set}^{(\mathbb{N},\leqslant)}$, then

$$t_1 \leqslant t_2 \qquad \text{implies} \qquad A(t_1 \leqslant t_2) : A(t_1) \longrightarrow A(t_2)$$

  - and when unfolding std. defs., **exponentials are given as**

$$(A \Rightarrow B)(t) \quad \stackrel{\text{def}}{=} \quad \Big(f_{t'} : A(t') \longrightarrow B(t')\Big)_{t' \in \{t' \in \mathbb{N} \,|\, t \leqslant t'\}}$$

where all $f_{t'}$ are also asked to be natural in $t'$

# Denotational semantics: (modal) types

- Want there to be **strong monoidal functor** (for temp. res. type)

$$[-] : (\mathbb{N}, \leqslant) \longrightarrow [\mathbb{C}, \mathbb{C}]$$

with the **strong monoidality** witnessed by the natural isos.[3]

$$\varepsilon_A : [0]\, A \xrightarrow{\cong} A \qquad \delta_{A,\tau_1,\tau_2} : [\tau_1 + \tau_2]\, A \xrightarrow{\cong} [\tau_1]\, ([\tau_2]\, A)$$

[3] In Fitch-style, the S4 modality $\Box$ is interpreted by an **idempotent comonad**

# Denotational semantics: (modal) types

- Want there to be **strong monoidal functor** (for temp. res. type)

$$[-] : (\mathbb{N}, \leqslant) \longrightarrow [\mathbb{C}, \mathbb{C}]$$

with the **strong monoidality** witnessed by the natural isos.[3]

$$\varepsilon_A : [0]\, A \xrightarrow{\cong} A \qquad \delta_{A,\tau_1,\tau_2} : [\tau_1 + \tau_2]\, A \xrightarrow{\cong} [\tau_1]\,([\tau_2]\, A)$$

- We then **interpret types** by straightforward struct. recursion, e.g.,

$$[\![ [\tau]\, X ]\!] \quad \stackrel{\text{def}}{=} \quad [\tau]\, [\![ X ]\!]$$

[3] In Fitch-style, the S4 modality $\Box$ is interpreted by an **idempotent comonad**

# Denotational semantics: (modal) types

- Want there to be **strong monoidal functor** (for temp. res. type)

$$[-] : (\mathbb{N}, \leqslant) \longrightarrow [\mathbb{C}, \mathbb{C}]$$

  with the **strong monoidality** witnessed by the natural isos.[3]

$$\varepsilon_A : [0]\, A \xrightarrow{\cong} A \qquad \delta_{A,\tau_1,\tau_2} : [\tau_1 + \tau_2]\, A \xrightarrow{\cong} [\tau_1]\,([\tau_2]\, A)$$

- We then **interpret types** by straightforward struct. recursion, e.g.,

$$[\![ [\tau]\, X ]\!] \quad \stackrel{\text{def}}{=} \quad [\tau]\, [\![ X ]\!]$$

- In the **presheaf example**, we define $[-]$ on objects as

$$([\tau]\, A)(t) \quad \stackrel{\text{def}}{=} \quad A(t + \tau)$$

[3] In Fitch-style, the S4 modality $\Box$ is interpreted by an **idempotent comonad**

# Denotational semantics: (modal) contexts

- Want there to be **strong monoidal functor** (for ctx. modalities)

$$\langle - \rangle : (\mathbb{N}, \leqslant)^{\mathsf{op}} \longrightarrow [\mathbb{C}, \mathbb{C}]$$

with the **strong monoidality** witnessed by the natural isos.[4]

$$\eta_A : A \xrightarrow{\cong} \langle 0 \rangle A \qquad \mu_{A,\tau_1,\tau_2} : \langle \tau_1 \rangle (\langle \tau_2 \rangle A) \xrightarrow{\cong} \langle \tau_1 + \tau_2 \rangle A$$

