NPRG014

Advanced types in TypeScript

Guest lecture

Tomáš Petříček, 309 (3rd floor)

✉ petricek@d3s.mff.cuni.cz

➜ https://tomasp.net | @tomaspetricek

➜ https://d3s.mff.cuni.cz/people/tomaspetricek/

Where I'm coming from?

PhD, University of Cambridge
Types for context-aware programming

Microsoft Research Cambridge
F# and applied functional programming

The Alan Turing Institute, London
Expert and non-expert tools for data science

University of Kent, Canterbury
Programming systems and history

Charles
University

What is my quest?

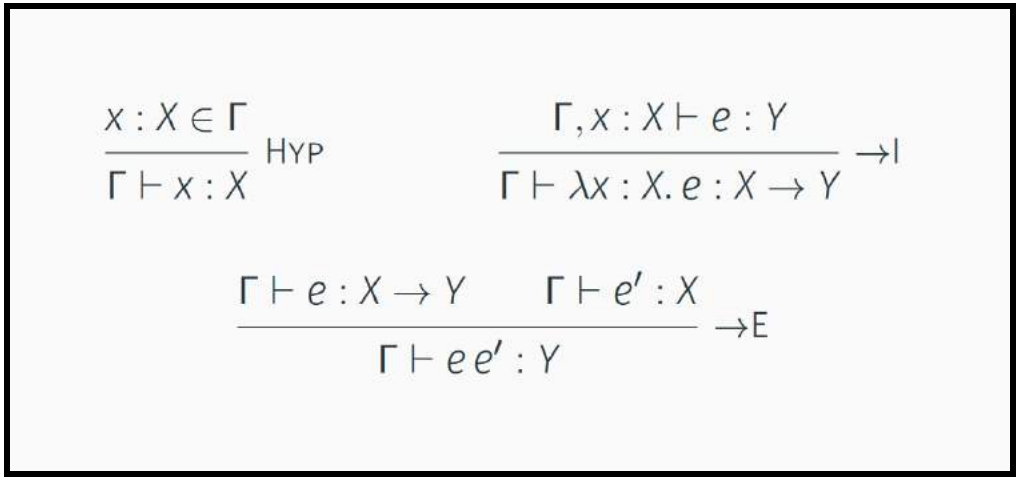
More programming language research in Prague

Ask about reading group!
Do a PL project or thesis!
Join us for a PhD?

Charles University

Language design

Different approaches



$$\frac{x : X \in \Gamma}{\Gamma \vdash x : X}\,\textsc{Hyp} \qquad \frac{\Gamma, x : X \vdash e : Y}{\Gamma \vdash \lambda x : X. e : X \to Y}\,{\to}\mathrm{I}$$

$$\frac{\Gamma \vdash e : X \to Y \qquad \Gamma \vdash e' : X}{\Gamma \vdash e\,e' : Y}\,{\to}\mathrm{E}$$

Formal properties

Language should be based on sound formal foundations!

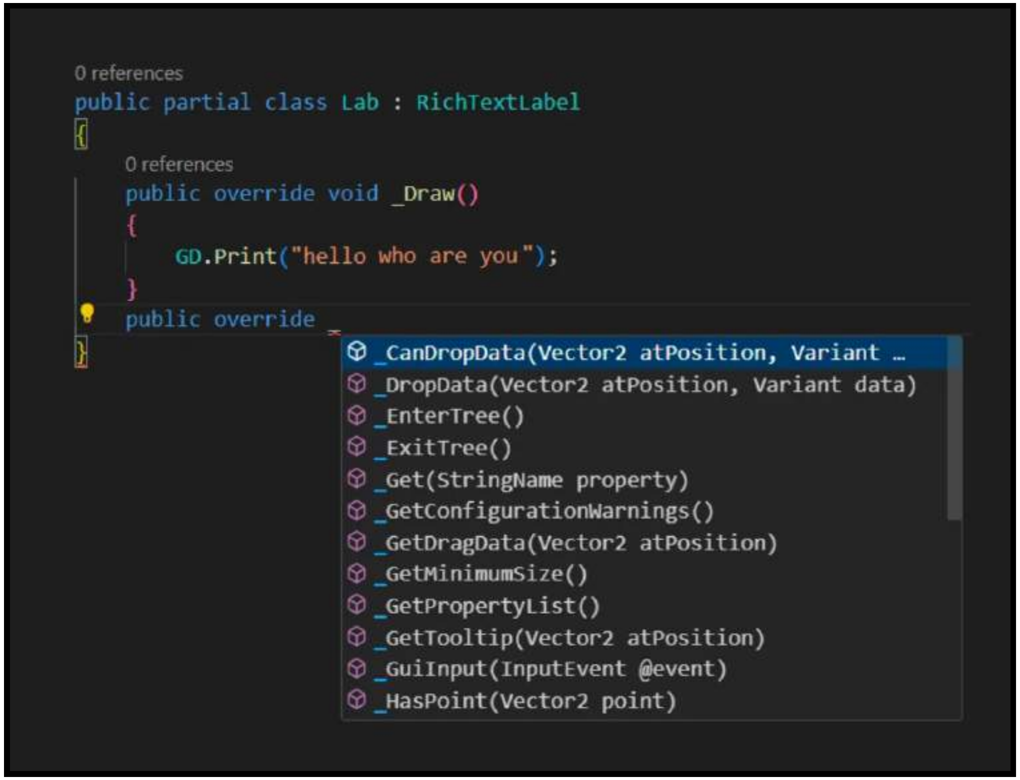
Haskell and pure functional languages

Self and pure object-oriented programming

Implementation properties

Should be easy to compile and close to the underlying system!

Fortran, C but also the Go language



```
0 references
public partial class Lab : RichTextLabel
{
    0 references
    public override void _Draw()
    {
        GD.Print("hello who are you");
    }
    public override _
}
```

_CanDropData(Vector2 atPosition, Variant …
_DropData(Vector2 atPosition, Variant data)
_EnterTree()
_ExitTree()
_Get(StringName property)
_GetConfigurationWarnings()
_GetDragData(Vector2 atPosition)
_GetMinimumSize()
_GetPropertyList()
_GetTooltip(Vector2 atPosition)
_GuiInput(InputEvent @event)
_HasPoint(Vector2 point)

User experience properties

Language should integrate well with the rest of the ecosystem

Scratch, HyperTalk, BASIC and aspects of Visual Basic

Case study: LINQ

LINQ queries in Visual Basic .NET and C#

```
Dim db As New northwindDataContext
Dim ukCompanies =
  From cust In db.Customers
  Where cust.Country = "UK"
  Select cust.CompanyName, cust.City
```

Why confuse programmers familiar with SQL?

```
SELECT [CompanyName], [City]
WHERE [Country] = 'UK'
FROM dbo.[Northwind]
```

TypeScript

Design & history

Charles
University

TypeScript

Language design motivations

- Full compatibility with JavaScript
- Support large-scale development
- ✓ Provide checking where possible
- Work with existing JavaScript code!

Charles University

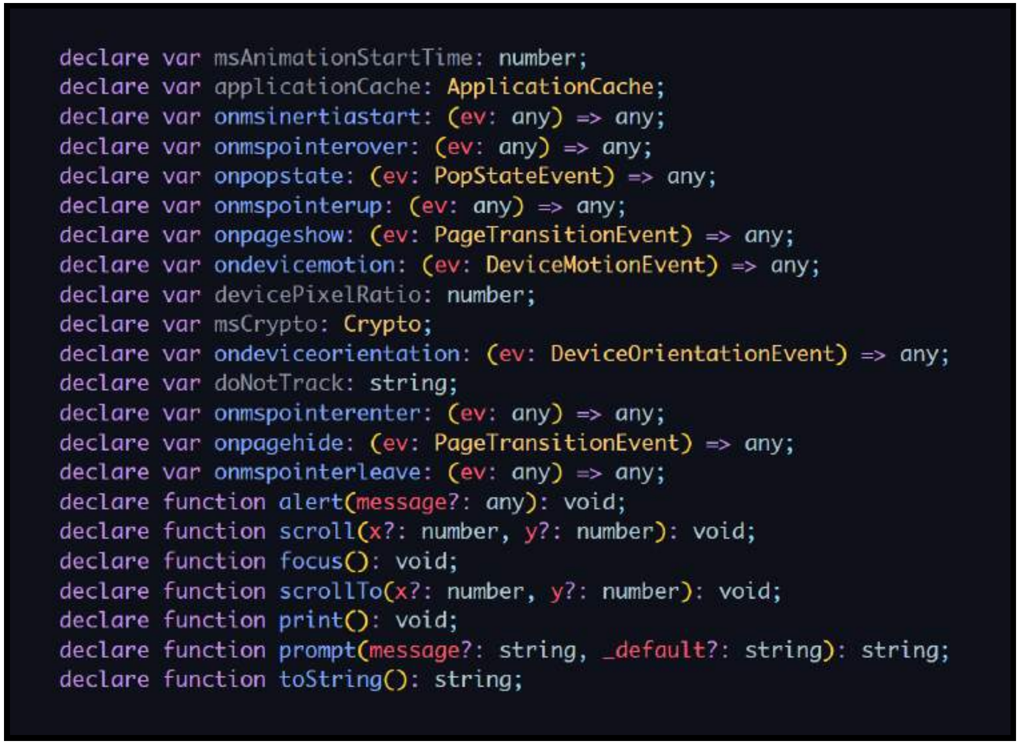
Retrofitting programming language to code?

