Revisiting Iso-Recursive Subtyping

Yaoda Zhou, Jinxu Zhao and Bruno C. d. S. Oliveira

In OOPSLA'20, TOPLAS'22

For SPLASH'22 Covid Time Papers

The University of Hong Kong

Recursive types

- Recursive types $\mu\alpha \,.\, \tau$ are used to represent recursive data structures like sequences or trees.

- Binary tree with integer leaves:

  $$\text{Tree} \triangleq \text{Int} + (\text{Tree} \times \text{Tree})$$

- Lists of integers:

  $$\text{List} \triangleq \text{Unit} + (\text{Int} \times \text{List})$$

What is the relation between the type $\mu\alpha\,.\,\tau$ and its one-step unfolding $[\alpha \mapsto \mu\alpha\,.\,\tau]\;\tau$?

equi-recursive:

$$\mu\alpha\,.\,\tau \qquad\qquad \triangleq \qquad\qquad [\alpha \mapsto \mu\alpha\,.\,\tau]\;\tau$$

iso-recursive:

$$\begin{tikzcd}
\mu\alpha\,.\,\tau \arrow[rr, bend left, "\mathsf{unfold}\ [\mu\alpha\,.\,\tau] : \mu\alpha\,.\,\tau \to [\alpha \mapsto \mu\alpha\,.\,\tau]\;\tau"] & & {[\alpha \mapsto \mu\alpha\,.\,\tau]\;\tau} \arrow[ll, bend left, "\mathsf{fold}\ [\mu\alpha\,.\,\tau] : [\alpha \mapsto \mu\alpha\,.\,\tau]\;\tau \to \mu\alpha\,.\,\tau"]
\end{tikzcd}$$

Most mainstream languages employ iso-recursive setting

```
data List = Nil | Cons Int List

map :: (Int -> Int) -> List -> List
map f Nil = Nil
map f (Cons x xs) = Cons (f x) (map f xs)
```

```
class Shape {
  int area() {...}
  boolean compareArea(Shape s) {
    return s.area() == area();
  }
  Shape clone() {return new Shape();}
}
```

Fold:	Use of one of the constructors	Class definition
Unfold:	Pattern matching for list	Method invoking

A natural question: can we define subtyping relations between recursive types?

The Amber rules

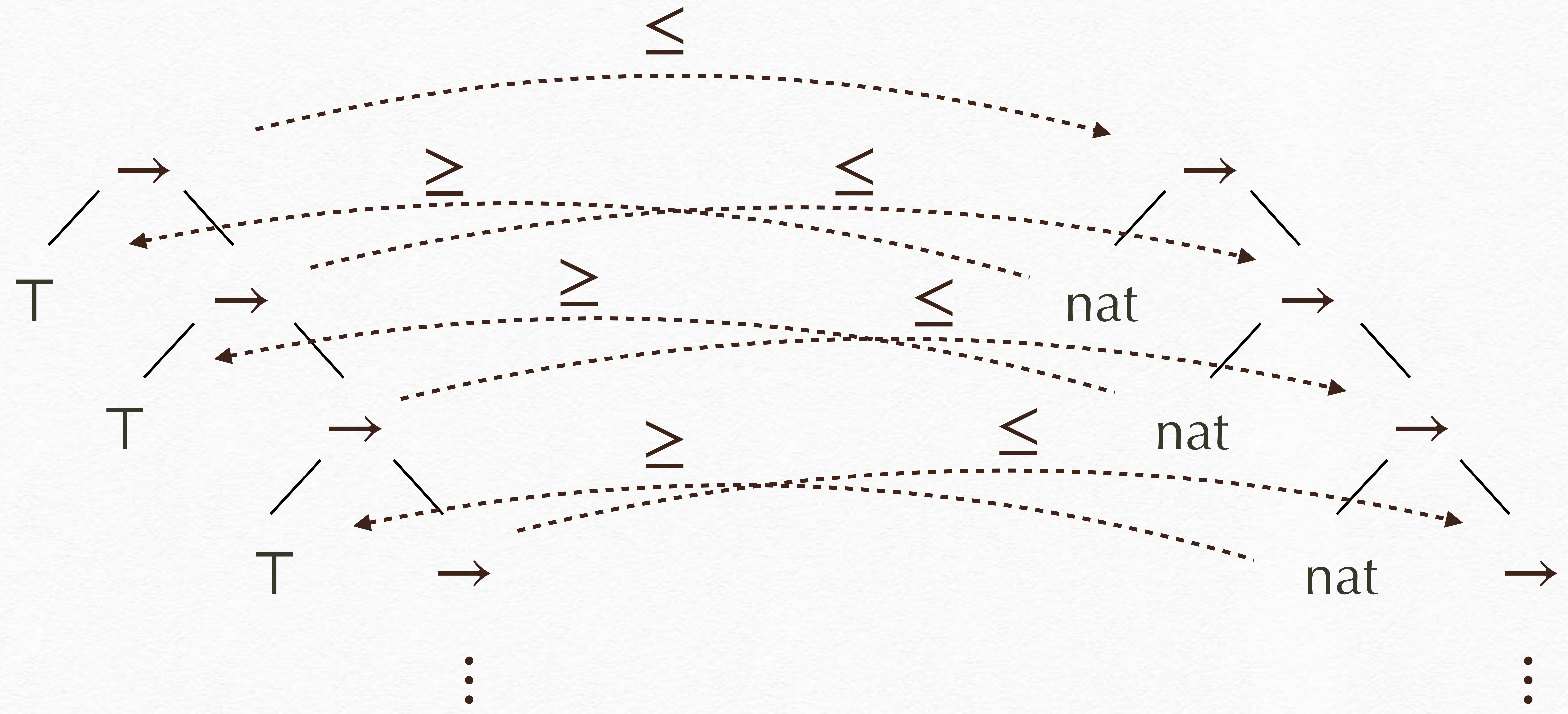
- First (briefly) mentioned in “Amber” in 1985 by Luca Cardelli
- The most famous comprehensive study of recursive subtyping
- “***Subtyping recursive types”***, in TOPLAS 1993, by Roberto M. Amadio and Luca Cardelli
- using an equi-recursive treatment of recursive types in λ-calculus