[4] In Fitch-style, the ctx. modality for S4 is interpreted by an **idempotent monad**

# Denotational semantics: (modal) contexts

- Want there to be **strong monoidal functor** (for ctx. modalities)

$$\langle - \rangle : (\mathbb{N}, \leqslant)^{\mathbf{op}} \longrightarrow [\mathbb{C}, \mathbb{C}]$$

with the **strong monoidality** witnessed by the natural isos.[4]

$$\eta_A : A \xrightarrow{\cong} \langle 0 \rangle A \qquad \mu_{A,\tau_1,\tau_2} : \langle \tau_1 \rangle (\langle \tau_2 \rangle A) \xrightarrow{\cong} \langle \tau_1 + \tau_2 \rangle A$$

- We then **interpret contexts** as $[\![\Gamma]\!] \stackrel{\text{def}}{=} [\![\Gamma]\!]^e \, \mathbb{1}$, where

$$[\![\Gamma]\!]^e : \mathbb{C} \longrightarrow \mathbb{C} \qquad [\![\Gamma, \langle \tau \rangle]\!]^e \, A \stackrel{\text{def}}{=} \langle \tau \rangle ([\![\Gamma]\!]^e \, A)$$

as we then conveniently have isos like $[\![\Gamma_1, \Gamma_2]\!] \cong [\![\Gamma_2]\!]^e \, ([\![\Gamma_1]\!])$

[4] In Fitch-style, the ctx. modality for S4 is interpreted by an **idempotent monad**

# Denotational semantics: (modal) contexts

- Want there to be **strong monoidal functor** (for ctx. modalities)

$$\langle - \rangle : (\mathbb{N}, \leqslant)^{\mathbf{op}} \longrightarrow [\mathbb{C}, \mathbb{C}]$$

  with the **strong monoidality** witnessed by the natural isos.[4]

$$\eta_A : A \xrightarrow{\cong} \langle 0 \rangle A \qquad \mu_{A,\tau_1,\tau_2} : \langle \tau_1 \rangle (\langle \tau_2 \rangle A) \xrightarrow{\cong} \langle \tau_1 + \tau_2 \rangle A$$

- We then **interpret contexts** as $[\![\Gamma]\!] \stackrel{\text{def}}{=} [\![\Gamma]\!]^e \, \mathbb{1}$, where

$$[\![\Gamma]\!]^e : \mathbb{C} \longrightarrow \mathbb{C} \qquad [\![\Gamma, \langle \tau \rangle]\!]^e \, A \stackrel{\text{def}}{=} \langle \tau \rangle ([\![\Gamma]\!]^e \, A)$$

  as we then conveniently have isos like $[\![\Gamma_1, \Gamma_2]\!] \cong [\![\Gamma_2]\!]^e \, ([\![\Gamma_1]\!])$

- In the **presheaf example**, we define $\langle - \rangle$ on objects as

$$(\langle \tau \rangle A)(t) \stackrel{\text{def}}{=} (\tau \leqslant t) \times A(t \mathbin{\dot{-}} \tau)$$

[4] In Fitch-style, the ctx. modality for S4 is interpreted by an **idempotent monad**

# Denotational semantics: mod. interaction

- Want there to be a **family of adjunctions**[5]

$$\langle \tau \rangle \dashv [\tau]$$

witnessed by natural transformations

$$\eta^{\dashv}_{A,\tau} : A \longrightarrow [\tau]\,(\langle \tau \rangle A) \qquad \varepsilon^{\dashv}_{A,\tau} : \langle \tau \rangle ([\tau]\, A) \longrightarrow A$$

  - satisfying the **standard triangle laws**, and
  - interacting well with the **strong monoidal structures**

[5] In Fitch-style modal $\lambda$-calculi, one also requires an **adjunction between mods.**