Well, actually...

TypeScript by Microsoft
Hack by Facebook

Any new language version?

Retrofitting JavaScript with types



```
declare var msAnimationStartTime: number;
declare var applicationCache: ApplicationCache;
declare var onmsinertiastart: (ev: any) => any;
declare var onmspointerover: (ev: any) => any;
declare var onpopstate: (ev: PopStateEvent) => any;
declare var onmspointerup: (ev: any) => any;
declare var onpageshow: (ev: PageTransitionEvent) => any;
declare var ondevicemotion: (ev: DeviceMotionEvent) => any;
declare var devicePixelRatio: number;
declare var msCrypto: Crypto;
declare var ondeviceorientation: (ev: DeviceOrientationEvent) => any;
declare var doNotTrack: string;
declare var onmspointerenter: (ev: any) => any;
declare var onpagehide: (ev: PageTransitionEvent) => any;
declare var onmspointerleave: (ev: any) => any;
declare function alert(message?: any): void;
declare function scroll(x?: number, y?: number): void;
declare function focus(): void;
declare function scrollTo(x?: number, y?: number): void;
declare function print(): void;
declare function prompt(message?: string, _default?: string): string;
declare function toString(): string;
```

External definitions for existing libs (`.d.ts`)

Not everything can be typed (allow `any`)

JavaScript quirks (`null` vs. `undefined`)

Charles University

TypeScript types evolution

Version <2 (2012-16)

- Conservative extension of JavaScript
- Type system follows the 80%/20% rule

Versions 2, 3 (2016-20)

- More type-level gymnastics, closer to 90%

Version >=4 (2020-...)

- Even more fancy types, closer to 99%?
- Allows some scary type system hacks!

TypeScript

Practical types

Demo

Typing the createElement function

Charles University

Types concepts

```
// Union type and null type

getElementById(elementId:string):
  HTMLElement | null

// Overloaded function
// String literal types

createElement(tagName:"input"):
  HTMLInputElement

createElement(tagName:"button"):
  HTMLButtonElement
```

Overloaded functions

But with single implementation!

Flow-sensitive typing

To eliminate nulls

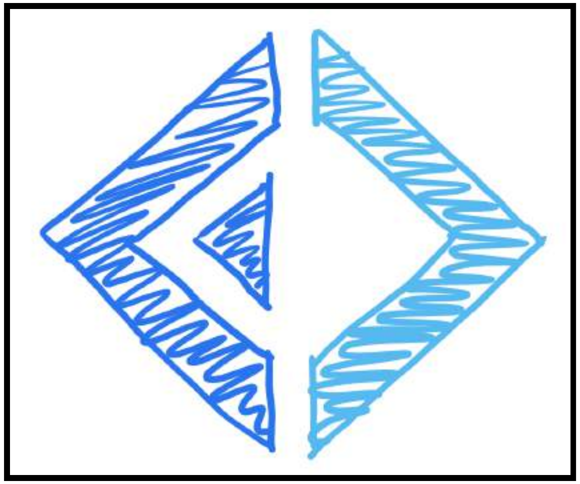
Union and literal types

More generally useful

"Clean" functional design

Options instead of nulls

```
type Option<'T> =
  | Some of 'T // Option has a value
  | None       // Option has no value
```



Matching instead of flow

```
match optElement with
| None -> failwith "out element missing"
| Some element ->
    el.appendChild(input)
    el.appendChild(btn)
```

Charles University

Analysis

Flow-sensitive typing

- Compatible with existing JavaScript code
- No need for extra syntax
- Somewhat unexpected behaviour!
- Type of variable changes with scope
- Valid patterns do not pass

Task #1

Break flow-sensitive typing!

