Rust 世界の二つのモナド

Rust でも **do**記法をして
プログラムを直感的に記述する件について

konn
関数型まつり 2025
2025-06-14

Slides available: `https://u.konn-san.com/qdo`

自己紹介

- 石井 大海（いしい ひろみ） / konn (Twitter: @mr_konn)
- 現職：株式会社 Jij ソフトウェア開発チーム（2024/11〜）
  - **Rust で数理最適化の埋込ドメイン特化言語 (EDSL) を開発**しています
    - **スポンサーブース出展中！**興味のある方もない方もぜひ！
- 前職では **Haskell** で数値計算向け **EDSL** を開発
  - EDSL 開発に縁があるね
- 今日の話題は EDSL の設計に欠かせない **do 記法**のお話です

本題

Rust 世界の二つのモナド

Rust でも **do**記法をして
プログラムを直感的に記述する件について

Rust 世界の二つのモナド

Rust でも **do記法**をして

プログラムを直感的に記述する件について

🤔 **モナド**とか **do記法**とかって

Haskeller が使ってるやつ？

🤔 Rust と関係あるの？

二つってなに？

お答えしましょう！

今回の内容

1. **`qualified_do`: Rust でも (Qualified)do する件について**
   - Haskell の **`QualifiedDo` 記法**を移植した `qualified_do` クレートを紹介
   - **Rust でも do記法は便利そ〜**と思ってもらう
2. **do記法でわかる**かもしれない**！モナド**
   - **「do記法が使える構造」**として**モナド**を捉えなおす
   - **Functor** や **Applicative** なども「**do記法の機能をどこまで使えるか**」という観点で説明
   - **GATs** を使って **Rust** で Functor / Applicative / Monad を定式化するトリックも紹介
3. **Rust 世界の二つのモナド 〜資源は大切に〜**
   - Rust や Linear Haskell など**資源の使い方に厳しい世界**では、**モナド階層が二つに分岐**することを紹介
   - Rust と Linear Haskell では**型システムの違い**によって**何がモナドになれるかに差異**が生まれることを紹介

おしながき

1. **`qualified_do`**: Rust でも (Qualified)do する件について
2. do記法でわかるかもしれない！モナド
3. Rust 世界の二つのモナド ～資源は大切に～

`qualified_do`:

Rust でも (Qualified)do する件について

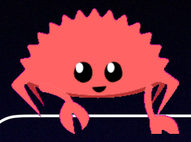
qualified_do クレート

- **qualified_do**: Haskell の QualifiedDo + ApplicativeDo 拡張相当を Rust の proc macro として実装したもの
  - 紹介記事（Zenn）：https://u.konn-san.com/zenn_qdo
  - Crates.io: https://crates.io/crates/qualified_do
- 半年前から仕事で Rust を使うようになり、**do記法が使いたい**！となって開発（業務でも一部利用中）
  - do 記法をサポートする crate は複数あったがどれもメンテナンスされておらず、またdo 記法の実装が**決め打ち**だった

用例1：proptest の Arbitrary

- proptest: Rust の性質ベーステストライブラリ
  - classify 等の統計機能を除けば必要な機能が一通り揃っている
- データ生成の Arbitrary trait は derive できるが、手で書く場合はが prop_compose! マクロと .map() や .flat_map() で頑張る
  .

余談：Rust と Haskell のマクロ比較



Rust: declarative / proc macro

- declarative macro = macro_rules! のやつ
- **proc macro = トークン列をトークン列に変換**する関数
  - 中間出力は**構文的に不正でもよい**
  - 構文をシームレスに拡張しやすい
    - トークンは共通なのである程度フォーマッタとも相性がいい
  - derive 記法も実体は proc macro
- **実際**：syn クレートでトークン列をマップ
  1. 独自の情報をトークン列からパーズ
  2. 中間形式に変換
  3. 中間形式にトークン列への変換を実装
  - 結構快適
- **難点**：マクロの**修飾対象以外の型・traitの定義情報が取れない**ので全てを**定義サイトに集約**する必要