# Denotational semantics: mod. interaction

- Want there to be a **family of adjunctions**[5]

$$\langle\, \tau \,\rangle \dashv [\tau]$$

witnessed by natural transformations

$$\eta^{\dashv}_{A,\tau} : A \longrightarrow [\tau]\,(\langle\, \tau \,\rangle\, A) \qquad \varepsilon^{\dashv}_{A,\tau} : \langle\, \tau \,\rangle\,([\tau]\, A) \longrightarrow A$$

  - satisfying the **standard triangle laws**, and
  - interacting well with the **strong monoidal structures**

- In the **presheaf example**,
  - $\eta^{\dashv}_{A,\tau}$ and $\varepsilon^{\dashv}_{A,\tau}$ are given by **id. on $A$-values**, plus by $\leqslant$-reasoning
  - $\varepsilon^{\dashv}_{A,\tau}$ is definable because of the $(\tau \leqslant t)$ **condition** in $(\langle\, \tau \,\rangle A)(t)$

[5] In Fitch-style modal $\lambda$-calculi, one also requires an **adjunction between mods.**

# Denotational semantics: comp. effects

- Want there to be a **graded monad** (disc. graded as no sub-eff.)

$$T : \mathbb{N} \longrightarrow [\mathbb{C}, \mathbb{C}]$$

with **unit** and **multiplication** (satisfying appropriate laws)

$$\eta^T_A : A \longrightarrow T\,0\,A \qquad \mu^T_{A,\tau_1,\tau_2} : T\,\tau_1\,(T\,\tau_2\,A) \longrightarrow T\,(\tau_1 + \tau_2)\,A$$

# Denotational semantics: comp. effects

- Want there to be a **graded monad** (disc. graded as no sub-eff.)

$$T : \mathbb{N} \longrightarrow [\mathbb{C}, \mathbb{C}]$$

with **unit** and **multiplication** (satisfying appropriate laws)

$$\eta^T_A : A \longrightarrow T\,0\,A \qquad \mu^T_{A,\tau_1,\tau_2} : T\,\tau_1\,(T\,\tau_2\,A) \longrightarrow T\,(\tau_1 + \tau_2)\,A$$

and with a $[-]$-**strength**[6] (satisfying variants of std. str. laws)

$$\mathsf{str}^T_{A,B,\tau} : [\tau]\,A \times T\,\tau\,B \longrightarrow T\,\tau\,(A \times B)$$

[6] Terminology follows the parlance of Bierman and de Paiva ($\Diamond$ was $\Box$-strong)

# Denotational semantics: comp. effects

- Want there to be a **graded monad** (disc. graded as no sub-eff.)

$$T : \mathbb{N} \longrightarrow [\mathbb{C}, \mathbb{C}]$$

with **unit** and **multiplication** (satisfying appropriate laws)

$$\eta^T_A : A \longrightarrow T\,0\,A \qquad \mu^T_{A,\tau_1,\tau_2} : T\,\tau_1\,(T\,\tau_2\,A) \longrightarrow T\,(\tau_1 + \tau_2)\,A$$

and with a **$[-]$-strength**[6] (satisfying variants of std. str. laws)

$$\mathrm{str}^T_{A,B,\tau} : [\tau]\,A \times T\,\tau\,B \longrightarrow T\,\tau\,(A \times B)$$

- The latter is equivalent to **$[-]$-variant of enrichment of T**, i.e.,

$$[\tau]\,(A \Rightarrow B) \;\longrightarrow\; (T\,\tau\,A \Rightarrow T\,\tau\,B)$$

---

[6] Terminology follows the parlance of Bierman and de Paiva ($\Diamond$ was $\Box$-strong)

# Denotational semantics: comp. effects

- Want there to be a **graded monad** (disc. graded as no sub-eff.)

