FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Math expression evaluator for literal types in TypeScript
Student:	Bc. Tat Dat Duong
Supervisor:	Ing. Jaroslav Šmolík
Study program:	Informatics
Branch / specialization:	Web Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2023/2024

Instructions

Template literal types [1], introduced in TypeScript 4.1, expand on string literal types for narrowing down a type to a particular string constant, with the ability to expand into many string literal types.

1. Analyze and describe relevant constructs of the TypeScript type system (concatenation, recursive types, conditional types etc.)
2. Implement a typesafe math expression evaluator with a set of basic operations, using a string literal type both as the input and output of the evaluator.
3. Pick appropriate tools for testing type annotations and ensure the validity of the evaluator with functional tests.
4. Discuss the practical uses of implemented meta types and theoretical and practical shortcomings of the TypeScript type system.
5. Publish the implementation as an open-source TypeScript library, which can be used for meta-programming, including source code and corresponding documentation.

[1] https://www.typescriptlang.org/docs/handbook/2/template-literal-types.html

---

Electronically approved by Ing. Jaroslav Kuchař, Ph.D. on 6 January 2023 in Prague.

Master's thesis

MATH EXPRESSION EVALUATOR FOR LITERAL TYPES IN TYPESCRIPT

Bc. Tat Dat Duong

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Jaroslav Šmolík
February 12, 2023

Czech Technical University in Prague
Faculty of Information Technology
© 2023 Bc. Tat Dat Duong. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Duong Tat Dat. *Math expression evaluator for literal types in TypeScript.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

List of Tables

List of code listings

Chtěl bych poděkovat především sit amet, consectetuer adipiscing elit. Curabitur sagittis hendrerit ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue.

Declaration

FILL IN ACCORDING TO THE INSTRUCTIONS. VYPLŇTE V SOULADU S POKYNY. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Curabitur sagittis hendrerit ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue. Donec ipsum massa, ullamcorper in, auctor et, scelerisque sed, est. In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Pellentesque pretium lectus id turpis.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Curabitur sagittis hendrerit ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue. Donec ipsum massa, ullamcorper in, auctor et, scelerisque sed, est. In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Pellentesque pretium lectus id turpis.

In Prague on February 12, 2023 .....

Abstract

Fill in abstract of this thesis in English language. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue.

Keywords enter, commma, separated, list, of, keywords, in, ENGLISH

Abstrakt

Fill in abstract of this thesis in Czech language. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue.

Klíčová slova enter, commma, separated, list, of, keywords, in, CZECH

Summary

Summary section

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem.

Summary section

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa.

Summary section

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Summary section

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Summary section

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Lorem lorem lorem.

Seznam zkratek

TC39	ECMA International, Technical Committee 39
W3C	World Wide Web Consortium

Introduction

0.1 Prior Art

JavaScript is a dynamically typed programming language, where users do not need to assign types to a variable or a function and the type is inferred automatically by the JavaScript engine. This is a great feature of JavaScript, which lowers the barrier of entry to writing JavaScript code and allows developers to prototype and write code quickly, proven by the growth of popularity of JavaScript in the last decade, making it the most commonly used programming language according to the 2022 Stack Overflow Developer Survey [1].

However, dynamic typing has its own drawbacks, as it is harder to spot trivial errors in the code without running it beforehand and it is more difficult to refactor the code without breaking it, which often lead to poor software quality [2]. Proponents of static typing insist that static types allows developers to spot potential bugs and mistakes earlier during development and that it allows for better tooling, such as more rich code completion and refactoring tools.

There is an upcoming TC39 proposal for adding type annotations, broadly inspired by TypeScript syntax [3]. These annotation are only used for build-time tooling, these annotation are ignored in runtime and the proposal suggests these annotations to be erased by an additional build-step. Even though users can already provide static types using JSDoc right now, the syntax is not as clean as the proposed TypeScript-like syntax.

