Т БАНК

ТИПИЗИРУЕМ `eval` И `new Function`: ИНТЕРПРЕТАТОР JAVASCRIPT НА TYPESCRIPT-ТИПАХ

Сергей Соловьев

HolyJS 2025 Spring

👋 ОБО МНЕ

СЕРГЕЙ СОЛОВЬЕВ

- 7+ лет промышленной разработки на TS
- Создаю B2B интерфейсы в Т-Банке
- Пишу кандидатскую по философии науки

ДИСКЛЕЙМЕР

Во время сессии возможны внезапные просветления и лёгкие галлюцинации, а также неконтролируемое желание типизировать все подряд. Вы пришли на доклад mad skills, и организаторы не несут ответственности за потерянный сон и другие последствия.

КАКОЙ У НАС ПЛАН?

TypeScript Download Docs Handbook Community Tools Search Docs

Playground TS Config Examples Help Settings

v5.8.3 Run Export Share

```
const square = new Function('a', 'b', 'c', `
    const s1 = a * b;
    const s2 = b * c;
    const s3 = a * c;

    return (s1 + s2 + s3) * 2;
`);

// const s: 178
const s = square(11, 4, 3);

console.log(s);
```

.JS .D.TS Errors Logs Plugins

```
"use strict";
Object.defineProperty(exports, "__esMo
exports.s = void 0;
const square = new Function('a', 'b',
    const s1 = a * b;
    const s2 = b * c;
    const s3 = a * c;

    return (s1 + s2 + s3) * 2;
`);
const s = square(11, 4, 3);
exports.s = s;
console.log(s);
```

КАКОЙ ВСЕ-ТАКИ ПЛАН?

- Полнота по Тьюрингу
  Зачем это все и почему это возможно
- Типзируем `eval`
  Арифметика на типах
- Типзируем `new Function`
  Интерпретатор JavaScript на типах

ВВЕДЕНИЕ

МОТИВАЦИЯ

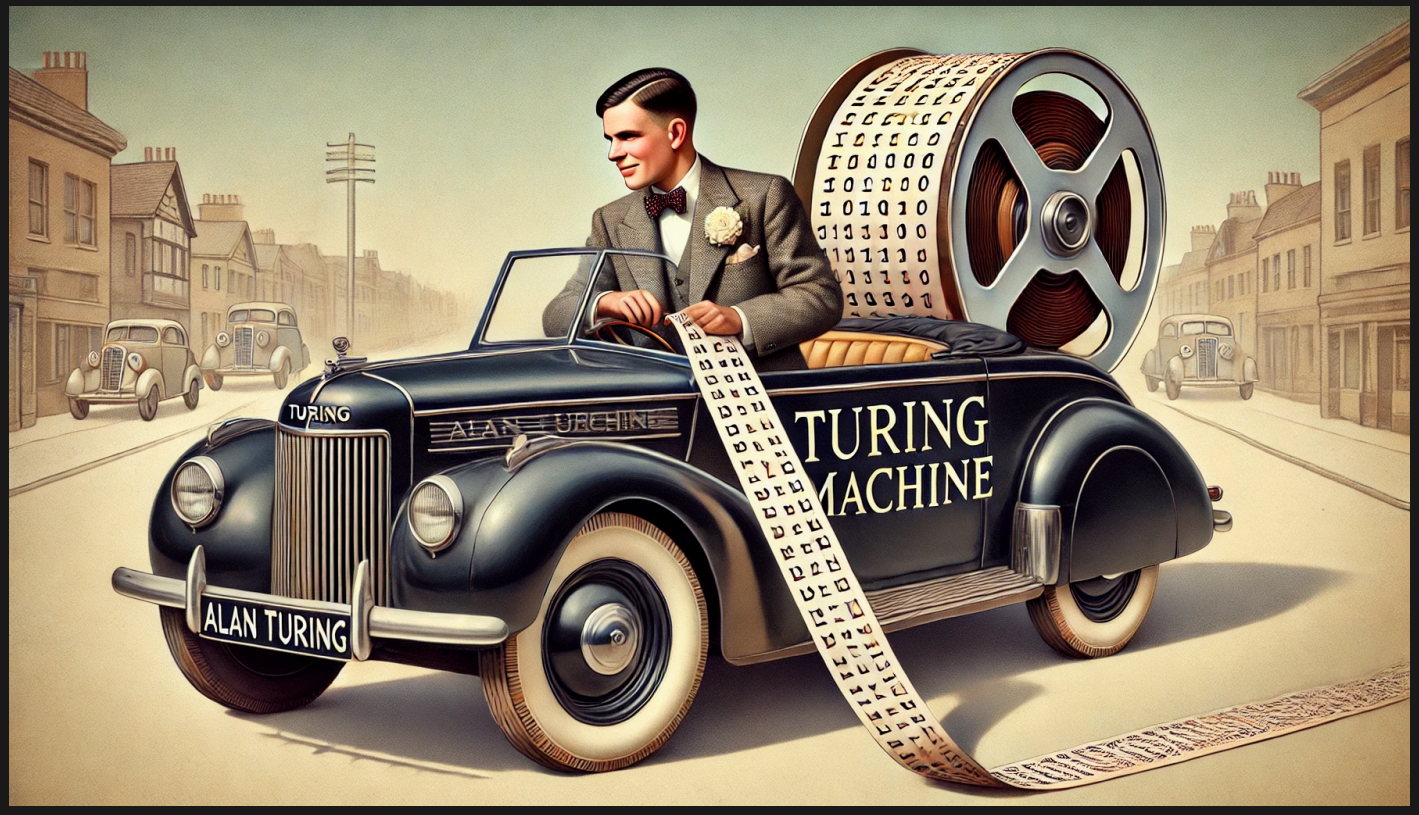
Type-level программирование:

