令和 6 年度 卒業論文

静的解析と実行時性能に焦点を当てた JavaScript コンパイラと TypeScript チェッカ

A JavaScript compiler and TypeScript checker with a focus on static analysis and runtime performance

鈴鹿工業高等専門学校

電子情報工学科

青山研究室

糸川　倫太朗

指導教員: 青山　敏弘

令和 7 年 1 月 17 日

概要

ここに要旨を書く！

目次

第１章　はじめに

第 2 章　概要

第 3 章　decaf の実装

decaf は Rust を用いて実装された．TypeScript のパースは `oxc_parser`[1] を参考に実装した．また型の推論と検査は `tsc`[2] を参考に実装した．

decaf は `https://github.com/re-taro/decaf` からインストールが可能である．Rust のパッケージマネージャである `cargo` があれば，以下のコマンドでインストールが可能である．

Listing 3.1: decaf のインストール

```
$ git clone git@github.com:re-taro/decaf.git
$ cd decaf
$ cargo install --path .
$ decaf --help
```

3.1　型検査

decaf は任意の TypeScript(`.tsx?`) ファイルを入力として受け取り，型検査する．

3.2 ように実行すると, 標準出力として 3.3 のような結果が得られる．

Listing 3.2: decaf の型検査

```
$ decaf check <file>\.tsx?$
```

Listing 3.3: decaf の型検査結果

```
error:
    ┌── all.tsx:726:3
    │
726 │ obj.prop2;
    │ ^^^^^^^^^ No property 'prop2' on { prop: 3, prop2: 6 } | { prop: 2 }

---

Diagnostics: 446
```

1) `https://docs.rs/oxc_parser/latest/oxc_parser/`
2) `https://github.com/microsoft/TypeScript/blob/v5.1.3/src/compiler/checker.ts`

```
10    Types:  5780
11    Lines:  2239
12    Cache read:  285.954µs
13    FS read:  169.096µs
14    Parsed in:  8.294198ms
15    Checked in:  4.887375ms
16    Reporting:  204.832µs
```

3.1.1 decaf が型を検査する手順

decaf が型検査する手順は以下の通りである．

1. 入力ファイルをパースし，AST[3] を生成する．
   (a) ここで変換される AST は ESTree[4] や swc[5] のものとは異なり，decaf 独自の AST である．
2. 生成された AST を decaf の型検査機が解釈しやすい形に変換する．
   (a) この変換により，型検査に他の言語で実装されているモダンなアルゴリズムを適用できるようになる．
   (b) これを decaf では TypeID と呼んでいる．
   (c) この機構を decaf では変換機と呼んでいる．
3. 型検査する
   (a) decaf の変換機はプログラムの値や構造を型として，関数やブロックの振る舞いをイベントとしてエンコードする
      i. 型やイベントは AST の走査中に生成され，decaf の型検査機に渡され，使用される．
      ii. decaf は型を**その値が取りうる可能性の集合**として捉える．
      iii. 条件分岐やループに差し掛かった時，そのブロック内で取りうる型の集合を広げる．
         - 3.4 の例では,`response` の宣言時，型は `Response` であり，プロパティ `data` の型は `any`[6] である．
           – 条件 `response.data instanceof Date` が真である場合，`response.data` の型は `Date` である．
           – 条件 `response.data instanceof Date` が偽である場合，`response.data`

3) 抽象構文木のこと
4) ECMAScript の抽象構文木の実質的な標準
5) https://github.com/swc-project/swc
6) ここで `any` 型は，便宜上型の全集合とする．

の型は `Date` 型の補集合を取った型である．

(b) AST を走査しながら型の集合を広げていくことで，実行時に起こりうるすべての可能性を型チェック時に考慮できる．

i. 従来の TypeScript Compiler は分岐の評価は，実行されるかされないかのみを考慮している．

ii. decaf は分岐を非決定論的に扱い，すべての分岐を評価し[7]型を拡大する．

Listing 3.4: 例

```
1  async function f(name: string) {
2    const response = await fetch(`/post/${name}`).then(res => res.json());
3    if(response.data instanceof Date) {
4      return response.data;
5    } else {
6      return response.data;
7    }
8  }
```

7) 条件が自明で真である場合などは除く．

第 4 章　実験

4.1　概要

4.2　実験内容

4.3　考察

4.4　decaf の改善点

第 5 章　まとめ

謝辞

最後に，電子情報工学科教授の青山敏弘教授からは本研究についての適切なご指導を賜りました．感謝を申し上げます．