Haskell: Template Haskell

- **構文木生成**：式・定義・パターン・型
  - 環境に問い合わせて AST を生成
    - **他の型**・クラス・関数の**情報を取得可能**
    - 直接構文木を書く代わりに**クォート**を使って `[| $x + $x |]` のように書ける
  - 式クォートには**型が付いている版**もある
- **準クォート**を使って独自の構文も定義可能
- **難点**：
  - 全てがクォートで書ける訳ではなく、**構文木は GHC のメジャーバージョンごとに変わる**ので互換性が大変（compat libraryはある）
  - 準クォートは事実上全ての文字列が書けるので、**フォーマッタと相性がわるい**

閑話休題

`qualified_do` まとめ

- **qualified_do**: proptest, Vec, Option, Result 等を統一的に扱える
  - Fallible な計算については、? 記法と相補的に使える
    - ? と違い、**異なる型同士の計算をネスト**できる
  - まだクロージャの move との兼ね合いに改善の余地あり
- Rust の proc macro は結構面白い
  - Template Haskell みたいに他モジュールの型定義を reify できたらもっと嬉しいなあ

おしながき

1. **`qualified_do`**: Rust でも (Qualified)do する件について

do 記法でわかるかもしれない！モナド

😄do記法はべんりそう！

🤔でもなんで

そんなもん作ったの？

A. 毎回

```
i.into_iter().map().zip()
 .enumerate().try_fold()
        .and_then()
```

とかって書くのつらくないですか？😭

そこでモナド！

do記法とモナド まとめ

おしながき

2. do記法でわかるかもしれない！モナド

Rust 世界の二つのモナド

～資源は大切に～

二つのモナド：まとめ

おしながき

3. Rust 世界の二つのモナド ～資源は大切に～

まとめ

まとめ

- **モナド = 無制限のdo記法を使える構造**
  - Functor = `do { x ← mx ; pure (f x) }` 型のdo記法が使える
  - Applicative = `do { x1 ← mx1 ; ...; xn ← mxn ; pure (f x1 ... xn)}` 型のdo記法 (mxi: 互いに独立)
  - Rust でも Generalised Associated Types (GATs) を使って「実装」にタグづけするトリックで実装可能
- Rust や Linear Haskell など線型・アファイン型のある世界では **Monad の階層が二つに分裂**する
  - **Data Functor**: 渡された関数を**複数回**呼び出し得る
  - **Control Functor**: 渡された関数は**（Rust ならば高々）一回しか**使われない
    - `Maybe` (`Option`) は**Linear Haskellでは Control Monad にならない**が、Rust では **(Affine) Control Monad** に！
- **QualifiedDo 記法**：複数の**「モナドっぽい」**クラスに do記法を使えるようにする Haskell の言語拡張
  - `qualified_do: QualifiedDo` + ApplicativeDo + α を Rust で使うための proc macro `qdo!` を提供

参考文献

- Linear Haskell の論文
- Shake before Building の論文
- Tale of Two Functors の記事
- ApplicativeDo や QualifiedDo のリファレンス
- わたしの Zenn

御清聴

ありがとう

ございました

Any Questions?

- **モナド = 無制限のdo記法を使える構造**
  - Functor = `do { x ← mx ; pure (f x) }` 型のdo記法が使える
  - Applicative = `do { x1 ← mx1 ; ...; xn ← mxn ; pure (f x1 ... xn)}` 型のdo記法 (mxi: 互いに独立)
  - Rust でも Generalised Associated Types (GATs) を使って「実装」にタグづけするトリックで実装可能
- Rust や Linear Haskell など線型・アファイン型のある世界では **Monad の階層が二つに分裂**する
  - **Data Functor**: 渡された関数を**複数回**呼び出し得る
  - **Control Functor**: 渡された関数は**（Rust ならば高々）一回しか**使われない
    - `Maybe` (`Option`) は**Linear Haskellでは Control Monad にならない**が、Rust では **(Affine) Control Monad** に！
- **QualifiedDo 記法**：複数の**「モナドっぽい」**クラスに do記法を使えるようにする Haskell の言語拡張
  - `qualified_do`: `QualifiedDo` + ApplicativeDo + α を Rust で使うための proc macro `qdo!` を提供