- помогает почувствовать возможности и ограничения TS
- помогает писать “умные” и компактные типы
- развивает алгоритмическое мышление

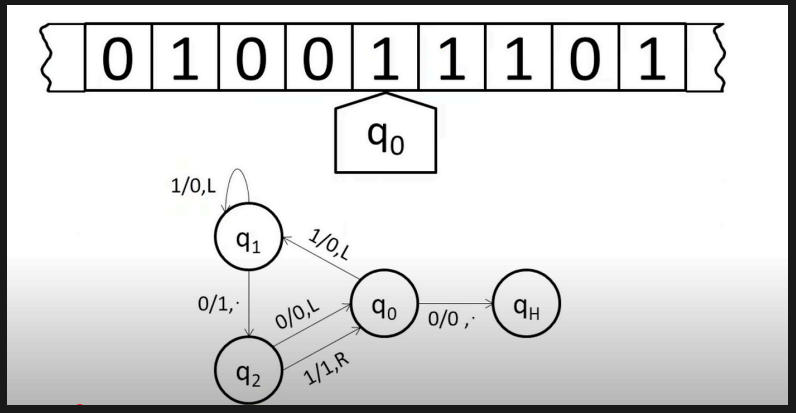
1. TypeScript Design Goals
2. Typescript: unsound behavior или поблажки надежности

ПОЧЕМУ ЭТО ВОЗМОЖНО?

ВСЕ ДЕЛО В МАШИНЕ ТЬЮРИНГА



МАШИНА ТЬЮРИНГА



Математическая модель вычислений, предложенная Аланом Тьюрингом в 1936 году для формализации понятия алгоритма.

УНИВЕРСАЛЬНАЯ МАШИНА ТЬЮРИНГА

Машина Тьюринга, которая способна эмулировать работу любой машины Тьюринга (включая саму себя)

ПОЛНОТА ПО ТЬЮРИНГУ

Полнота по Тьюрингу – это характеристика ЯП, которая означает возможность реализации любого алгоритма

