Ugh! So Many Nontrivial Subtyping Relations...

But the definition of subtyping is actually very simple...

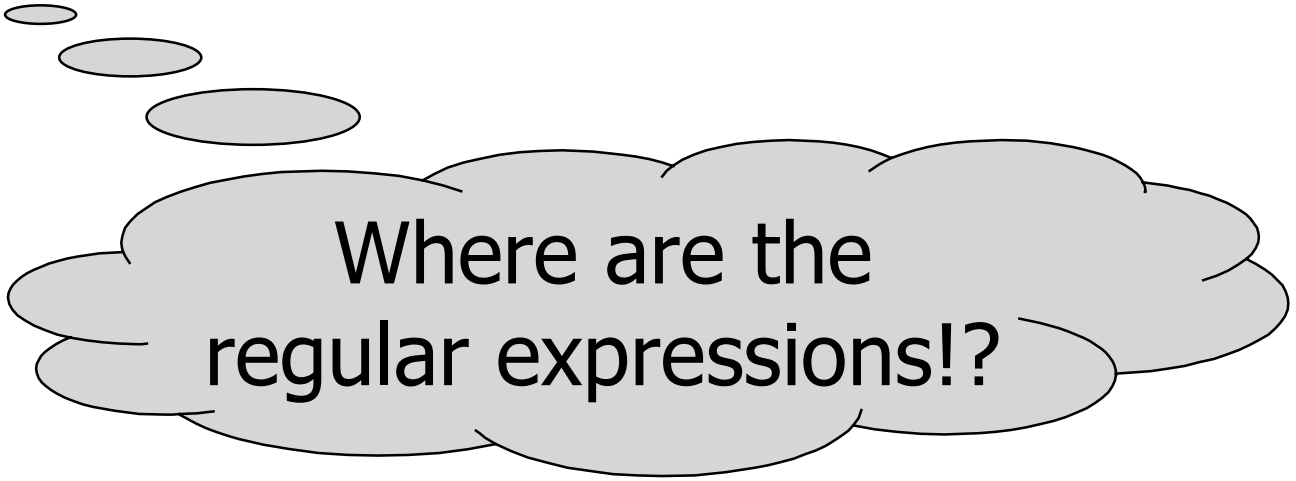
Syntax of Types

- Mutually recursive top-level type definitions:

  type X1=T1, X2=T2, ...

- Types:

  T ::= X | B | a[T] | T,T | () | T|T



Derived Forms

T? ≡ T | ()

T+ ≡ T, T*

T* ≡ X where X = T, X | ()

Semantics of Types

A type denotes a set of sequences:

$$[\![\ a[T]\ ]\!] \ = \ \{\ a[v]\ \mid\ v \in [\![\ T\ ]\!]\ \}$$

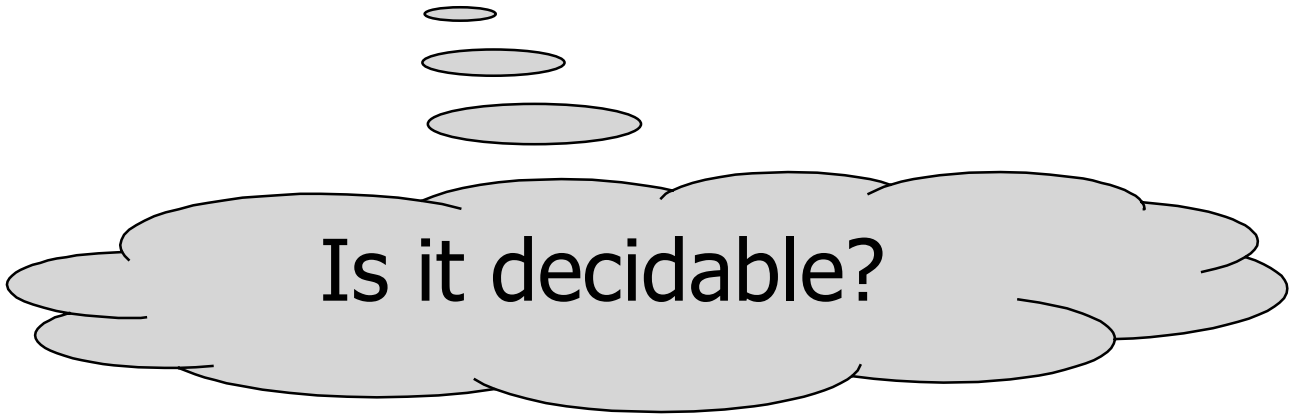
$$[\![\ T1,T2\ ]\!] \ = \ \{\ v1,v2\ \mid\ v1 \in [\![\ T1\ ]\!]\ v2 \in [\![\ T2\ ]\!]\ \}$$

$$[\![\ T1 \mid T2\ ]\!] \ = \ [\![\ T1\ ]\!] \cup [\![\ T2\ ]\!]$$

Interpret recursive types via least fixpoints

Definition of Subtyping

$$T <: U \quad \text{iff} \quad [\![T]\!] \subseteq [\![U]\!]$$



Subtyping Algorithm

Algorithm (Overview)

