NPRG085: Theory of Programming Languages

Tuples, records and subtyping

Tomáš Petříček and Vít Šefl

petricek@d3s.mff.cuni.cz | sefl@ksvi.mff.cuni.cz

@tomasp.net | https://tomasp.net

Charles University

Demo

Can we formalise something useful?



Covariant arrays

TypeScript, Java and others



Dog can be used where Animal is expected



Dog is a subtype of Animal (`Dog <: Animal`)



Covariant arrays, i.e. `T1 <: T2` then `T1[] <: T2[]`



Conscious decision in TypeScript and Java



Not sound! Can we show this?

Charles University

Tuples and records

Extending the language



Tuples and records

Operations op := + | ≥

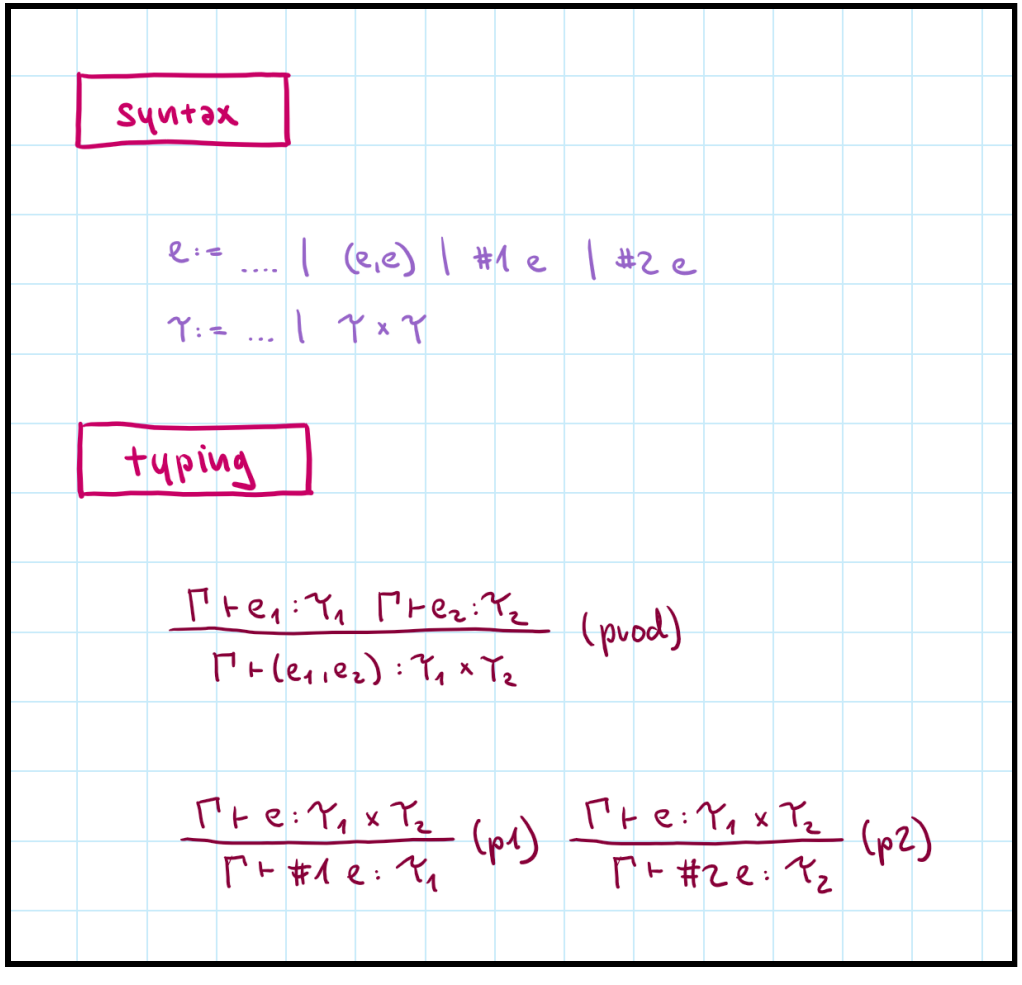
Expressions e := n | b | e op e
| if e then e else e
| while e do e
| l := e | !e
| skip | e ; e
| (e,e) | #1 e | #2 e (tuples)
| {lab = e, ..., lab = e} | #lab e (records)