НЕОЖИДАННАЯ ПОЛНОТА ПО ТЬЮРИНГУ ПОВСЮДУ

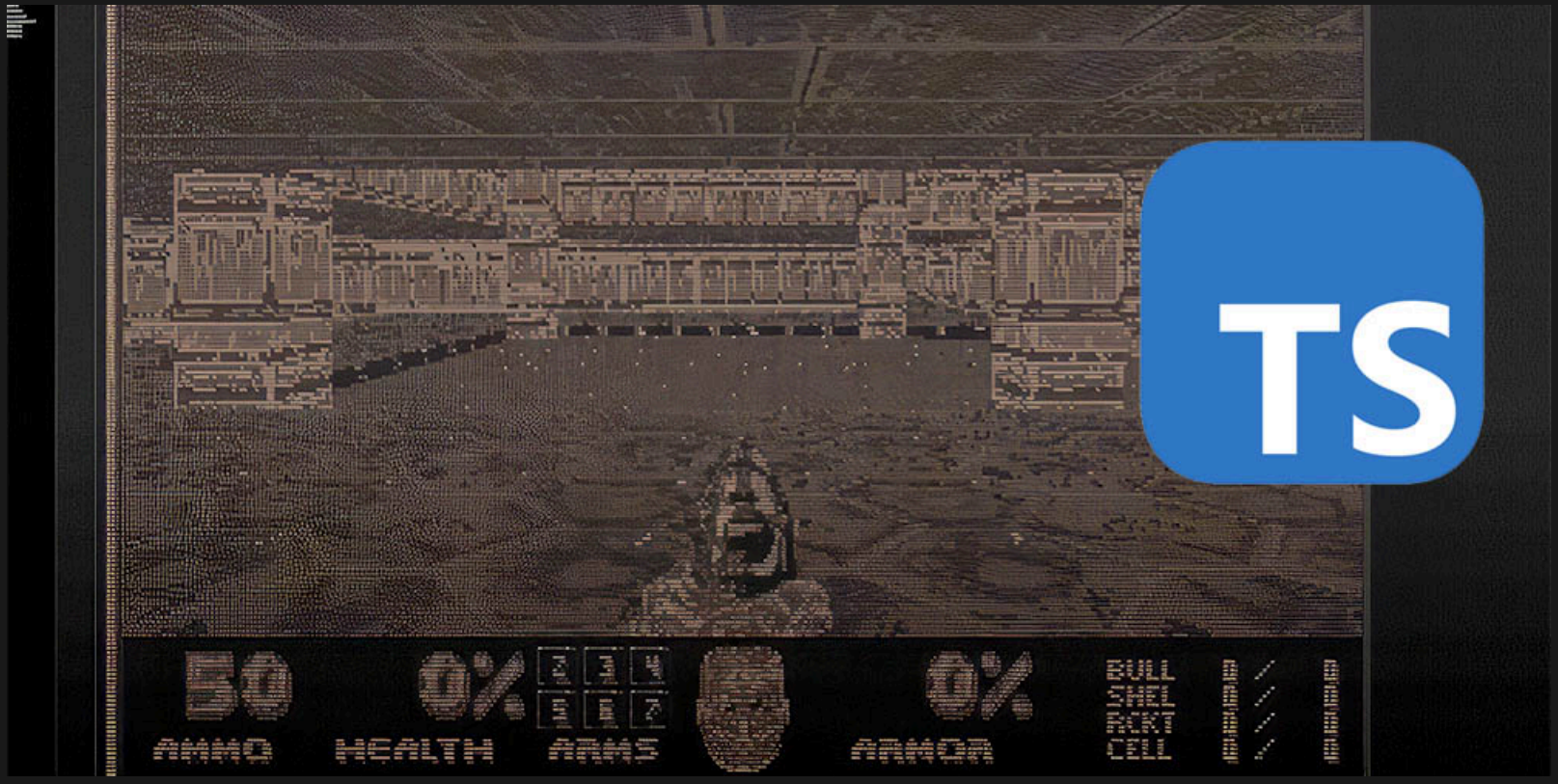
- Шаблоны в C++
- Арифметика Пеано
- ДНК-вычисления
- Minecraft
- **Система типов** TypeScript

1. Неожиданная полнота по Тьюрингу повсюду
2. TypeScripts Type System is Turing Complete #14833

ТИПЫ КАК ЯЗЫК ПРОГРАММИРОВАНИЯ

Уровень значений	Уровень типов
примитивы	литеральные типы
массив	кортеж
функция	обобщённый тип (generic)
ветвление `if`	условные типы `extends`
циклы	рекурсия/mapped types
переменная `const`	условный тип с `infer`

DOOM НА ТИПАХ TYPESCRIPT



by Dimitri Mitropoulos

ВЫВОД

Система типов TS полна по Тьюрингу, значит в ней можно реализовать любой алгоритм, включая интерпретатор другого языка (без учета ограничений памяти)

ТИПИЗИРУЕМ `eval`

```
1 // const r: 170
2 const r = eval('(15 + 2) * 10');
```

ЗАМЕЧАНИЯ

- Используется TypeScript 5.8.2 в strict mode
- Работаем только с числовыми данными
- Фокус на happy path без обработки ошибок
- Параметры обобщенных типов именуются с префиксом T или одной буквой