Literal and union types

Union types

- Value of one of several types
- Safe languages prefer tagged unions!
- Flow-sensitive typing with `typeof` works

Literal types

- Type of a unique TypeScript value
- Works on values beyond strings, e.g. `[42]`
- Composes with other types, e.g. `[0|1, number]`

Demo

Unions, typeof, array literal types

Task #2

Implement bit increment

Function type overloading

```
// Multiple overload signatures
function speak(animal:"dog") : void
function speak(animal:"cat") : void

// One compatible implementation
function speak(animal:string) {
  switch(animal) {
    case "dog":
      console.log("woof")
      return
    case "cat":
      console.log("meow")
      return
  }
}
```

Implementation signature is hidden

Overload signature has to be compatible with the implementation

Overloads can differ in return type!

Task #3

Implementing createAnimal function

Special types

- **void** - No value, but usable as return type
- **never** - No value, represent unreachable code
- **unknown** - Cannot be used in any way
- **any** - Unknown, but can be used anywhere
- **undefined** - Special undefined value
- **null** - Special null value

Demo

What can and cannot be done with never

Union and intersection types

Union (`T1 | T2`)

- Represents set union of the values of the two types

Intersection (`T1 & T2`)

- Represents set intersection of the values of the types

TypeScript

Charles University

Demo

Record extension via intersection types

TypeScript

Algebraic data types

Algebraic types

```
// Algebraic data type that
// represents an expression
type Expr =
  // Numerical constant
  | Const of int

  // Named variable
  | Variable of string

  // Unary & binary operators
  | Unary of string * Expr
  | Binary of string * Expr * Expr

  // Function with arguments
  | Function of string * Expr list
```

Popular functional modelling tool

Entity consists of multiple sub-entities (operator + arguments)

Entity can be one of several options (number, variable, ...)

Demo

Algebraic data types in TypeScript

Algebraic data types

Literal & union types

- Tag to represent the kind of type
- `{ t:"s1" } | { t:"s2" }`
  `= { t:"s1" | "s2" }`

Flow-sensitive typing

- Check for a tag determines case
- Similar to pattern matching
- Works well with **`switch`**!

```
declare var msAnimationStartTime: number;
declare var applicationCache: ApplicationCache;
declare var onmsinertiastart: (ev: any) => any;
declare var onmspointerover: (ev: any) => any;
declare var onpopstate: (ev: PopStateEvent) => any;
declare var onmspointerup: (ev: any) => any;
declare var onpageshow: (ev: PageTransitionEvent) => any;
declare var ondevicemotion: (ev: DeviceMotionEvent) => any;
declare var devicePixelRatio: number;
declare var msCrypto: Crypto;
declare var ondeviceorientation: (ev: DeviceOrientationEvent) => any;
declare var doNotTrack: string;
declare var onmspointerenter: (ev: any) => any;
declare var onpagehide: (ev: PageTransitionEvent) => any;
declare var onmspointerleave: (ev: any) => any;
declare function alert(message?: any): void;
declare function scroll(x?: number, y?: number): void;
declare function focus(): void;
declare function scrollTo(x?: number, y?: number): void;
declare function print(): void;
declare function prompt(message?: string, _default?: string): string;
declare function toString(): string;
```

Task #4

Write a simple expression evaluator

Charles
University

Compare two representations

```
// Contact is an algebraic type
type Contact =
    ContactEmail
  | ContactPhone
  | ContactAll

interface ContactEmail {
  type: 'email'
  email: string
}
interface ContactPhone {
  type: 'phone'
  number: number[]
}
interface ContactAll {
  type: 'all'
  email: string
  number: number[]
}
```

Do they represent the same data?

```
interface Student1 {
  name: string
  email: string | null
  number: number[] | null
}

interface Student2 {
  name: string
  contact: Contact
}
```

Calculating with types

Type constructor algebra

- Record behaves as $A * B$ or $A \times B$
- Unions behave as $A + B$ or $A \cup B$
- Functions `A->B` behave as B^A
- Unit type is $\mathbf{1}$ and void (never) is $\mathbf{0}$

Usual algebraic laws work!

- $A * (B + C) = A * B + A * C$
- $A * 1 = A$ and $A * 0 = 0$

كتاب الخوارزمي

محمد بن موسى الخوارزمي رضي الله عنه [illegible]

Calculating with types

$$Contact = (Phone * Email) + Email + Phone$$

$$\begin{aligned} &Student2 \\ &\quad = Name * Contact \\ &\quad = Name * ((Phone * Email) + Email + Phone) \end{aligned}$$

$$\begin{aligned} &Student1 \\ &\quad = Name * (Phone + 1) * (Email + 1) \\ &\quad = Name * ((Phone + 1) * Email + (Phone + 1) * 1) \\ &\quad = Name * ((Phone * Email) + Email + Phone + 1) \end{aligned}$$

TypeScript

Type-level computation

Charles University

Motivation

```
// Returns book information consisting
// of title and year and description
// (only when detailed=true)
function getBook(index, detailed) {
  let b = lookup(index);
  if (detailed)
    return {
      title:b.title, year:b.year,
      description:b.description }
  else
    return {
      title:b.title, year:b.year }
}
```

How to check more dynamic behaviour?

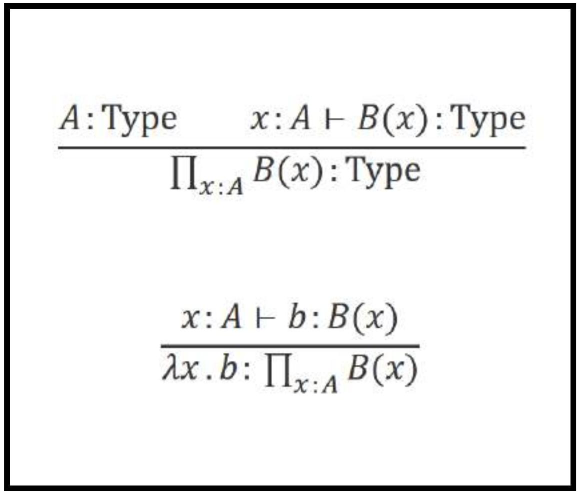
Dependent types!?

Return type depends on the value (type) of the argument

Dependent types

Serious type theory business!

- Types corresponding to terms in predicate logic
- Used in theorem provers



$$\frac{A : \mathrm{Type} \qquad x : A \vdash B(x) : \mathrm{Type}}{\prod_{x:A} B(x) : \mathrm{Type}}$$

$$\frac{x : A \vdash b : B(x)}{\lambda x \, . \, b : \prod_{x:A} B(x)}$$

More practical uses exist

- Track number of elements in a list `List n T`
- `List n T -> List m T -> List (n+m) T`

Conditional types

```
type LabelOrId<T> =
  // If T is a number,
  // result has id:number
  T extends number ?
    { id:number } :
    // If T is a string
    // result has label:string
    T extends string ?
      { label:string } :
      // Otherwise return
      // unusable type
      never;
```

```
cond ? type1 : type2
```

Result type depends on another type

Condition can only check using **`extends`** (subtype)

Demo

Book info with optional details

Transforming record types

Keyof types

- Union type of type's keys as literals
- `{ age:number; name:string }`
- `keyof(P) = "age"|"name"`

Indexed access types

- Type of a property of a type
- `P["age"] = number`
- `P[keyof P] = "number"|"string"`

Demo

Filtering book records by key

Task

Implement well-typed join operation

Charles University

Demo

Transforming record types using keyof

Galaxy brain?

Pattern matching at the type level using infer/extends

Type level array and string interpolation

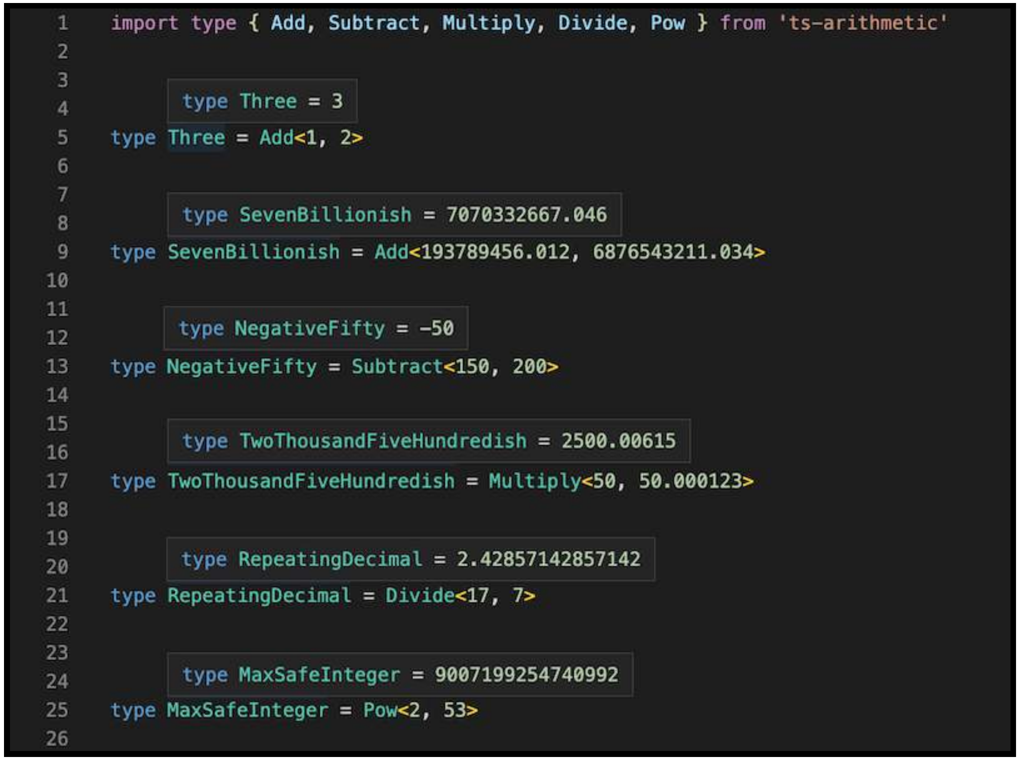
Representing numbers at the type level

Demo

Flattening array types

Demo

Strings & type-level numbers



```
import type { Add, Subtract, Multiply, Divide, Pow } from 'ts-arithmetic'


      type Three = 3
type Three = Add<1, 2>


      type SevenBillionish = 7070332667.046
type SevenBillionish = Add<193789456.012, 6876543211.034>


      type NegativeFifty = -50
type NegativeFifty = Subtract<150, 200>


      type TwoThousandFiveHundredish = 2500.00615
type TwoThousandFiveHundredish = Multiply<50, 50.000123>


      type RepeatingDecimal = 2.42857142857142
type RepeatingDecimal = Divide<17, 7>


      type MaxSafeInteger = 9007199254740992
type MaxSafeInteger = Pow<2, 53>

```

Arithmetic

You can do (big) arithmetic in TypeScript types

Why?

Good question... but maybe matrix operations?

Charles University

Task

Figure out how ts-arithmetic works!

TypeScript

Conclusions

Excuse to talk about

- Programming language design
  Design based on existing codebase is funny!
- Algebraic data types
  Nice and powerful modelling tool
- ≠ Dependent type system concepts
  Weird in TypeScript, but interesting!
- Some practical TypeScript tricks
  Lets you do lots of useful things...

Charles University

PL project ideas?

Safe data manipulation or charting libraries in TypeScript?

Types for untyped languages with existing codebase (R, Python)?

Conclusions

Advanced types in TypeScript

- Hidden surprises in a widely used language
- Happens to be unsafe and undecidable!
- Going beyond 80:20 requires lots of magic
- Talk to me about projects & PL reading group!

Tomáš Petříček, 309 (3rd floor)

✉ petricek@d3s.mff.cuni.cz

➜ https://tomasp.net | @tomaspetricek

➜ https://d3s.mff.cuni.cz/teaching/nprg077

Charles University