Demo

Tuples and records in F#



Syntax and types

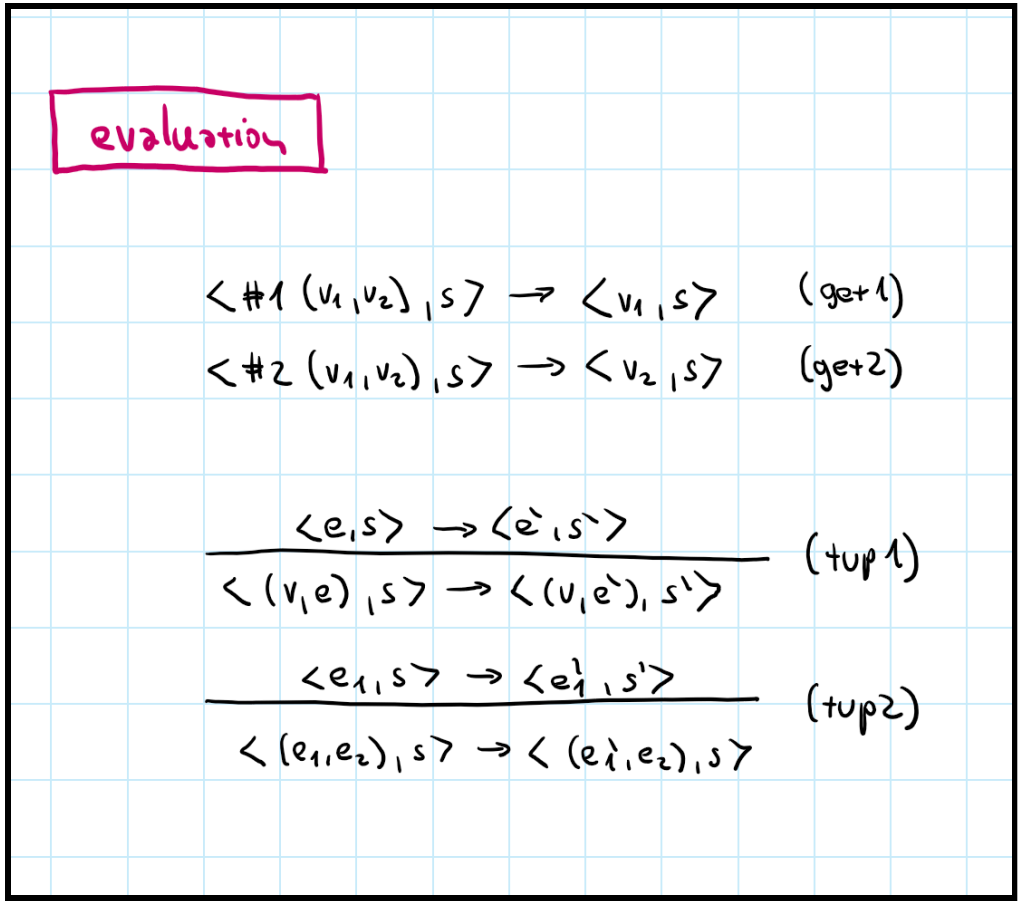


Element accessors (`fst` and `snd`) rather than pattern matching

Obvious typing rules!

Charles University

Evaluation rules



Eliminating destructor & constructor pair is a common pattern!

#1 of tuple of values
#2 of tuple of values

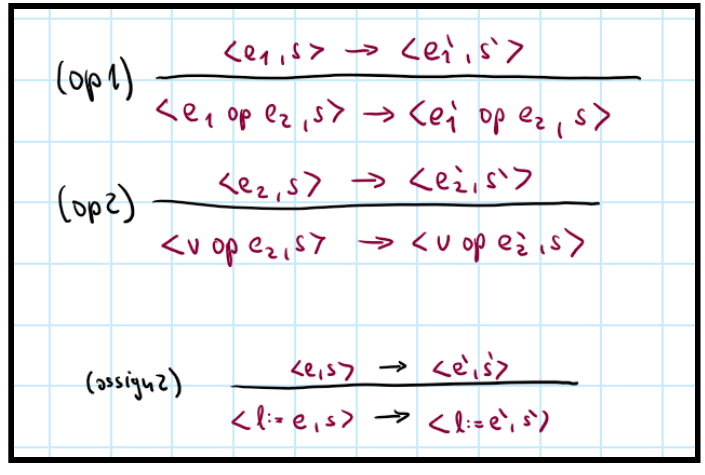
Specify left-to-right evaluation order of sub-expressions

Charles University

Evaluation contexts

Specifying evaluation order

- Expression reduces
- In a given syntactic form
- When sub-expression reduces

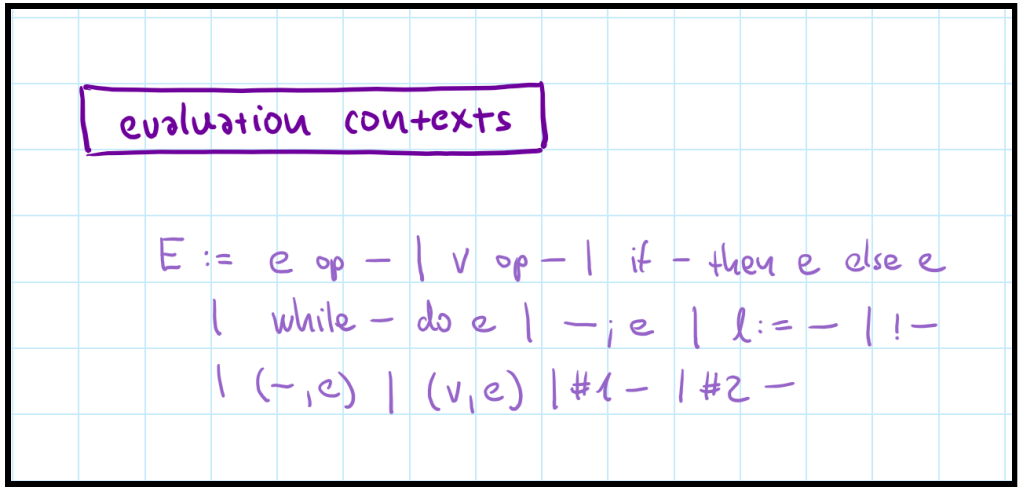


Use just a single rule

- Specify evaluation context $E[-]$
- Expression in a context $E[e]$
- E includes $(v + -)$ and $(- + e)$ but not $(e + -)$

Charles University

Evaluation context

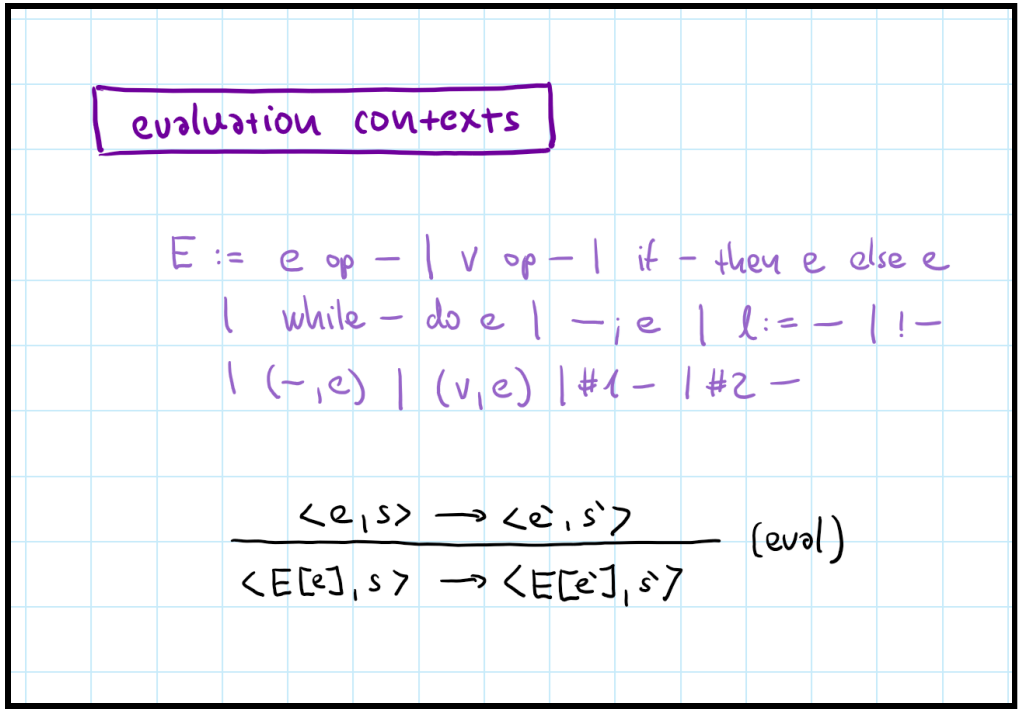


Left to right order

Specify all the positions where sub-expression can reduce

Charles University

Evaluation context



Left to right order

Specify all the positions where sub-expression can reduce

Single rule for sub-expression reduction!

Makes proofs harder

Charles University

```
// Structural types
// (not named in advance)
let pt : { x:number, y:number }
  = { x: 0, y: 0 }

// Type can be inferred
let pt = { x: 0, y: 0 }

// Field access checked
pt.x + pt.y
```

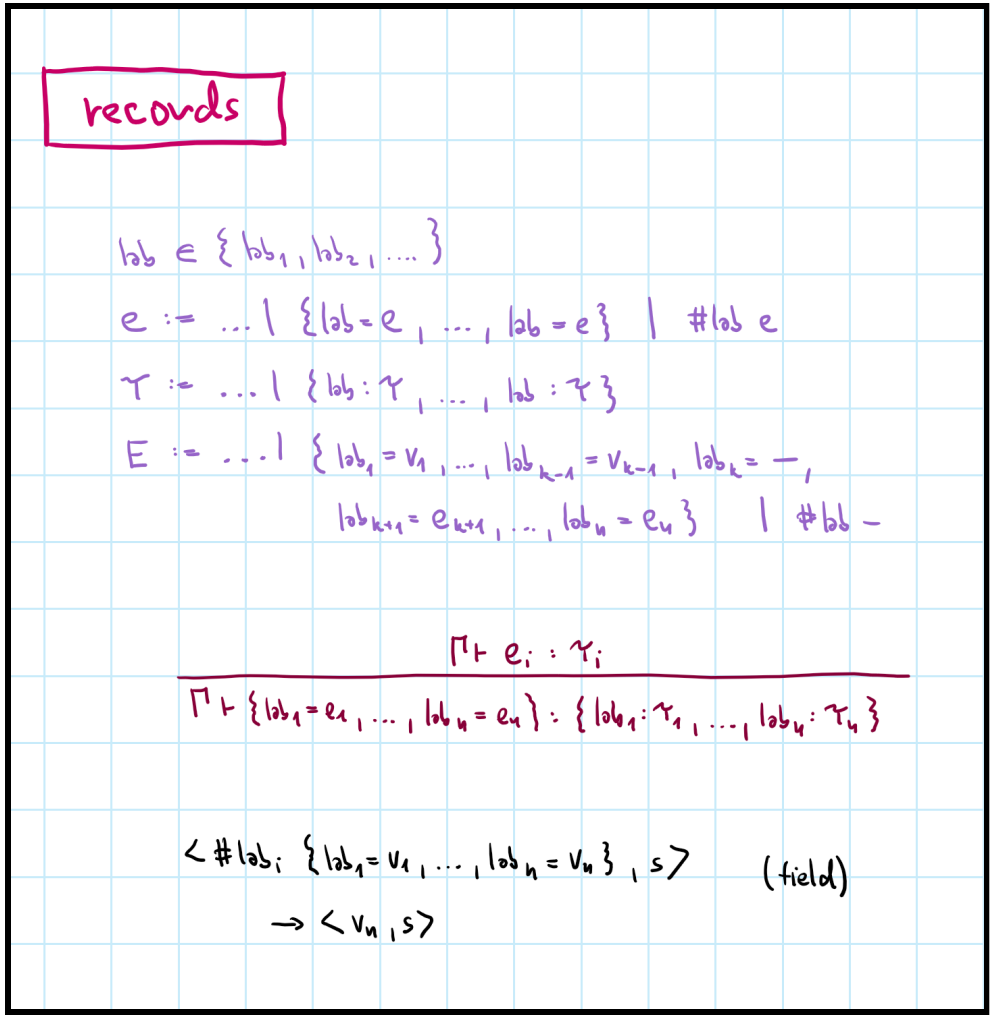
Records

Structural types
as in TypeScript

Nominal types
as in ML/F# or Java

Structural types are easier - no definitions in advance

Charles University



Adding records

Type syntax mirrors expression syntax

Type uses `lab:T`
Value uses `lab=e`

Destructor/constructor evaluation pattern

Sub-expression reduction for free!

Charles University

Subtyping

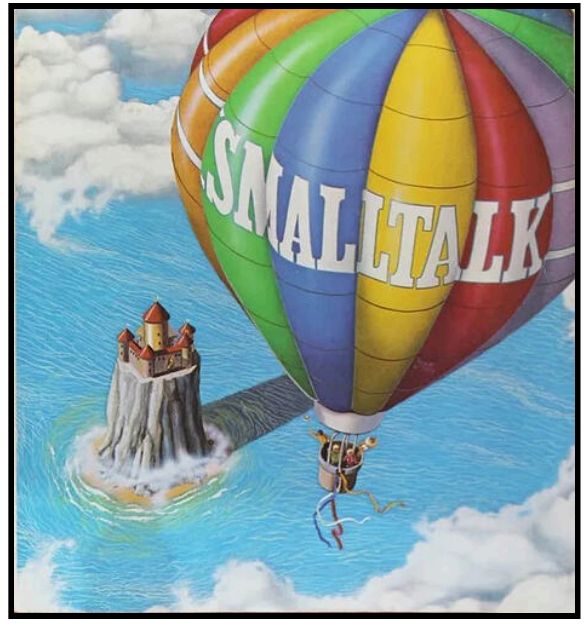
Formal model



Subtyping

Definition

- A subtype `S` can be used anywhere where a supertype `T` is expected
- Written as `S <: T`



In practice

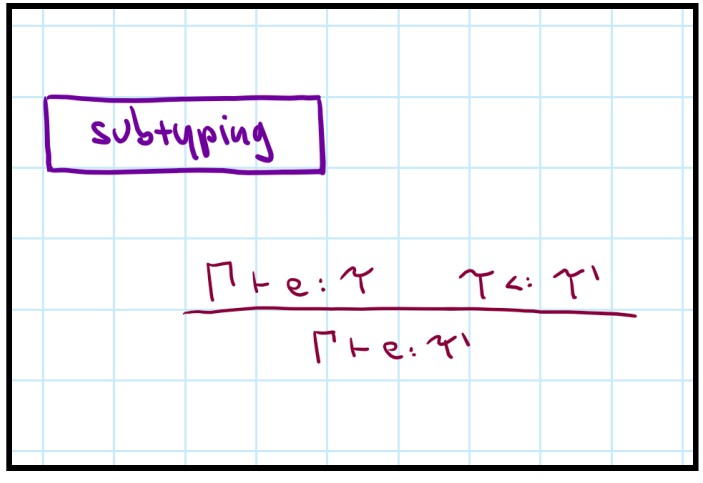
- Dynamic types - based on runtime
- Static types - subtyping relation
- Inheritance introduces subtyping
- Type structure, interfaces, etc.

Charles University

Extending type system

Easy to define

- Expression of subtype τ can be types using any supertype τ'

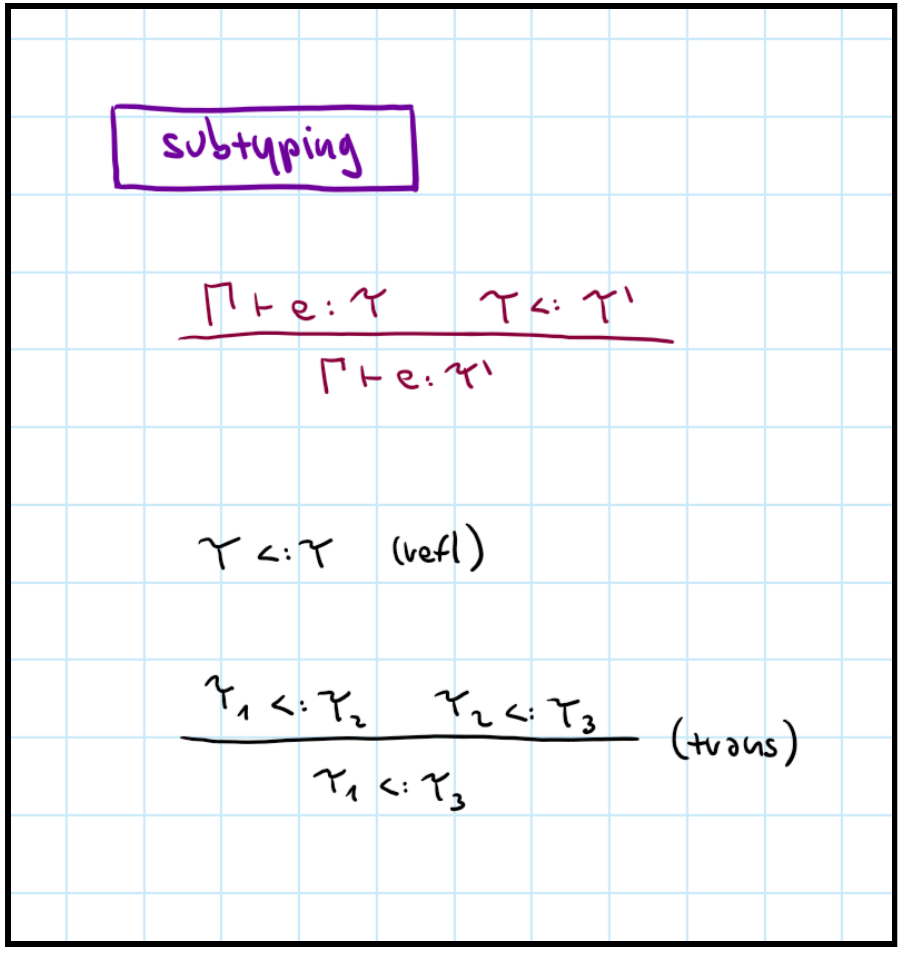


Breaks nice properties!

- Type preservation gets tricky
- Uniqueness of types no longer holds
- Not syntax-directed typing rule

Charles University

Subtyping rules



Typing judgement using subtyping relation

Derivation rules for subtyping - preorder

- Reflexive
- Transitive

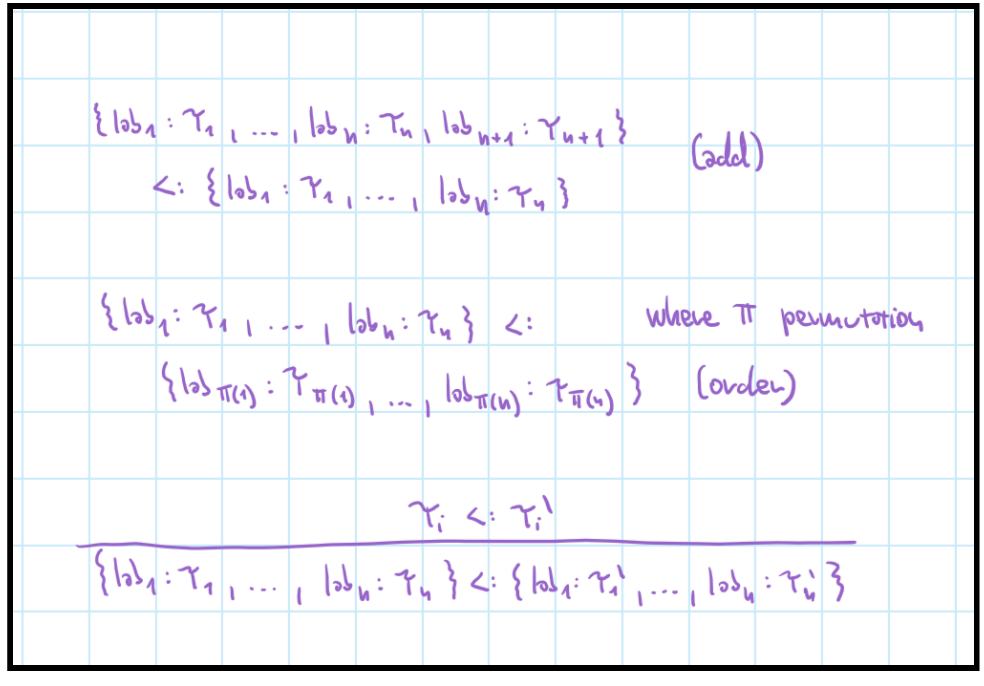
Neither symmetric nor anti-symmetric

Charles University

Subtyping for records

Three record rules

Additional field at the end
Reordering of fields
Subtypes of nested values



Primitive rules!

Often need a combination

Charles University

Which of these are true?

① $\{ \text{point} : \{ x : \text{num}, y : \text{num}, z : \text{num} \} \}$

$<: \{ \text{point} : \{ x : \text{num}, y : \text{num} \} \}$

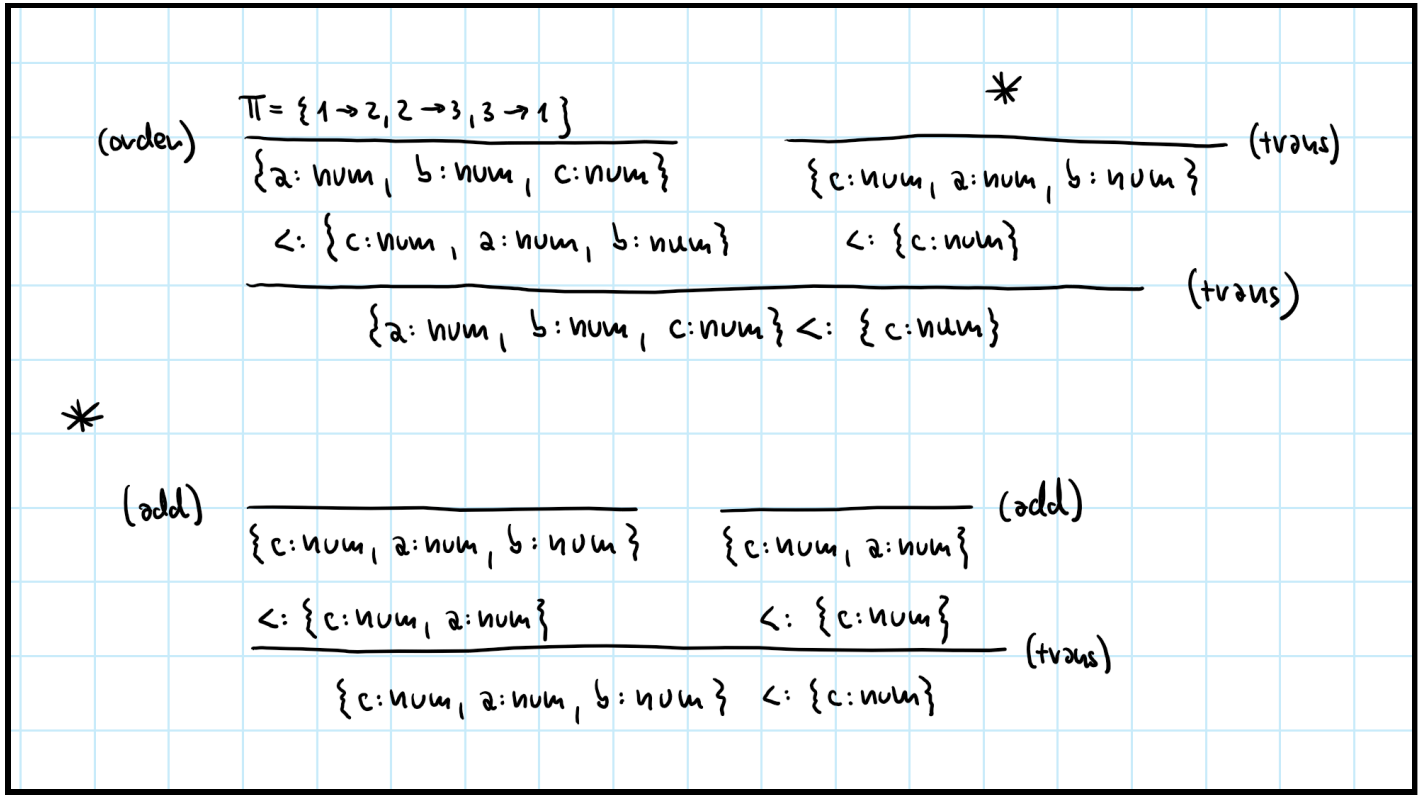
② $\{ v : \text{num}, b : \text{bool} \} <: \{ v : \text{num}, b : \text{num} \}$

③ $\{ a : \text{num}, b : \text{num}, c : \text{num} \}$

$<: \{ c : \text{num}, b : \text{num} \}$

Charles University

Writing subtyping derivation



Same as typing derivation or reduction derivation

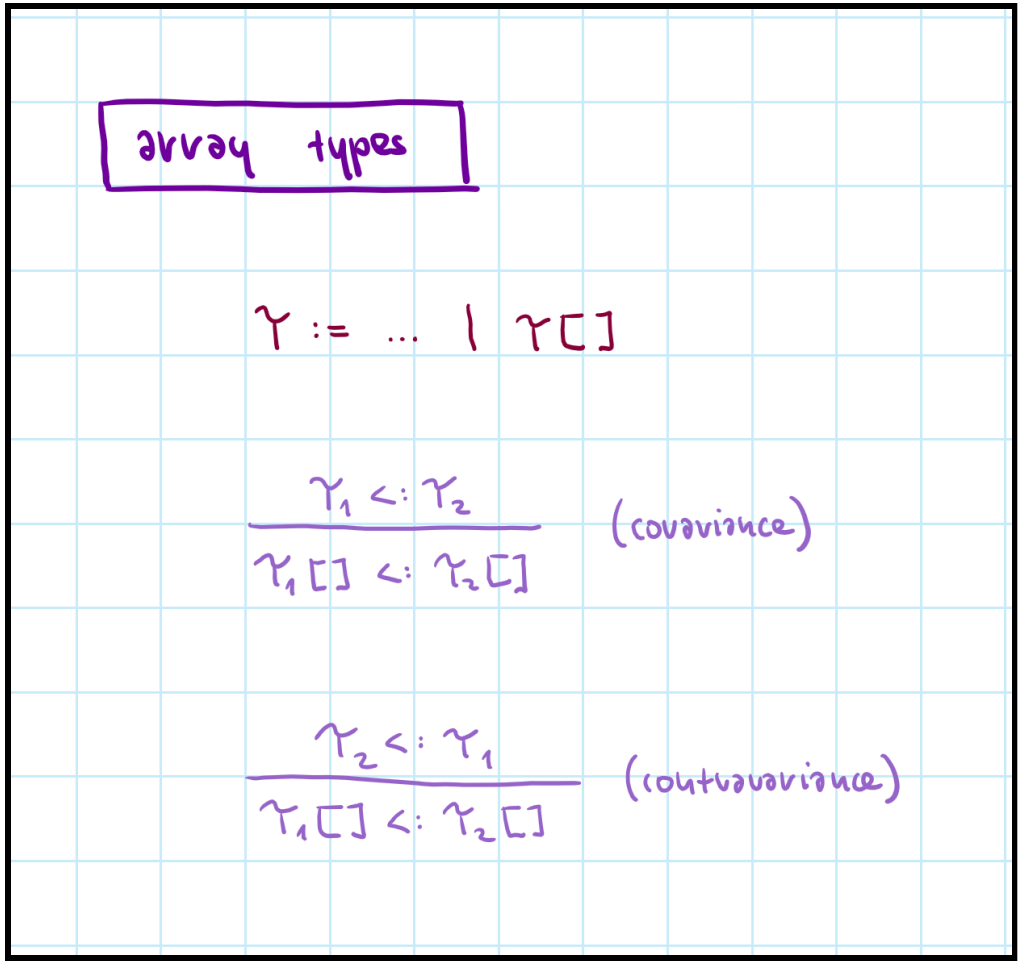
One relation can have multiple different derivations!

Arrays

Subtyping and arrays



TypeScript arrays



What subtyping rule is right for arrays?

- Covariance
- Contravariance

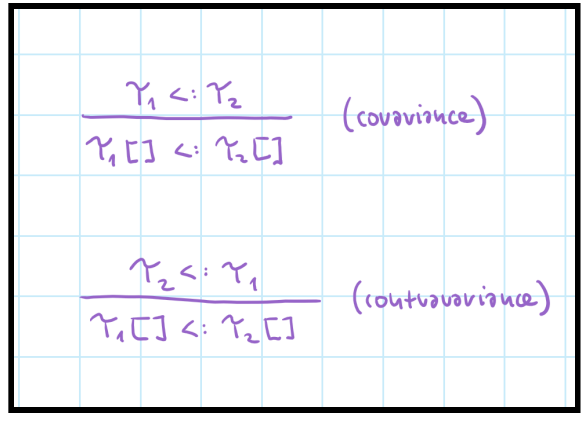
Both of them break!

Charles University

The trouble with arrays

How the rules break

Covariance - can put `Cat` in `Dog[]`
Contravariance - can get `Cat` from `Dog[]`



Would the proofs fail?

- Typing preservation would break!

Correct array handling

- Covariance for read-only access
- Contravariance for write-only access

Charles University

Conclusions

Tuples, records and subtyping



Conclusions

Tuples, records and subtyping

- → Reduction using destructor/constructor pattern
-  Subtyping as yet another kind of derivation

  Getting close to formalizing useful things!
-  Real-world languages are large and messy

Charles University