СЛОЖЕНИЕ ЧИСЕЛ

```ts
// type S = 17
type S = Sum<15, 2>;
```

TypeScript умеет конкатенировать кортежи

```ts
type Sum<A extends number, B extends number> =
  [...TupleOfLength<A>, ...TupleOfLength<B>]['length'];
```

TupleOfLength

```
1 const tupleOfLength = n => new Array(n).fill(undefined);
```

```
1 const tupleOfLength = (n: number, acc: unknown[] = []) =>
2   acc['length'] === n
3     ? acc
4     : tupleOfLength(n, [...acc, undefined]);
```

```
1 type TupleOfLength<
2   N extends number,
3   Acc extends unknown[] = []
4 > = Acc['length'] extends N
5   ? Acc
6   : TupleOfLength<N, [...Acc, unknown]>;
```

ОГРАНИЧЕНИЕ НА ГЛУБИНУ РЕКУРСИИ

```
1 if (instantiationDepth === 100 || instantiationCount >= 5000000) {
```

В TS ограничение по умолчанию 100, а для хвостовой рекурсии – 1000.

1. Tail recursive evaluation of conditional types #45711
2. main/src/compiler/checker.ts

ОБХОД ОГРАНИЧЕНИЙ

- Рекурсия по чанкам
- Эксплойт за счет `Identity<T> = T`

```ts
type TupleOfLength<
  N extends number,
  Acc extends unknown[] = []
> = Acc['length'] extends N
    ? Acc
    : Identity<
      TupleOfLength<N, [...Acc, Acc['length']]>
    >;

type LongTuple = TupleOfLength<1500>;
```

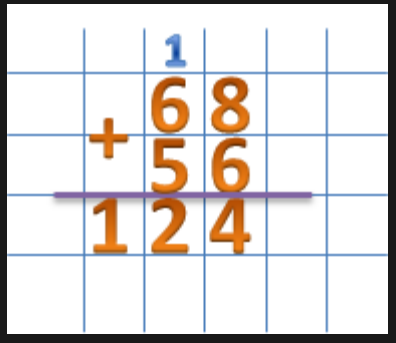
1. Tail recursion optimization limit 999 may be manipulated #49459

ПРОБЛЕМЫ ОСТАЮТСЯ

- предел длины кортежа – 10000 элементов
- низкая производительность

СЛОЖЕНИЕ В СТОЛБИК

На помощь приходит решение, призванное некогда обойти ограничения RAM кожаных мешков!



ЛИТЕРАЛЬНЫЕ, ШАБЛОННЫЕ И УСЛОВНЫЕ ТИПЫ

```ts
type Hello = 'Hello';

// type HelloWorld = 'Hello, world!'
type HelloWorld = `${Hello}, world!`

// type World = 'world'
type World =
  HelloWorld extends `Hello, ${infer S}!` ? S : never;
```

1. Template Literal Types
2. Conditional Types

ПРИВЕДЕНИЕ ТИПОВ

ПРИВЕДЕНИЕ ТИПОВ

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

type AsNumber<S extends string> =
    S extends `${infer N extends number}` ? N : 0;
```

Примеры

```ts
// type S = "123456789"
type S = AsString<123456789>;

// type N = 123456789
type N = AsNumber<S>;
```

ПОСИМВОЛЬНАЯ ОБРАБОТКА СТРОК

```ts
type Parsed<S extends string> =
  S extends `${infer H}${infer T}` ? [H, T] : never;
```

```ts
// type P1 = ['H', 'ello!']
type P1 = Parsed<'Hello!'>;

// type P2 = ['!', '']
type P2 = Parsed<'!'>;

// type P3 = never
type P3 = Parsed<''>;
```

REVERSE

```
type Reverse<S extends string> =
  S extends `${infer Head}${infer Tail}`
    ? `${Reverse<Tail>}${Head}`
    : '';
```

```
type Reverse<S extends string, Acc extends string = ''> =
  S extends `${infer Head}${infer Tail}`
    ? Reverse<Tail, `${Acc}${Head}`>
    : Acc;
```

ГОТОВИМСЯ СКЛАДЫВАТЬ

```
type Sum<A extends number, B extends number> =
  AsNumber<Reverse<SumReversedStrings<
    Reverse<AsString<A>>,
    Reverse<AsString<B>>
  >>>;
```

```
type SumReversedStrings<
  A extends string,
  B extends string
> = ???
```

РЕАЛИЗАЦИЯ СЛОЖЕНИЯ