Tree view

$\mu\alpha\,.\,\top \to \alpha$

$\mu\alpha\,.\,\mathsf{nat} \to \alpha$

$\mu\alpha\,.\,\mathsf{nat} \to (\mu\alpha\,.\,\mathsf{nat} \to \alpha)$



Unrolling to the limit, then subtying for the whole tree

Equi-recursive Amber rules

$$\frac{\tau \triangleq \sigma}{\Gamma \vdash \tau \leq \sigma}\text{Amber-refl}$$

$$\frac{\Gamma \vdash \tau \leq \phi \quad \Gamma \vdash \phi \leq \sigma}{\Gamma \vdash \tau \leq \sigma}\text{Amber-trans}$$

$$\frac{\alpha \leq \beta \in \Gamma}{\Gamma \vdash \alpha \leq \beta}\text{Amber-assump}$$

$$\frac{\Gamma, \alpha \leq \beta \vdash \tau \leq \sigma}{\Gamma \vdash \mu\alpha\,.\,\tau \leq \mu\beta\,.\,\sigma}\text{Amber-rec}$$

E.g. $\mu\alpha\,.\,\text{nat} \to \alpha \triangleq \mu\alpha\,.\,\text{nat} \to \text{nat} \to \alpha$

$\mu\alpha\,.\,\text{nat} \to \alpha \triangleq \mu\alpha\,.\,\text{nat} \to (\mu\alpha\,.\,\text{nat} \to \alpha)$

Iso-recursive Amber rules

$$\frac{\tau = \sigma}{\Gamma \vdash \tau \leq \sigma}\text{Amber-refl}$$

$$\frac{\Gamma \vdash \tau \leq \phi \quad \Gamma \vdash \phi \leq \sigma}{\Gamma \vdash \tau \leq \sigma}\text{Amber-trans}$$

$$\frac{\alpha \leq \beta \in \Gamma}{\Gamma \vdash \alpha \leq \beta}\text{Amber-assump}$$

$$\frac{\Gamma, \alpha \leq \beta \vdash \tau \leq \sigma}{\Gamma \vdash \mu\alpha\,.\,\tau \leq \mu\beta\,.\,\sigma}\text{Amber-rec}$$

Drawback of Iso-recursive Amber rules

- Iso-recursive Amber rules are simple, fast, easy to implement, but:
  - Reflexivity cannot be eliminated.

$$\frac{\alpha \le \beta \vdash \beta \le \alpha (\text{fail!}) \qquad \ldots}{\vdash \mu\alpha\,.\,\alpha \to \top \le \mu\beta\,.\,\beta \to \top}$$

  - Finding an algorithmic formulation: transitivity elimination is non-trivial.
  - Proofs of transitivity and other lemmas are hard.
  - Non-modularity of the proofs.