$$T : \mathbb{N} \longrightarrow [\mathbb{C}, \mathbb{C}]$$

with **unit** and **multiplication** (satisfying appropriate laws)

$$\eta^T_A : A \longrightarrow T\,0\,A \qquad \mu^T_{A,\tau_1,\tau_2} : T\,\tau_1\,(T\,\tau_2\,A) \longrightarrow T\,(\tau_1 + \tau_2)\,A$$

and with a **$[-]$-strength**[6] (satisfying variants of std. str. laws)

$$\mathsf{str}^T_{A,B,\tau} : [\tau]\,A \times T\,\tau\,B \longrightarrow T\,\tau\,(A \times B)$$

- The latter is equivalent to **$[-]$-variant of enrichment of T**, i.e.,

$$[\tau]\,(A \Rightarrow B) \;\longrightarrow\; (T\,\tau\,A \Rightarrow T\,\tau\,B)$$

- Also require $T$ to have **alg. ops.** and support for **eff. handling**

[6] Terminology follows the parlance of Bierman and de Paiva ($\Diamond$ was $\Box$-strong)

# Denotational semantics: comp. effects

- In the **presheaf example**, the **graded monad**[7] is given by cases

$$\frac{a \in A(t)}{\mathsf{ret}\ a \in (T\ 0\ A)(t)}$$

$$\frac{a \in [\![A_{\mathsf{op}}]\!](t) \qquad k \in \big([\tau_{\mathsf{op}}]\ ([\![B_{\mathsf{op}}]\!] \Rightarrow\ T\ \tau\ A)\big)(t)}{\mathsf{op}\ a\ k \in (T\ (\tau_{\mathsf{op}} + \tau)\ A)(t)}$$

$$\frac{k \in [\tau]\ (T\ \tau'\ A)(t)}{\mathsf{delay}\ \tau\ k \in (T\ (\tau + \tau')\ A)(t)}$$

with the graded-monadic structure given by unsurprising recursion

[7] This $T$ is for the setting where there are **no delay-equations** in the calculus

# Denotational semantics: comp. effects

- In the **presheaf example**, the **graded monad**[7] is given by cases

$$\frac{a \in A(t)}{\mathsf{ret}\, a \in (T\, 0\, A)(t)}$$

$$\frac{a \in [\![A_{\mathsf{op}}]\!](t) \qquad k \in \big([\tau_{\mathsf{op}}]\, ([\![B_{\mathsf{op}}]\!] \Rightarrow T\, \tau\, A)\big)(t)}{\mathsf{op}\, a\, k \in (T\, (\tau_{\mathsf{op}} + \tau)\, A)(t)}$$

$$\frac{k \in [\tau]\, (T\, \tau'\, A)(t)}{\mathsf{delay}\, \tau\, k \in (T\, (\tau + \tau')\, A)(t)}$$

with the graded-monadic structure given by unsurprising recursion

- Direct def. in our **Agda** formalisation uses **induction-recursion**
  - IR needed so that $k$ is natural for continuations in effect handling

[7] This $T$ is for the setting where there are **no delay-equations** in the calculus

# Denotational semantics: **(value) terms**

- The **interpretation of terms is unsurprising**
  - follows usual patterns of interpreting FGCBV terms
  - just need to carefully manage the $\langle - \rangle$ and $[-]$ modalities

# Denotational semantics: (value) terms

- The **interpretation of terms is unsurprising**
  - follows usual patterns of interpreting FGCBV terms
  - just need to carefully manage the $\langle - \rangle$ and $[-]$ modalities

- For example, **variables** are interpreted **as (product) projections**

$$\llbracket \Gamma, x : X, \Gamma' \vdash x : X \rrbracket \stackrel{\text{def}}{=}$$

$$\llbracket \Gamma, x : X, \Gamma' \rrbracket \xrightarrow{\cong} \llbracket \Gamma' \rrbracket^{e} (\llbracket \Gamma \rrbracket \times \llbracket X \rrbracket) \xrightarrow{\mathsf{e}}$$

$$\langle \mathsf{time}\, \Gamma' \rangle (\llbracket \Gamma \rrbracket \times \llbracket X \rrbracket) \xrightarrow{\varepsilon^{\langle\rangle}} \llbracket \Gamma \rrbracket \times \llbracket X \rrbracket \xrightarrow{\mathsf{snd}} \llbracket X \rrbracket$$