```ts
type SumReversedStrings<
  A extends string,
  B extends string,
  TCarry extends string = '',
  TAcc extends string = ''
> =
  A | B extends '' ? `${TAcc}${TCarry}`
  : A extends '' ? SumReversedStrings<'0', B, TCarry, TAcc>
  : B extends '' ? SumReversedStrings<A, '0', TCarry, TAcc>
  : [A, B] extends [`${infer AHead}${infer ATail}`, `${infer BHead}${infer BTail}`]
  ? SumReversedDigits<AHead, BHead, TCarry> extends `${infer R}${infer TCarryNext}`
    ? SumReversedStrings<ATail, BTail, TCarryNext, `${TAcc}${R}`>
    : never
  : never;
```

```ts
type Tuple<L extends string> = TupleOfLength<AsNumber<L>>;

type SumReversedDigits<A extends string, B extends string, C extends string> =
  Reverse<AsString<[...Tuple<A>, ...Tuple<B>, ...Tuple<C>]['length'] & number>>
```

1. Еще 51 вариант реализации суммы

ВЫВОД

Благодаря посимвольной обработке,
мы реализовали сложение больших чисел

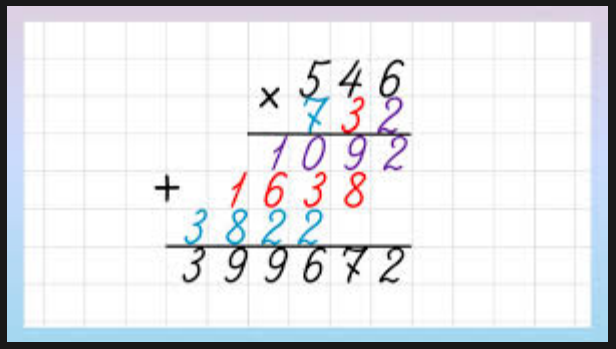
```
1 // type S = 22145
2 type S = Sum<2578, 19567>;
```

УМНОЖЕНИЕ

Чтобы умножить число a на число b, необходимо сложить b чисел a.

1. Умножение

УМНОЖЕНИЕ В СТОЛБИК



В худшем случае умножение двух чисел длиной L1 и L2 требует (L1 ∗ L2 ∗ 8) операций сложения.

АЛГОРИТМ НАКОПЛЕНИЯ

Действие	a	b	r	a * b + r
	15	212	0	3180
r += a; b--	15	211	15	3180
r += a; b--	15	210	30	3180
a *= 10; b /= 10	150	21	30	3180
r += a; b--	150	20	180	3180
a *= 10; b /= 10	1500	2	180	3180
r += a; b--	1500	1	1680	3180
r += a; b--	1500	0	3180	3180

ГОТОВИМСЯ УМНОЖАТЬ

```
type Product<A extends number, B extends number> =
    AsNumber<Reverse<ProductReversedStrings<
      Reverse<AsString<A>>,
      Reverse<AsString<B>>
    >>>
```

Как реализовать `ProductReversedStrings`?

РЕАЛИЗАЦИЯ УМНОЖЕНИЯ

```ts
type ProductReversedStrings<A extends string, B extends string, R extends string = "0"> =
  A extends "0" ? R :
  B extends "0" ? R
  : A extends `${infer AH}${infer AT}`
  ? AH extends "0"
    ? ProductReversedStrings<AT, `0${B}`, R>
    : ProductReversedStrings<DecrementReversed<A>, B, SumReversedStrings<R, B>>
  : R;
```

```ts
type DecrementMap = { "1": "0"; /* ... */ "9": "8"; };

