型システムのしくみ輪講（第一章）

古殿直也 2025-05-09

輪講でやること（復習）

ここで決めたものを抜粋する
https://www.notion.so/pepabo/1e371085cfd98046b18ac3045d276174

- 担当者を持ち回りでやる（今回はぼく）
- 担当範囲が決まっている（今回は第一章）
- 担当範囲の解釈を説明する（以降のスライドで説明します）
- 参加者はツッコミとか質問とかをする。すべての参加者が少なくとも一回は質問すること
    - いつでも質問してOK。担当者の発言を遮ってください
- 終わらなかったら次回に繰り越し

このように本のトピックを議論して理解していきましょう

第1章 型システムとは

型システムはプログラムのOK / NGを判定する

以下のような OK / NG の判定基準を定めるのが型システム

```
1 + true // よくない。NGと判定したい
1 + 2 // 良さそう。OKと判定したい
```

判定基準は（すなわち型システムは）どのような基準で定義するのが良いだろうか

誰がどのように判定基準（型システム）を定義するか

- 「実はとても難しい問題」
- 定義の方針を考えるにあたっての歴史的背景としてプログラムの未定義動作がある

1.1 プログラムの未定義動作

CやC++には未定義動作がある

言語仕様の定義では、プログラムの動作の一部を仕様で定義しない、とする場合がある

未定義動作の定義をC17（C言語の仕様）のドラフトから定義を引用

> behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this document imposes no requirements

Geminiによる翻訳

> 移植性のない、または誤ったプログラム構成要素や誤ったデータを使用した場合の振る舞いであって、本ドキュメントが何の要求も課さないもの。

未定義動作の例

- 無効なポインタを参照する
- 最大の整数に1足した値を計算する
- 配列の範囲外を参照する
- そのほかもっと込み入ったものも（C17のドラフトを "undefined behavior"で検索するとたくさん出てくる）

未定義動作をするプログラムを書くと、プログラマは辛い

未定義動作をするプログラムは、その振る舞いが仕様で定められていない

- 派手に壊れるかもしれない
- それっぽく動くかもしれない
- しばらくはうまく動くけど、ある日突然壊れるかもしれない

実際のところはどうか

- それっぽく動作する
- OSの保護機能でプロセスが落ちる
    - segmentation faultで落ちるのはこれ
- などなど

辛いのでなんとかしたい

1.2 歴史的解決策としての「型安全性」

未定義動作への対処

- コンパイル時に型検査する
  - この本で扱うような型システムが活躍するのはこちらのアプローチ
  - 言語の例: OCaml, Haskell, Go, ...
- 実行時にたくさん検査する
  - 言語の例: JavaScript, Ruby, Smalltalk, ...

コンパイル時に型検査するアプローチ

- 未定義動作になるようなプログラムはプログラムと認めない
- 型検査器でプログラムを検証する。検証が通ったならば、プログラムは決して未定義動作にならないことを型検査器と言語仕様に求める

実行時にたくさん検査するアプローチ（おまけ、蛇足）

- 未定義動作に踏み込みかけたときに例外を投げたりする
- 例外を投げるという動作を定義しているので未定義動作はなくせる
- C言語でも、どんなときに未定義動作になるかは定義されている。それを愚直に実行時に検証すればよい

型安全性は「型システム」と「プログラムの振る舞いの定義」の間の関係

TaPLでは型安全性を「型検査器がOKと判定したプログラムは、定義されていない状態（行き詰まり状態、stuck）にならない」という性質としている

以下のように解釈しました

- 型システム： 型検査の仕様
- 定義されていない状態：未定義動作
- 型安全性：型検査器がOKと判定したら未定義動作にならないという、型システムとプログラムの振る舞いの定義の間の関係

まとめ: 未定義動作に型システムで対処できる

- 未定義動作は辛い
- 型検査をして、未定義動作するかもしれないプログラムはコンパイル時に弾く
- 結果的に、プログラムを動かす段階では絶対に未定義動作に陥らないことを保証できる

1.3 本書における型システムと型安全性

前提の確認

- TypeScriptプログラムは普通、JavaScriptプログラムに変換されてからJavaScriptとして実行される
- JavaScript言語には未定義動作がない（少なくともC言語にあるような未定義動作はなさそう）
  - https://stackoverflow.com/questions/14863430/does-javascript-have-undefined-behaviour
    - ここにあるように、implementation defined な動作はある
    - 細かい話だし、どちらに転んでも型システムのしくみを感覚として理解するためには問題にならなさそう

TypeScriptでは未定義動作ではなく「望ましくなさそうな動作」に対処するために型検査する

- JSには未定義動作がない
  - 何かしら動作が定義されている。**望ましい動作かはさておき**
    - 実行時に検証して、例外を投げる
    - 不思議な計算結果を無理やり返す
    - 実装に依存するとしている（これはCの仕様でもでもやられている）
- TSでは未定義動作を防ぐことではなく、「望ましくなさそうな動作」を防ぐことを目的に型検査をする
  - `1 + true` を計算する際のプログラムの振る舞いはJSで定義されているけど望ましくなさそう

1.4 実装する型検査器のプログラムについて

型検査器と対象言語

型検査器が検査対象とする言語を「対象言語」と呼ぶ

例：

- `tsc` の対象言語は TypeScript
  - JavaScriptではない
- `go` の対象言語は Go

今後実装していく型検査器の対象言語はTypeScriptのサブセットである

サブセットであるとはどういうことだろうか

例えば arith.tsで実装する型検査器の対象言語 (arith-langとしましょう）が、TypeScript のサブセットであるとはどういうことか

ぼくの持っている答え

arith-langの（型検査が通る）プログラムはすべて、TypeScriptの（型検査が通る）プログラムでもある、ということ。

言い換え: arith.tsで実装する型検査器がOKとするプログラムはすべて、tsc（TypeScriptの型検査器）がOKとする、ということ。

クイズ: 最小の対象言語を持つ型検査器はどんな型検査器でしょう

クイズの答え

どんなプログラムが来てもNGとする型検査器。

型検査器の実装を追加して、対象言語を徐々に大きくしていく

- (arith) 真偽値・数値、条件演算子・足し算
- (basic) arith + 関数と変数定義
- (obj) basic + オブジェクト
- (rec) obj + 再帰関数 + 再帰型
- ...
- TypeScript

演習問題

みんな動かせましたか？まだなら早めに試すのが吉です。苦戦したらSlackであらかじめヘルプを求める、その辺にいる人を捕まえて一緒になんとかしてもらうなどしましょう

次回

- 2章の全てを予定しています
- https://www.notion.so/pepabo/1e371085cfd98046b18ac3045d276174

参考文献

- C17 のドラフトをここから引用: https://www.open-std.org/jtc1/sc22/wg14/www/projects#9899