# Denotational semantics: (value) terms

- The **interpretation of terms is unsurprising**
  - follows usual patterns of interpreting FGCBV terms
  - just need to carefully manage the $\langle - \rangle$ and $[-]$ modalities

- For example, **variables** are interpreted **as (product) projections**

$$\llbracket \Gamma, x : X, \Gamma' \vdash x : X \rrbracket \stackrel{\text{def}}{=}$$

$$\llbracket \Gamma, x : X, \Gamma' \rrbracket \xrightarrow{\cong} \llbracket \Gamma' \rrbracket^e \, (\llbracket \Gamma \rrbracket \times \llbracket X \rrbracket) \xrightarrow{\mathsf{e}}$$

$$\langle \mathrm{time}\, \Gamma' \rangle (\llbracket \Gamma \rrbracket \times \llbracket X \rrbracket) \xrightarrow{\varepsilon^{\langle\rangle}} \llbracket \Gamma \rrbracket \times \llbracket X \rrbracket \xrightarrow{\mathsf{snd}} \llbracket X \rrbracket$$

and **boxing** is interpreted **using the unit of $\langle \tau \rangle \dashv [\tau]$**

$$\llbracket \Gamma \vdash \mathsf{box}_\tau \; V : [\tau]\, X \rrbracket \stackrel{\text{def}}{=} \llbracket \Gamma \rrbracket \xrightarrow{\eta^{\dashv}} [\tau]\, (\langle \tau \rangle \llbracket \Gamma \rrbracket) \xrightarrow{[\tau]\, (\llbracket V \rrbracket)} [\tau]\, \llbracket X \rrbracket$$

# Denotational semantics: comp. terms

- **Seq. comp.** is interpreted **using** $\eta^{\dashv}$ **and** $\mathsf{str}^{T}$**-followed-by-**$\mu^{T}$

$$[\![\Gamma \vdash \mathsf{let}\ x = M\ \mathsf{in}\ N : Y\ !\ \tau + \tau']\!] \stackrel{\text{def}}{=}$$

$$[\![\Gamma]\!] \xrightarrow{\langle \eta^{\dashv}, [\![M]\!] \rangle} [\tau]\,(\langle \tau \rangle [\![\Gamma]\!]) \times T\,\tau\,[\![X]\!] \xrightarrow{\mathsf{str}^{T}}$$

$$T\,\tau\,(\langle \tau \rangle [\![\Gamma]\!] \times [\![X]\!]) \xrightarrow{T\,\tau\,([\![N]\!])}$$

$$T\,\tau\,(T\,\tau'\,[\![Y]\!]) \xrightarrow{\mu^{T}} T\,(\tau + \tau')\,[\![Y]\!]$$

# Denotational semantics: comp. terms

- **Seq. comp.** is interpreted **using $\eta^{\dashv}$ and $\mathsf{str}^T$-followed-by-$\mu^T$**

$$[\![\Gamma \vdash \mathsf{let}\ x = M\ \mathsf{in}\ N : Y \ !\ \tau + \tau']\!] \quad \stackrel{\text{def}}{=}$$

$$[\![\Gamma]\!] \xrightarrow{\langle \eta^{\dashv}, [\![M]\!] \rangle} [\tau]\,(\langle \tau \rangle [\![\Gamma]\!]) \times T\,\tau\,[\![X]\!] \xrightarrow{\mathsf{str}^T}$$

$$T\,\tau\,(\langle \tau \rangle [\![\Gamma]\!] \times [\![X]\!]) \xrightarrow{T\,\tau\,([\![N]\!])}$$

$$T\,\tau\,(T\,\tau'\,[\![Y]\!]) \xrightarrow{\mu^T} T\,(\tau + \tau')\,[\![Y]\!]$$