type DecrementReversed<A> = A extends `${infer AH}${infer AT}`
  ? AH extends "0"
    ? `9${DecrementReversed<AT>}`
    : `${DecrementMap[AH & keyof DecrementMap]}${AT}`
  : never;
```

1. 517 - Multiply by Sobes76rus

ВЫВОД

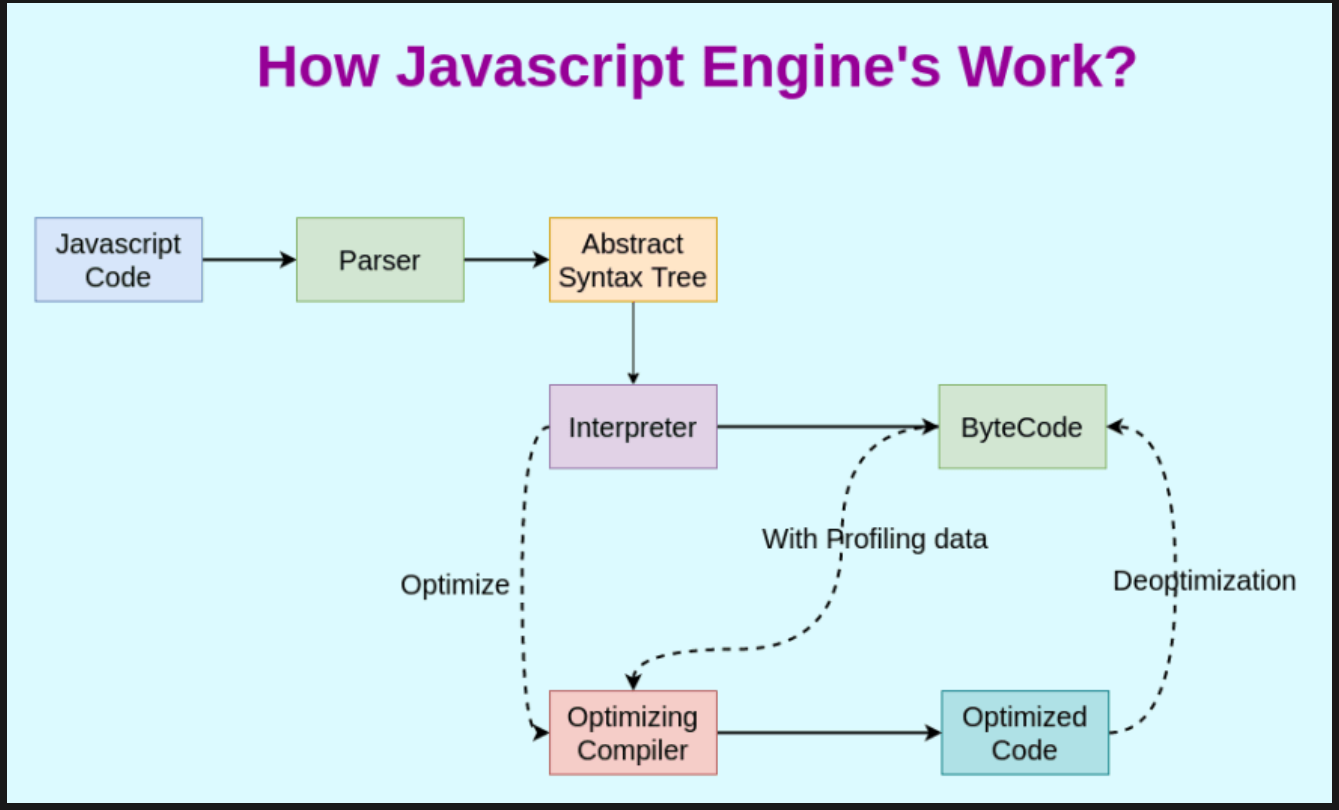
Используя декремент, сумму и умножение/деление на 10 мы реализовали умножение произвольных чисел

```
1 // type S = 50443726
2 type S = Product<2578, 19567>;
```

ВЫЧИСЛЯЕМ ВЫРАЖЕНИЕ

```
1 // type R = 170
2 type R = Evaluated<'(15 + 2) * 10'>;
```

КАК ИНТЕРПРЕТАТОР JAVASCRIPT ВЫПОЛНЯЕТ КОД



1. Путь к пониманию байт-кода V8

ОБРАБОТКА

- исходный код
- список токенов
- дерево разбора
- AST
- байт-код
- машинный код

УПРОЩЕННЫЙ ПОДХОД

Отложим все это в сторону

РАЗБИРАЕМ СТРОКУ

```
type Evaluated<
  TExpression extends string,
> = TExpression extends `${infer L}(${infer TNested})${infer R}`
    ? Evaluated<`${L}${Evaluated<TNested>}${R}`>
    : TExpression extends `${infer A} + ${infer B}`
    ? Sum<Evaluated<A>, Evaluated<B>>
    : TExpression extends `${infer A} * ${infer B}`
    ? Product<Evaluated<A>, Evaluated<B>>
    ? AsNumber<TExpression>
```