Regardless, there are many languages which aim to introduce static typing to JavaScript, such as Flow or TypeScript, or alternative languages which compile back to JavaScript, such as Elm or ReScript.

0.1.1 Elm

Elm is a functional programming language designed specifically for building web applications [4]. The language compiles to JavaScript and has a strong static Hindley-Milner based type system, which allows to infer types more often and reliably. Elm does not provide any escape hatches such as `any` in TypeScript, thus it is harder to write unsafe code, as the types must be valid in order for the code to be compiled.

Elm also includes a lot of quality-of-life improvements and benefits, for instance: enforced purity of functions, out of the box immutability, `case` pattern matching, JSON decoders and encoders for strict parsing, `Maybe` and `Result` monads for avoiding `null` and `undefined` references or its own virtual DOM implementation for efficient rendering of interactive user interfaces. Notably, the Elm Architecture, where the application code is organized into three parts: model, update and view [5], has greatly inspired other libraries and frameworks like Redux [6].

0.1.2 ReScript

ReScript is a programming language built on top of OCaml toolchain. Unlike Flow or TypeScript, ReScript is not a superset of JavaScript, instead the language compiles to JavaScript. ReScript was created as a spin-off from Reason programming language and accompanying BuckleScript compiler, aiming to vertically integrate and streamline the adoption barrier caused by the need to be familiar with multiple unrelated tools and toolchains [7].

The language aims to be more sound with more powerful type inference than TypeScript, borrowing the Hindler-Milner type system from OCaml implementation [8, 9], thus most of times the types can be inferred automatically without the need to annotate them explicitly, whereas TypeScript utilizes bidirectional type checking [10].

0.1.3 Flow

Flow is a static type checker for JavaScript [11, 12], which allows developers to annotate their code with static types. Flow is developed by Meta and is internally used in production by Facebook, Instagram and React Native. Type annotations in Flow are fully eraseable, which means that the type annotations can be fully removed from the Flow code to emit valid JavaScript code. The checking of these types is occurring at compile-time before removal in build-time. Flow is also a superset of JavaScript, which means any JavaScript code is a valid Flow code.

One the primary goals of Flow is to provide type soundness; the ability to catch every error that might happen in runtime at compile-time, no matter how likely it is to happen. This means, that a valid Flow code can provide developers some guarantees about the type a value has in runtime, at the expense of catching errors, which are unlikely to happen in runtime.

Both Flow and TypeScript are similar in regards to features at the time of writing. Most of the soundness differences between Flow and TypeScript has been addressed with the newer versions of TypeScript, even though soundness is a specific non-goal by the TypeScript team [13]. However, developers must opt-in to these features by setting `"strict"` to `"true"` in `tsconfig.json`, whereas in Flow these features are enabled by default.

0.1.4 TypeScript

TypeScript is a staticly typed programming language developed and maintained by Microsoft [14]. It is a language that compiles to JavaScript and adds static type checking to JavaScript [15]. Unlike Elm or ReScript, TypeScript is a syntactical subset of JavaScript, which means that

any valid JavaScript code can be a valid TypeScript code[1]. Similar to Flow, type annotation provided by the developer are fully eraseable either by the TypeScript `tsc` compiler or by other community build tools, such as `babel`[16], `esbuild`[17] or `swc`[18].

Type system in TypeScript is considered to be less sound and more forgiving, as soundness is stated as an explicit non-goal for the design team of TypeScript [13], emphasis on striking a balance between productivity and correctness. By default, TypeScript compiler is not strict and the language itself includes an escape hatch for developers to opt-out of type checking by using the `any` type or using `@ts-ignore` comment annotations. Nevertheless, with proper compiler configuration, the type system of TypeScript can be as sound as in Flow.

Both Flow and TypeScript support advanced features such as generics and utility types, with the latter supporting template string literal types and better support for conditional types, unlocking the potential of writing more expressive types, which this master thesis will further explore in more detail.