and **unboxing** is interpreted **using $|-|_\tau \dashv \langle \tau \rangle$ and $\langle \tau \rangle \dashv [\tau]$**

$$[\![\Gamma \vdash \mathsf{unbox}_\tau\ V\ \mathsf{as}\ x\ \mathsf{in}\ N : Y \ !\ \tau']\!] \quad \stackrel{\text{def}}{=}$$

$$[\![\Gamma]\!] \xrightarrow{\langle \mathsf{id}, \mathsf{e}' \rangle} [\![\Gamma]\!] \times \langle \tau \rangle ([\![\,|\Gamma|_\tau]\!]) \xrightarrow{\mathsf{id} \times \langle \tau \rangle ([\![V]\!])}$$

$$[\![\Gamma]\!] \times \langle \tau \rangle ([\tau]\,[\![X]\!]) \xrightarrow{\mathsf{id} \times \varepsilon^{\dashv}} [\![\Gamma]\!] \times [\![X]\!] \xrightarrow{[\![N]\!]} T\,\tau'\,[\![Y]\!]$$

# Denotational semantics: soundness

- The **soundness theorem**

$$\Gamma \vdash I \equiv J \qquad \text{implies} \qquad [\![\Gamma \vdash I]\!] \equiv [\![\Gamma \vdash J]\!]$$

  is proved

  - by **unsurprising induction** on given derivations
  - by using the **categorical structure** we required above
  - by proving **semantic renaming and substitution lemmas**
  - by **relating syntactic renamings with semantic morphisms**

# Denotational semantics: soundness

- The **soundness theorem**

$$\Gamma \vdash I \equiv J \qquad \text{implies} \qquad [\![\Gamma \vdash I]\!] \equiv [\![\Gamma \vdash J]\!]$$

  is proved

  - by **unsurprising induction** on given derivations
  - by using the **categorical structure** we required above
  - by proving **semantic renaming and substitution lemmas**
  - by **relating syntactic renamings with semantic morphisms**

- The **completeness theorem**

$$[\![\Gamma \vdash I]\!] \equiv [\![\Gamma \vdash J]\!] \text{ in all models} \qquad \text{implies} \qquad \Gamma \vdash I \equiv J$$

  **is left for future work** (e.g., when sub-eff. question is resolved)

# Conclusion

# Conclusion

- **Temporal resources** can be naturally captured using
  - **modal temporal resource types** $[\tau]\, X$
  - **context modalities** $\Gamma, \langle\, \tau\, \rangle$
  - with a **time-graded variant of Fitch-style presentation**
  - with a **temporally aware type-and-effect system**
  - with a **natural category-theoretic semantics**

- **Draft paper**: When programs have to watch paint dry

  https://arxiv.org/abs/2210.07738

- (Work in progress) **Agda formalisation**

  https://github.com/danelahman/temporal-resources

# Some ongoing/future work directions

- **Sub-effecting**
  - as sub-effecting $M$ = all-possible-ways-to-insert-delays-into-$M$?
- **(Primitive) recursion**
  - grade of $\mathsf{rec}\ V\ M_z\ x.k.M_s$ computed by iteration/recursion
  - $M_z$ and $M_s$ being temporally aware depending on iteration no.
  - leads to needing type dependency (on $V$s being recursed on)
- **Generalising gradings**
  - other $(\mathbb{N}, 0, +, \dot{-}, \leqslant)$-like structures, e.g., (sets of) traces or states
  - different structures, e.g., as $\Gamma, \langle\, \tau(\mathsf{trace}) \,\rangle, x : X \vdash N : Y \,!\, \mathsf{trace}'$
  - maybe more generally as $\Gamma, \langle\, \tau(\Gamma, \mathsf{trace}) \,\rangle, x : X \vdash N : Y \,!\, \mathsf{trace}'$
- **Expiring resources**
  - where resources are usable only for an interval, e.g., as $[\,\tau, \tau'\,]\, X$