Iso-recursive subtyping should

- Fact 1: Any type should be a subtype of itself
  - $\mu\alpha\,.\,\alpha \to \alpha \leq \mu\alpha\,.\,\alpha \to \alpha$
  - $\mu\alpha\,.\,\alpha \to \text{nat} \leq \mu\alpha\,.\,\alpha \to \text{nat}$
  - $\mu\alpha\,.\,\text{nat} \to \alpha \leq \mu\alpha\,.\,\text{nat} \to \alpha$
- Fact 2: Positive subtyping should be accepted
  - $\mu\alpha\,.\,\top \to \alpha \leq \mu\alpha\,.\,\text{nat} \to \alpha$

Iso-recursive subtyping should

- Fact 3: Many forms of negative subtyping should be rejected
  - $\mu\alpha\,.\,\alpha \to \text{nat} \nleq \mu\alpha\,.\,\alpha \to \top$
- Unfolding lemma:

  If $\mu\alpha\,.\,\sigma \leq \mu\alpha\,.\,\tau$ then $[\alpha \mapsto \mu\alpha\,.\,\sigma]\;\sigma \leq [\alpha \mapsto \mu\alpha\,.\,\tau]\;\tau$

Novel specification

- We propose a new formal specification for iso-recursive subtyping, called finite unfolding rule:

$$\frac{\Gamma, \alpha \vdash [\alpha \mapsto \tau]^n\ \tau \leq [\alpha \mapsto \sigma]^n\ \sigma \quad \forall n = 1, 2, \cdots, \infty}{\Gamma \vdash \mu\alpha\,.\,\tau \leq \mu\alpha\,.\,\sigma}\ \text{S-rec}$$

where $\quad [\alpha \mapsto \tau]^n\ \tau \triangleq \underbrace{[\alpha \mapsto \tau][\alpha \mapsto \tau]\cdots[\alpha \mapsto \tau]}_{n-1}\ \tau$

Finite unfolding

Two recursive types are subtype if and only if

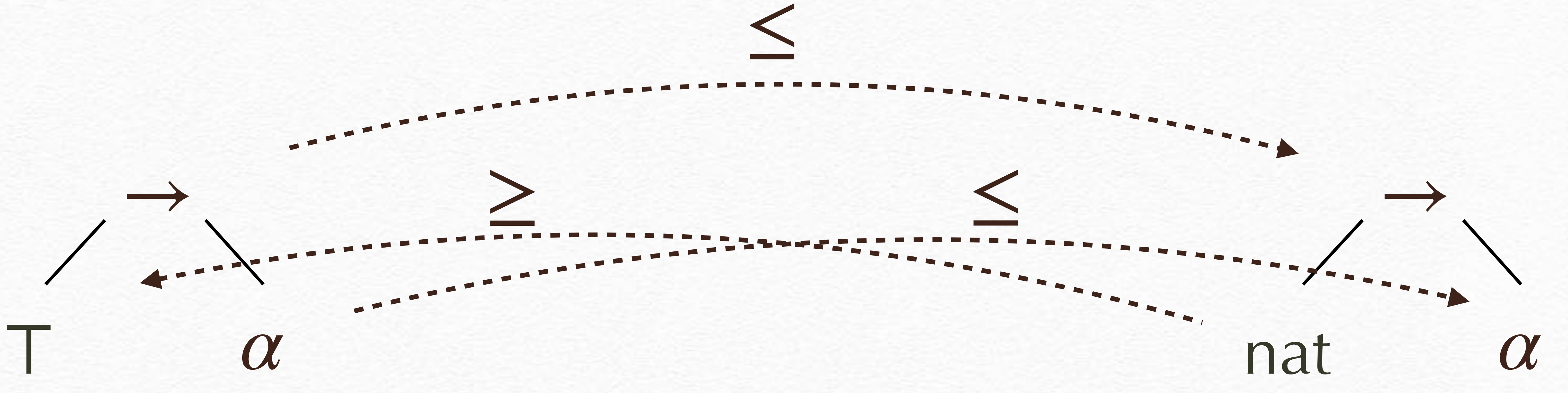
<u>*all their* n*-times finite unfoldings are subtypes*</u>

for any positive integer n

Tree view — for iso-recursive subtyping

$\mu\alpha\,.\,\top \to \alpha$ $\qquad\qquad$ $\mu\alpha\,.\,\mathrm{nat} \to \alpha$

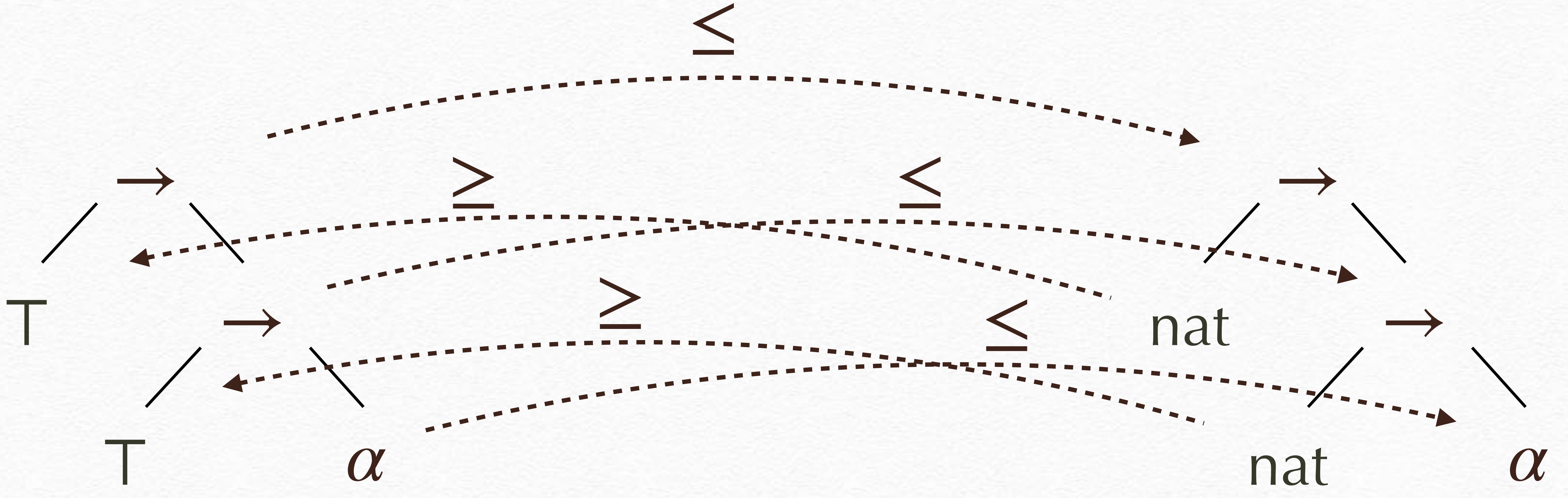
Rank 1:



Tree view — for iso-recursive subtyping

$\mu\alpha\,.\,\mathsf{T} \to \alpha$ $\qquad\qquad$ $\mu\alpha\,.\,\mathsf{nat} \to \alpha$