TypeScript has become the de-facto standard for writing JavaScript code with static types. With deep integration with Visual Studio Code [19], rich build ecosystem and high compatibility with existing JavaScript libraries and tools, TypeScript has become one of the fastest growing language according to 2022 Octoverse report by Github [20].

0.2 Typescript syntax

- Structural Typing - interfaces, objects
- Nominal Typing - classes, enums
- Generics
- Utility types
- Type inferrence, overriding types
- Template literal types

0.3 Typescript internals

- Checker
- Compiler
- AST

[1] With a lax compiler configuration

Appendix A

Nějaká příloha

Sem přijde to, co nepatří do hlavní části.

Bibliography

1. *Stack Overflow Developer Survey 2022* [online] [visited on 2023-01-29]. Available from: https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022.
2. FARD, Amin Milani; MESBAH, Ali. JSNOSE: Detecting JavaScript Code Smells. In: *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 2013, pp. 116–125. Available from DOI: 10.1109/SCAM.2013.6648192.
3. *ECMAScript Proposal: Type Annotations* [online]. 2023 [visited on 2023-01-29]. Available from: https://github.com/tc39/proposal-type-annotations.
4. *Elm - Delightful Language for Reliable Web Applications* [online] [visited on 2023-01-31]. Available from: https://elm-lang.org/.
5. *The Elm Architecture · An Introduction to Elm* [online] [visited on 2023-01-31]. Available from: https://guide.elm-lang.org/architecture/.
6. *Prior Art — Redux* [online]. 2022-02-07 [visited on 2023-01-31]. Available from: https://redux.js.org/understanding/history-and-design/prior-art.
7. *BuckleScript & Reason Rebranding* [online] [visited on 2023-01-29]. Available from: https://rescript-lang.org/blog/bucklescript-is-rebranding.
8. *Efficient and Insightful Generalization* [online] [visited on 2023-02-12]. Available from: https://okmij.org/ftp/ML/generalization.html.
9. *History — ReScript* [online]. 2022-02-12 [visited on 2023-02-12]. Available from: https://github.com/rescript-lang/rescript-compiler/blob/master/CREDITS.md.
10. *Reconstructing TypeScript, Part 0: Intro and Background* [online] [visited on 2023-01-24]. Available from: https://jaked.org/blog/2021-09-07-Reconstructing-TypeScript-part-0.
11. CHAUDHURI, Avik; VEKRIS, Panagiotis; GOLDMAN, Sam; ROCH, Marshall; LEVI, Gabriel. Fast and Precise Type Checking for JavaScript. *Proceedings of the ACM on Programming Languages* [online]. 2017, vol. 1, 48:1–48:30 [visited on 2023-01-29]. Available from DOI: 10.1145/3133872.

12. *Flow* [online]. 2023 [visited on 2023-01-29]. Available from: https://github.com/facebook/flow.

13. *TypeScript Design Goals* [online] [visited on 2023-01-29]. Available from: https://github.com/microsoft/TypeScript/wiki/TypeScript-Design-Goals.

14. *TypeScript: JavaScript With Syntax For Types* [online] [visited on 2023-02-01]. Available from: https://www.typescriptlang.org/.

15. *Documentation - TypeScript for JavaScript Programmers* [online] [visited on 2023-01-14]. Available from: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

16. *Babel · The Compiler for next Generation JavaScript* [online] [visited on 2023-02-01]. Available from: https://babeljs.io/.

17. *Esbuild - An Extremely Fast Bundler for the Web* [online] [visited on 2023-02-01]. Available from: https://esbuild.github.io/.

18. *SWC - Rust-based Platform for the Web* [online] [visited on 2023-02-01]. Available from: https://swc.rs/.

19. *Visual Studio Code - Code Editing. Redefined* [online] [visited on 2023-02-01]. Available from: https://code.visualstudio.com/.

20. *Octoverse 2022: The State of Open Source* [online] [visited on 2023-01-29]. Available from: https://octoverse.github.com/.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF