Extreme TypeScript

Mastering Recursion and Inference via Type-Level Arithmetic

CKEditor

CKEditor

👋

Ondřej Chrastina

Developer Advocate

CKEditor

ondrej.chrastina.dev

How it all started ?

Jirka 💜 Haskell

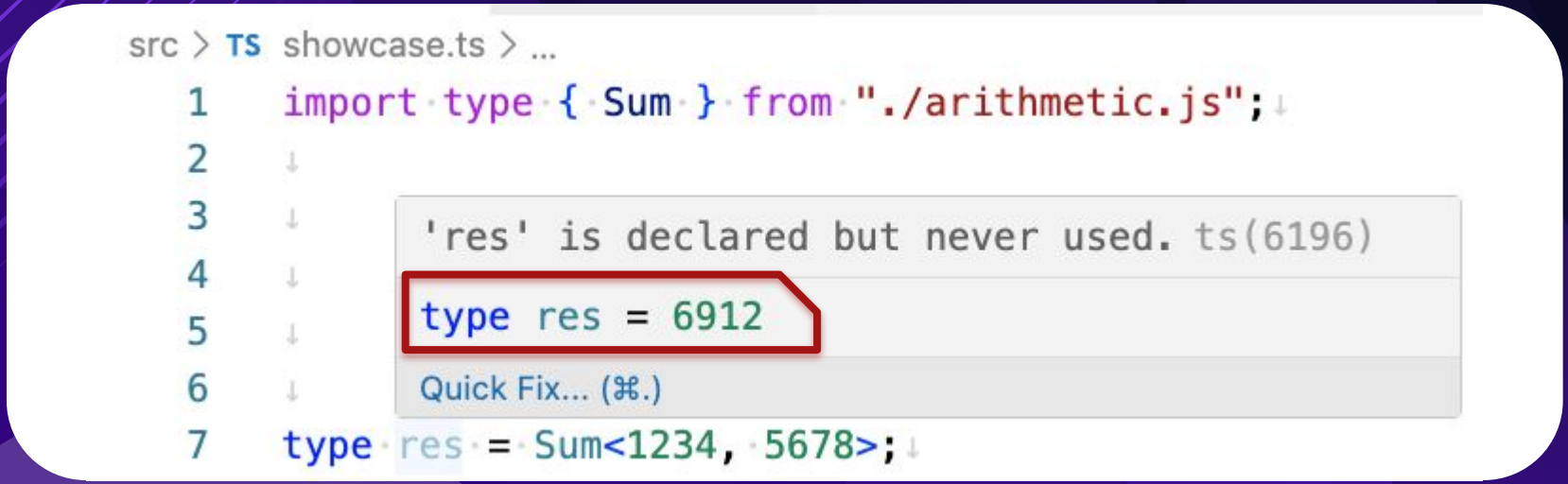
https://github.com/JiriLojda

Review: “ What is that? ”

yargs.ts

```
export type RegisterCommand = <Result, InitialParams extends LogOptions =
LogOptions>(
  obj: Readonly<{
    command: <CommandParams extends InitialParams = InitialParams>(
      cmdModule: StandaloneCommandModule<InitialParams, CommandParams>,
    ) => Result;
  }>,
) => Result;
```

Goal: READ complex types



Building blocks
Sum 2 DIGITS - the trick!

Literal definition

```
type Digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
```

Map

```
type DigitToTupleMap = {
  0: [];
  1: [0];
  2: [0, 0];
  3: [0, 0, 0];
  // ... up to 9
  9: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
};
```

```
type res = DigitToTuple[2] // [0,0]
```

Generics & Spread operator

```
type ConcatTuples<A extends Digit, B extends Digit> =
  [...DigitToTuple[A], ...DigitToTuple[B]];

type res = ConcatTuples<2, 3>; // [0,0,0,0,0]
```

Length

```
// 5
type res = [0, 0, 0, 0, 0]['length']
```

Building blocks
Sum 2 DIGITS - the trick!

Literal definition

```
type Digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
```

Map

```
type DigitToTupleMap = {
  0: [];
  1: [0];
  2: [0, 0];
  3: [0, 0, 0];
  // ... up to 9
  9: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
};
```

```
type SimpleSum<A extends Digit, B extends Digit> =
  [...DigitToTuple[A], ...DigitToTuple[B]]['length'];
```

Gene

```
type ConcatTuples<A extends Digit, B extends Digit> =
  [...DigitToTuple[A], ...DigitToTuple[B]];

type res = ConcatTuples<2, 3>; // [0,0,0,0,0]
```

gth

```
// 5
type res = [0, 0, 0, 0, 0]['length']
```

First TS calculator!

```
type A = DigitToTuple[3]; // [0,0,0]
type B = DigitToTuple[5]; // [0,0,0,0,0]
type LengthA = A['length'] // 3
type ABConcat = [...A, ...B] // [0,0,0,0,0,0,0,0]
type SumAB = ABConcat['length'] // 8
```