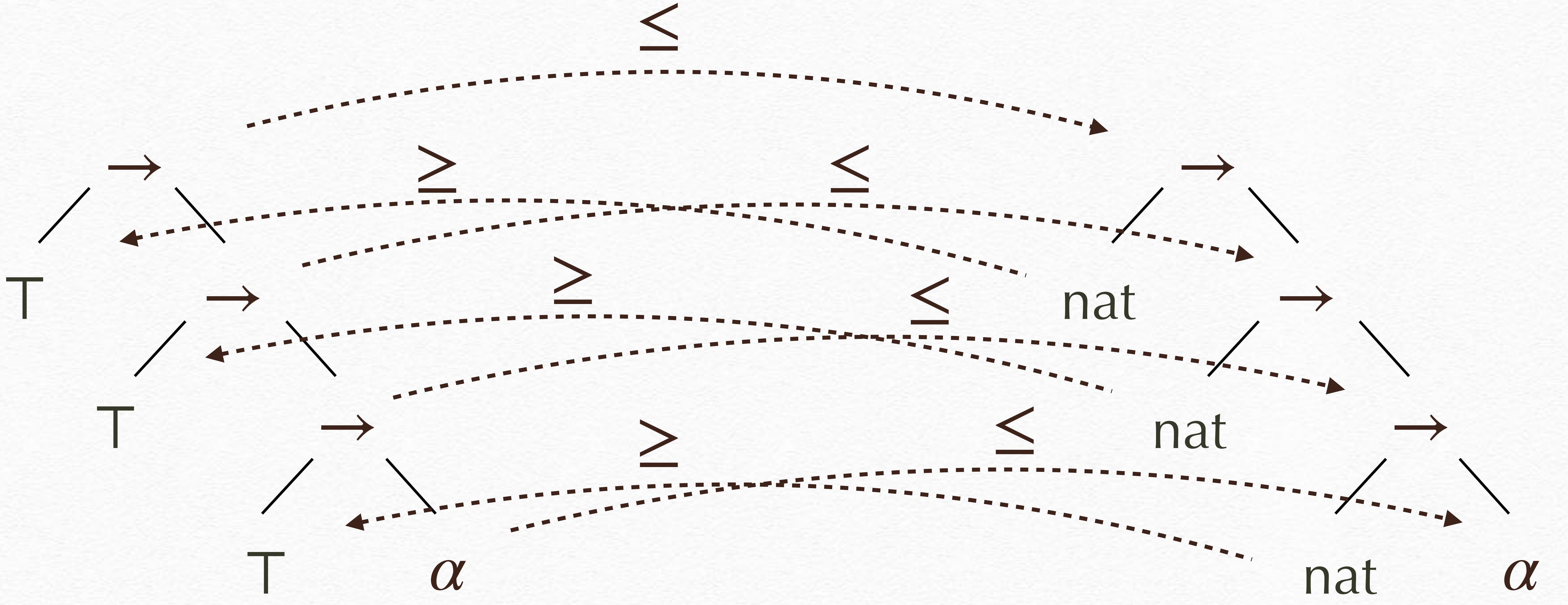
Rank 2:



Tree view — for iso-recursive subtyping

$\mu\alpha\,.\,\top \to \alpha$ $\qquad\qquad$ $\mu\alpha\,.\,\mathsf{nat} \to \alpha$

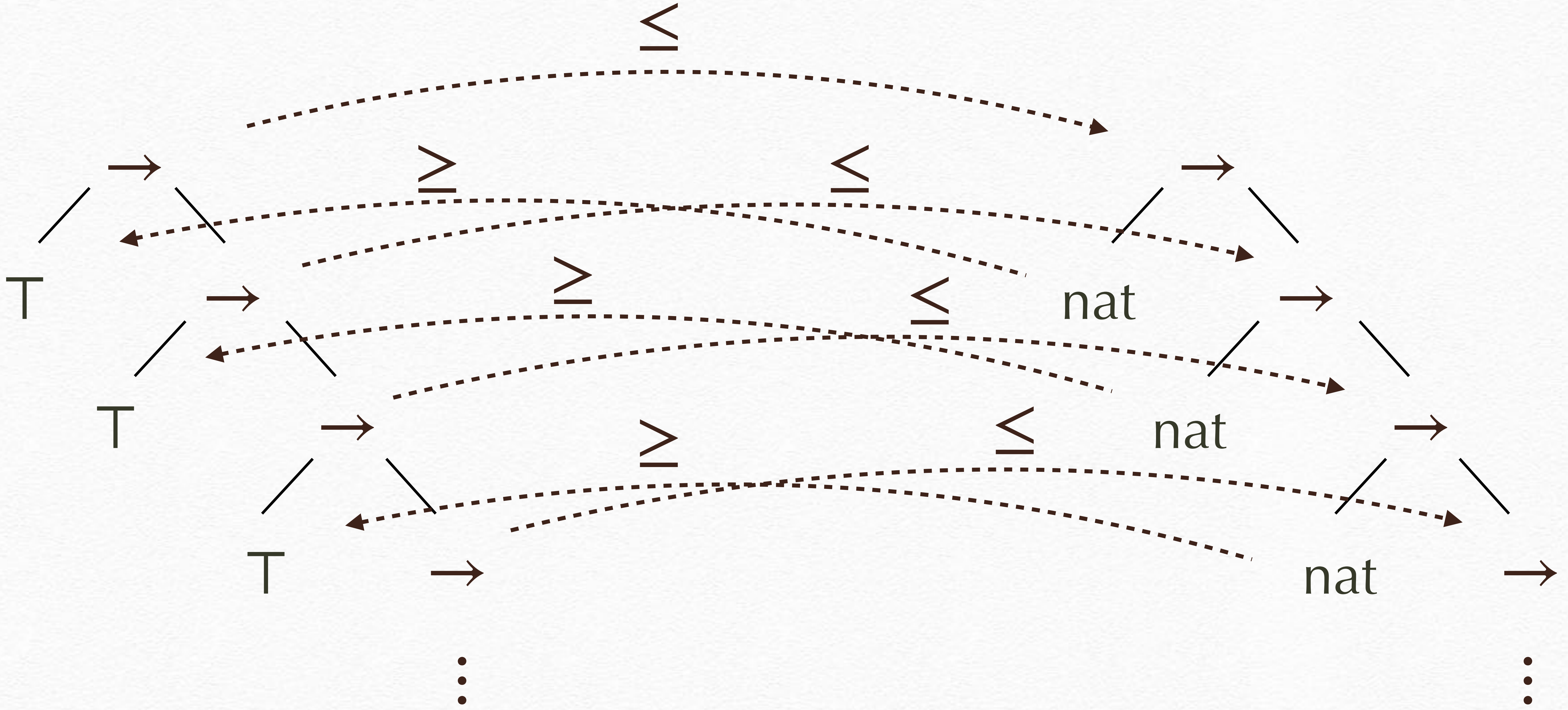
Rank 3:



Tree view — for iso-recursive subtyping

$\mu\alpha\,.\,\top \to \alpha$ $\qquad\qquad$ $\mu\alpha\,.\,\mathsf{nat} \to \alpha$

Rank n:



In a tree view

- Equi-recursive:

  Unrolling to the limit, then subtying for <u>the whole tree</u>

- Iso-recursive:

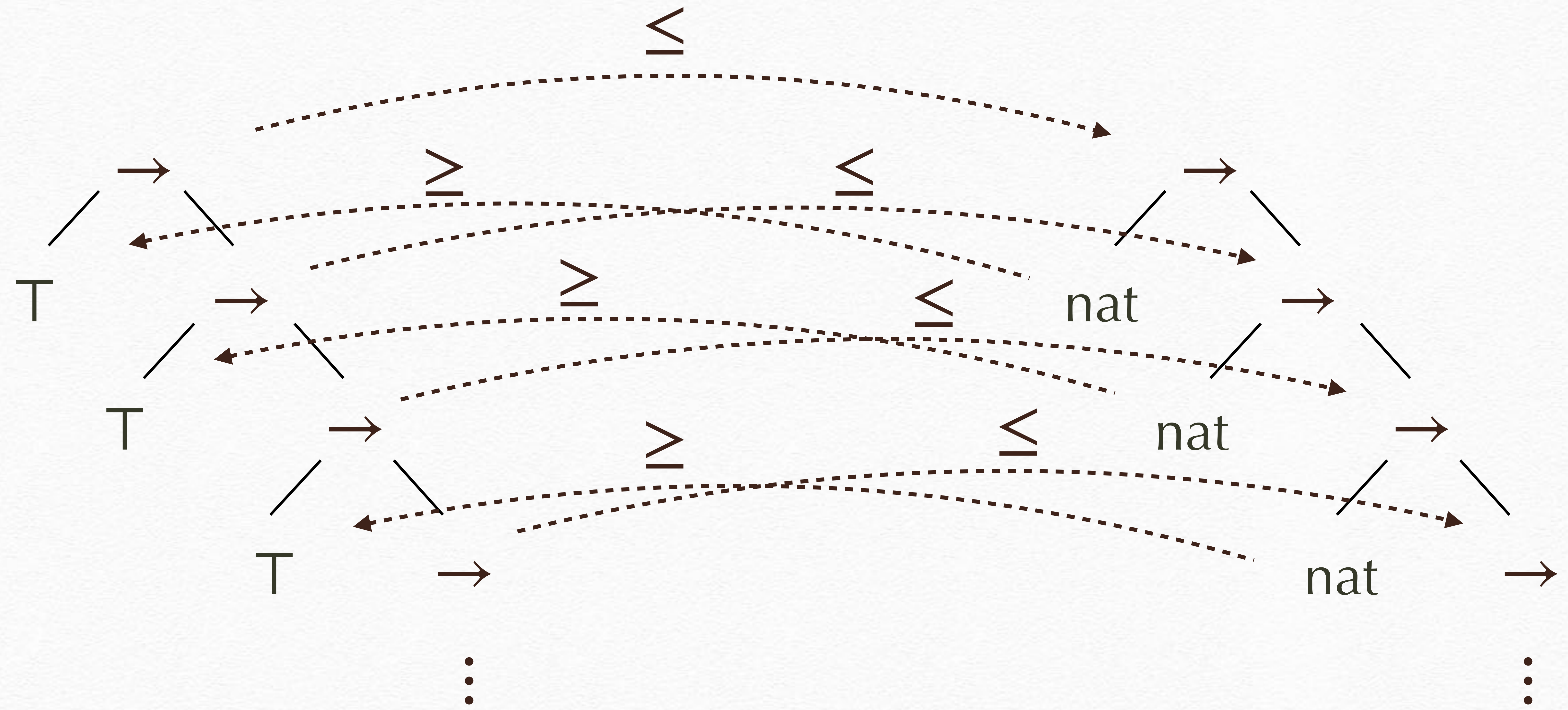
  Unrolling to the limit, then subtying for <u>all rank-n subtrees</u>

Tree model — comparison

$\mu\alpha\,.\,\top \to \alpha$

✔ $\mu\alpha\,.\,\mathsf{nat} \to \alpha$

✘ $\mu\alpha\,.\,\mathsf{nat} \to (\mu\alpha\,.\,\mathsf{nat} \to \alpha)$



Advantages of new specification

- Reflexivity is derivable.
- Transitivity (and other lemmas) is easy to prove.
- Enable modular proofs.
- Are applicable to non-antisymmetric subtyping relations.

Algorithmic Iso-Recursive Subtyping

- Our new specification is not algorithmic, because it need to test <u>infinite</u> subtyping

- Checking rank-n (n > 2) subtrees repeats checking rank-1 and rank-2 subtrees

- Double unfolding:

$$\frac{\Gamma, \alpha \vdash \tau \leq \sigma \quad \Gamma, \alpha \vdash [\alpha \mapsto \tau]\, \tau \leq [\alpha \mapsto \sigma]\, \sigma}{\Gamma \vdash \mu\alpha\,.\,\tau \leq \mu\alpha\,.\,\sigma}\text{S-double}$$

Nominal unfolding

- We use an extra label to help compare corresponding subtree
- We call it nominal unfolding:

$$\frac{\Gamma, \alpha \vdash [\alpha \mapsto \tau^{\alpha}]\ \tau \leq [\alpha \mapsto \sigma^{\alpha}]\ \sigma}{\Gamma \vdash \mu\alpha\,.\,\tau \leq \mu\alpha\,.\,\sigma}\text{S-nominal}$$

- The nominal unfolding is proven to have same expressiveness as double unfolding in a simple setting (like Simply Typed Lambda Calculus).
- The α in the labels is fresh and is different from α as the recursive variable

Weakly Positive Subtyping

- We prove that our new specification has **same expressiveness** as iso-recursive Amber rules

- Directly equivalence proof is difficult

- The key idea in this relation is to have a special rule for recursive types:

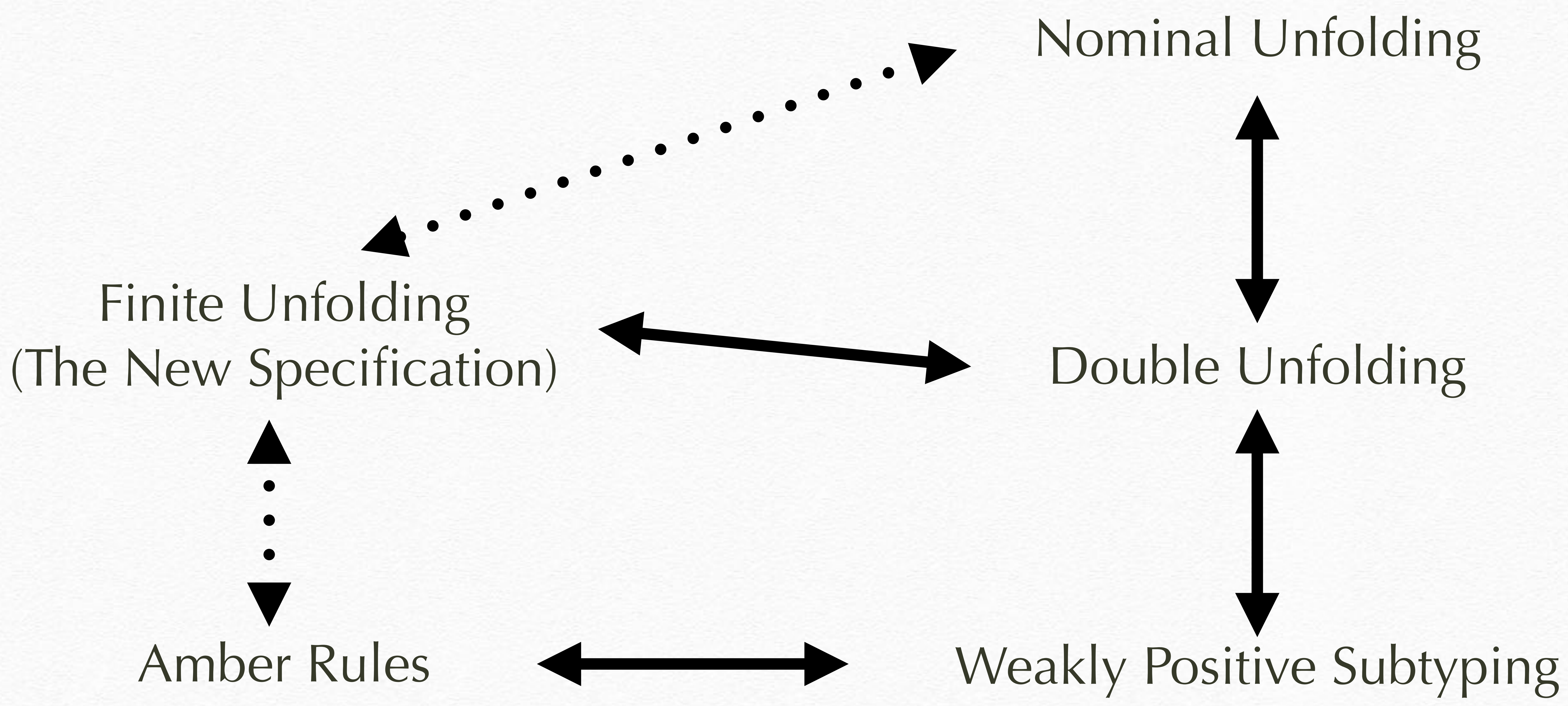
$$\frac{\Gamma, \alpha \vdash \tau \leq_{+} \sigma \quad \alpha \in_{+} \tau \leq \sigma}{\Gamma \vdash \mu\alpha\,.\,\tau \leq_{+} \mu\alpha\,.\,\sigma}$$

- $\alpha \in_m \tau \leq \sigma$ means α exists in positive(m is +) or negative(m is -) position in the weakly positive subtyping.

Weakly Positive Restriction

- $\alpha \in_+ \tau \leq \sigma$: we never find a contravariant subderivation $\alpha \leq \alpha$ for recursive types except for:
    - Equal types, i.e. two recursive types are equal.
        - E.g. $\mu\alpha . \alpha \to \text{nat} \leq \mu\alpha . \alpha \to \text{nat}$
    - The recursive type variable is a subtype of the top.
        - E.g. $\mu\alpha . \top \to \text{nat} \leq \mu\alpha . \alpha \to \text{nat}$

Equivalence of all formulations



Follow-up work

- With intersection types:
  - ***A Calculus with Recursive Types, Record Concatenation and Subtyping*** (Zhou et al. 2022, APLAS 2022)
- With bounded quantification:
  - ***Recursive Subtyping for All*** (Zhou et al. 2023, POPL 2023)

Thanks for listening!