```
// type R = 170
type R = Evaluated<'(15 + 2) * 10'>;
```

1. Калькулятор на типах TypeScript

ПЕРЕГРУЗКА eval

```
1 declare global {
2   function eval<E extends string>(expr: E): Evaluated<E>;
3 }
```

```
1 // const r: 170
2 const r = eval('(15 + 2) * 10');
```

ПРОВЕРЯЕМ РЕЗУЛЬТАТ

TypeScript Download Docs Handbook Community Tools Search Docs

Playground TS Config Examples Help Settings

v5.8.3 Run Export Share

```
1   // const result: 100500
2   const result = eval("(888 + 111 + 1) * (5 * 20) + 500");
3
4   console.log(result); // 100500
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

.JS .D.TS Errors Logs Plugins

```
"use strict";
Object.defineProperty(exports, "__esMo
exports.result = void 0;
const result = eval("(888 + 111 + 1) *
exports.result = result;
console.log(result);
```

ТИПИЗИРУЕМ
new Function

ЧЕГО НЕ ХВАТАЕТ?

- переменных
- `return`
- параметров функции

ПОДДЕРЖКА ПЕРЕМЕННЫХ И return

```
1 const count = new Function(`
2     const a = 2;
3     const b = 2 * 10;
4 
5     return a + b;
6 `);
7 
8 // const r: 22
9 const r = count();
```

ПЕРЕМЕННЫЕ

Поддержка переменных означает наличие контекста выполнения

```
1 type Context = Record<string, number>;
```

ПОЧТИ REDUX

```
1 action :     'const a = 2;'
2 state  : {} ============> {a: 2}
```

```
1 action :         'const b = a + 2;'
2 state  : {a: 2} ==================> {a: 2, b: 4}
```

Нам остается написать редьюсер

ИНТЕРПРЕТАЦИЯ ИНСТРУКЦИИ

```ts
type Interpreted<
  TInstruction extends string,
  TContext extends Context
> = TInstruction extends `const ${infer TVariable} = ${infer TExpression}`
  ? WithVar<TContext, TVariable, Evaluated<TExpression, TContext>>
  : TInstruction extends `return ${infer TExpression}`
  ? WitReturn<TContext, Evaluated<TExpression, TContext>>
  : TContext;
```

```ts
// type C = { a: 6; }
type C = Interpreted<'const a = 2 + 4', {}>;

// type C2 = { a: 6; '@return': 12; }
type C2 = Interpreted<'return a * 2', C>;
```

ПРИМЕНЕНИЕ ПЕРЕМЕННЫХ

```ts
type Evaluated<
  TExpression extends string,
  TContext extends Context = {},
> = TExpression extends `${infer L}(${infer TNested})${infer R}`
    ? Evaluated<`${L}${Evaluated<TNested, TContext>}${R}`, TContext>
    : TExpression extends `${infer A} + ${infer B}`
    ? Sum<Evaluated<A, TContext>, Evaluated<B, TContext>>
    : TExpression extends `${infer A} * ${infer B}`
    ? Product<Evaluated<A, TContext>, Evaluated<B, TContext>>
    : VarValue<TContext, TExpression> extends never
    ? AsNumber<TExpression>
    : VarValue<TContext, TExpression>;
```

```ts
// type A = 2
type A = Evaluated<'a', { a: 2 }>
```

АЛГОРИТМ ВЫПОЛНЕНИЯ ФУНКЦИИ

- разбиваем листинг по `;` на инструкции
- триммируем каждую инструкцию
- готовим начальное состояние
- последовательно проходим инструкции, применяя соответствующий “редьюсер”

ПАРСИМ ИНСТРУКЦИИ

```
type ParsedInstructions<TBody extends string> =
  TrimLines<Split<Trim<TBody, ";">, ";">>;
```

```
type Trim<
  TString extends string,
  TOccurence extends string = " " | "\n"
> = TString extends (
  | `${TOccurence}${infer TTrimmed}`
  | `${infer TTrimmed}${TOccurence}`
) ? Trim<TTrimmed> : TString;
```

ПОТОК ВЫПОЛНЕНИЯ

```
type InterpretedInstructions<
  TInstructions extends string[],
  TContext extends Context
> = TInstructions extends [
  infer TInstruction extends string,
  ...infer TRest extends string[]
]
  ? InterpretedInstructions<
    TRest,
    Interpreted<TInstruction, TContext>
  >
  : TContext;
```

ПЕРЕГРУЗКА new Function

```
declare global {
  interface FunctionConstructor {
    new<TBody extends string>(
      body: TBody
    ): () => FunctionResult<TBody>
  }
}

type FunctionResult<TBody extends string> =
  ReturnedValue<InterpretedInstructions<
    ParsedInstructions<TBody>,
  >>;
```

ВЫВОД

Благодаря последовательному применению инструкций к контексту, мы вычисляем функции без параметров на уровне типов

```
const count = new Function(`
    const a = 2;
    const b = 2 * 10;

    return a + b;
`);

// const r: 22
const r = count();
```

ПАРАМЕТРЫ ФУНКЦИИ

На основе имен формальных параметров и значений фактических формировать начальное состояние контекста выполнения.

FunctionResult С ПАРАМЕТРАМИ

```ts
type FunctionResult<
  TBody extends string,
  TArgNames extends string[],
  TArgValues extends number[]
> = ReturnedValue<InterpretedInstructions<
  ParsedInstructions<TBody>,
  FunctionInitContext<TArgNames, TArgValues>
>>;
```

```ts
// type R = 150
type R = FunctionResult<'return a * b', ['a', 'b'], [10, 15]>
```

FunctionInitContext

```
type FunctionInitContext<
  TArgNames extends string[],
  TArgValues extends number[],
> = FromEntries<
  Zip<TArgNames, TArgValues> & [string, number][]
>;
```

ПЕРЕГРУЗКА new Function

```
type ConstructedFunction<TArgNames extends string[], TBody extends string> =
    (<TArgValues extends number[]>(
        ...args: TArgValues
      ) => FunctionResult<TBody, TArgNames, TArgValues>)
    & ((...args: unknown[]) => unknown)

```

```
declare global {
  interface FunctionConstructor {
    new <TArgNames extends string[], TBody extends string>(
      ...params: [...TArgNames, TBody]
    ): ConstructedFunction<TBody, TArgNames>;
  }
}
```

ВЫВОД

На основе имен формальных параметров и значений фактических мы формируем начальное состояние контекста выполнения. Это позволяет вычислять результат функции с произвольными параметрами на уровне типов.

ПОЛНЫЙ КОД РЕШЕНИЯ

▶️ TypeScript Playground

TypeScript Download Docs Handbook Community Tools Search Docs

Playground TS Config Examples Help Settings

v5.8.3 Run Export Share

```
1   //   _______  __    _    __  __ ____  _     _____ ____
2   //  | ____\ \/ /   / \  |  \/  |  _ \| |   | ____/ ___|
3   //  |  _|  \  /   / _ \ | |\/| | |_) | |   |  _| \___ \
4   //  | |___ /  \  / ___ \| |  | |  __/| |___| |___ ___) |
5   //  |_____/_/\_\/_/   \_\_|  |_|_|   |_____|_____|____/
6
7
8   // const result: 100500
9   const result = eval("(888 + 111 + 1) * (5 * 20) + 500");
10
11  console.log(result); // 100500
12
13  /**
14   * Calculates the square of an orthogonal parallelepiped
15   **/
16  const square = new Function('a', 'b', 'c', `
17    const s1 = a * b;
18    const s2 = b * c;
19    const s3 = a * c;
```

.JS .D.TS Errors Logs Plugins

```
"use strict";
Object.defineProperty(exports, "__esMo
exports.result = exports.s = void 0;
const result = eval("(888 + 111 + 1) *
exports.result = result;
console.log(result);
const square = new Function('a', 'b',
  const s1 = a * b;
  const s2 = b * c;
  const s3 = a * c;

  return (s1 + s2 + s3) * 2;
`);
const s = square(11, 4, 3);
exports.s = s;
console.log(s);
```

ВЫВОД

- Система типов TS полна по Тьюрингу
- Она представляет собой функциональный ЯП
- В ней реализуемы интерпретаторы других ЯП

ДЛЯ САМОСТОЯТЕЛЬНОГО ИЗУЧЕНИЯ

1. type-challenges – задачки на типы
2. ts-toolbelt, utility-types, SimplyTyped – библиотеки утилитных типов
3. Assembly interpreter in typescripts type system
4. Doom in TypeScript types
5. ts-pattern – практическое применение

СПАСИБО ЗА ВНИМАНИЕ

Приглашаю в дискуссионку обсудить типы