Building blocks
Sum 2 numbers WITHOUT CARRY

Number to String (via template literal)

```ts
type NumberToString<N extends number> = `${N}`;

type A = NumberToString<3>;   // "3"
type B = NumberToString<123>; // "123"
type C = `${42}`;             // "42"
```

String to Tuple (via recursion & template literal)

```ts
type StrToTuple<
  T extends string,
  Accum extends readonly string[] = []
> = T extends `${infer Fst}${infer Rest}`
  ? StrToTuple<Rest, [...Accum, Fst]>
  : Accum;

// ["1", "2", "3"]
type res = StrToTuple<"123">
```

```ts
type StrToTupleMap = {
  '0': [];
  '1': [0];
  '2': [0, 0];
  '3': [0, 0, 0];
  // ... up to '9'
  '9': [0, 0, 0, 0, 0, 0, 0, 0, 0];
};
```

*key is string

String to Number (infer & condition)

```ts
type StringToNumber<T extends string> = T extends
`${infer Res extends number}`
  ? Res
  : never;
```

```ts
// 123
StringToNumber<'123'>;
```

Concat Strings (via recursion & inference)

```ts
type ConcatStrings<
  T extends readonly string[],
  Accum extends string = ""
> = T extends [
  infer Fst extends string,
  ...infer TRest extends readonly string[]
]
  ? ConcatStrings<TRest, `${Accum}${Fst}`>
  : Accum;
```

```ts
// '123'
ConcatStrings<['1', '2', '3']>;
```

No Carry Calculator

```
type A = 123
type B = 456
type StrA = `${A}` // "123"
type StrB = `${B}` // "456"
type TupleA = StrToTuple<StrA> // ["1","2","3"]
type TupleB = StrToTuple<StrB> // ["4","5","6"]
type StrSum = SumTupleNoCarry<TupleA, TupleB> // "579"
type NumSum = StringToNumber<StrSum> // 579 🎉
```

*

```ts
// Main recursive type: process right-to-left, accumulate result
// Structure matches arithmetic.ts but WITHOUT carry parameter
type SumTupleNoCarry<
  Num1 extends readonly string[], // ["1", "2", "3"], | ["1", "2"] | ["1"] | []
  Num2 extends readonly string[], // [ "4", "5", "6"], | ["4", "5"] | ["4"] | []
  Accum extends string = '' //       ""                | "9"        | "79"  | "579"
> =
  // Base case: one side empty, prepend remaining digits
  Num1 extends []
    ? `${ConcatStrings<Num2>}${Accum}`
    : Num2 extends []
      ? `${ConcatStrings<Num1>}${Accum}`
      // Extract last digit from each (right-to-left processing)
      : Num1 extends [...infer Rest1 extends string[], infer Last1 extends string]
        ? Num2 extends [...infer Rest2 extends string[], infer Last2 extends string]
          // Sum last digits, prepend to accumulator, recurse on rest
          ? SumTwoStrDigits<Last1, Last2> extends infer TSum extends number
            ? SumTupleNoCarry<Rest1, Rest2, `${TSum}${Accum}`>
            : never
          : never
        : never;
```

CKEditor

Full Calculator
& much more!

https://github.com/Simply007/ts-type-arithmetic

What's in it for me ?

```ts
import { messages, createIntl } from "@ccssmnn/intl"

// 1. Define your messages
const copy = messages({
    greeting: "Hello {$name}!",
    count: "You have {$num :number} items",
})

// 2. Create translator
const t = createIntl(copy, "en")

// 3. Use it
t("greeting", { name: "World" }) // "Hello World!"
t("count", { num: 42 }) // "You have 42 items"
t("count", { num: "No" }) // TYPE Error
```

```
Type 'string' is not assignable to type 'number'. ts(2322)

types.d.ts(45, 70): The expected type comes from property 'num' which is declared here on type '{ num: number; }'

(property) num: number
```

https://github.com/ccssmnn/intl

What's in it for me ?

```ts
import type { EditorConfig, GetSubConfig } from 'ckeditor5';

// ✅
type Toolbar = GetSubConfig<EditorConfig, 'toolbar'>;
// ❌
const brokenItems: GetSubConfig<EditorConfig, 'toolbar.items'> = 'bold, italic';
```

```
'brokenItems' is declared but its value is never read. ts(6133)
Type 'string' is not assignable to type 'ToolbarConfigItem[]'. ts(2322)
+ const brokenItems: ToolbarConfigItem[] | undefined
```

https://github.com/ckeditor/ckeditor5