- Idea is similar to usual subtyping algorithm
  - Given two input types
  - Check if their top-most type constructors match
  - Go recursively to the subcomponents

- Substantially complicated by "untaggedness" of union types

Tagged (Disjoint) Sum Types

- Require a distinct tag on each component type
- Subtyping is easy

$$\frac{\begin{array}{ll} & B \text{ of } S <: C \text{ of } T \\ \text{or} & B \text{ of } S <: D \text{ of } U \end{array}}{B \text{ of } S <: C \text{ of } T \mid D \text{ of } U}$$

- Distinctness of tags C and D ensures that either
  - all values in (B of S) are in (C of U) or
  - all values in (B of S) are in (D of U)

Untagged Union Types

- Impose no restriction on tags
  - Multiple component types can have the same tags
  - Some component types can have no tag
- Subtyping is **not** analogous to tagged sums

$$\frac{a[S \mid T] <: a[S] \quad \text{or} \quad a[S \mid T] <: a[T]}{a[S \mid T] <: a[S] \mid a[T]}$$

- Some values in a[S | T] are in a[S] and others are in a[T]

Our Rule

R <: T and S <: U

or R <: V and S <: W

or R <: T and R <: V and S <: U | W

or R <: T | V and S <: U and S <: W

---

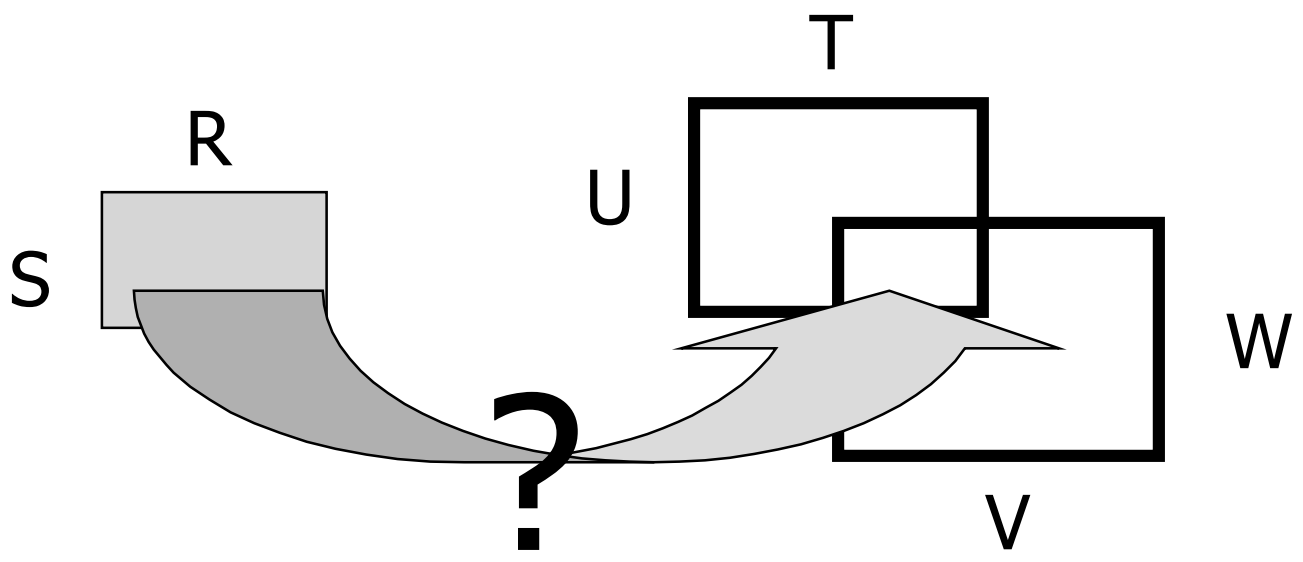
a[R], S <: a[T], U | a[V], W

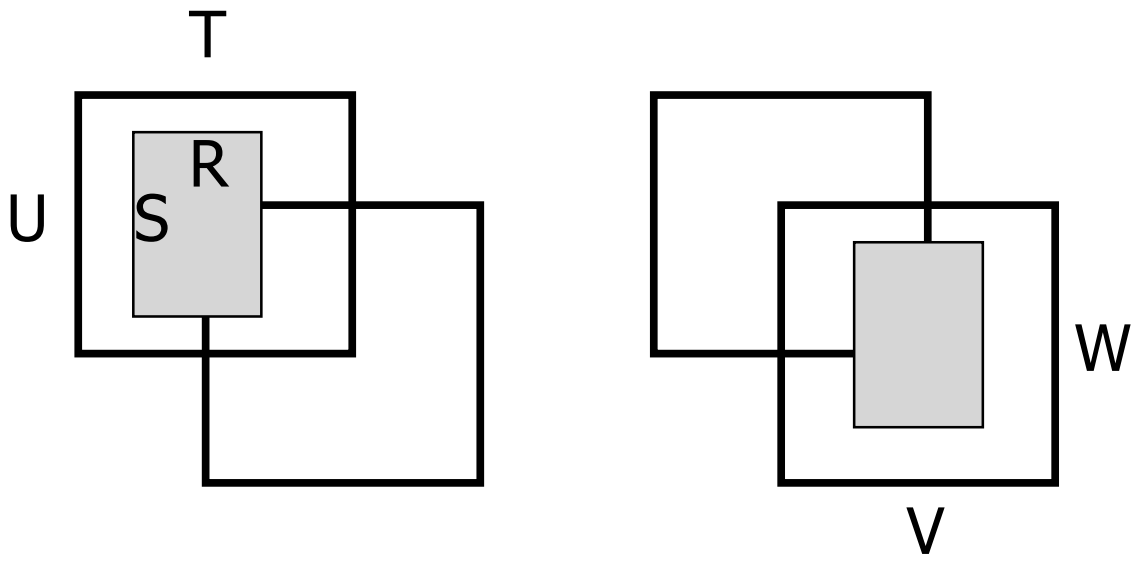
Or, more intutively...

2D-geometry Intuition

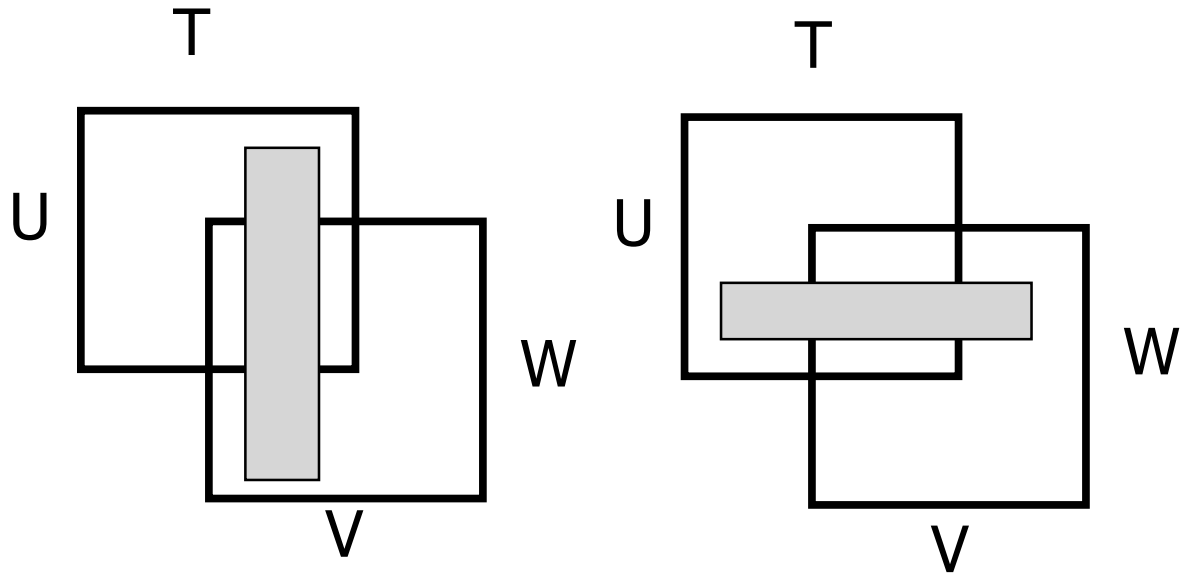
- (a[R], S) is isomorphic to cross product (R x S)
- Cross products can be depicted by rectangles

a[R], S <: a[T], U | a[V], W





These two cases are similar to the subtyping rule for tagged sum types



These two occur exactly because union operator is untagged, which can cause types to overlap