

ほう自作言語ですか 大したものですね

～Nexusの場合～

2026/10/22

@自作プログラミング言語を自慢する会

びしょ～じょ

自己紹介



こんにちは、
びしょ〜じょです

- 株式会社HERP エンジニア
- プログラム言語がすき

本日のお題

自作言語を語ろう

<https://nexus-llm-lang.github.io>



ところで

7月にこんな話をしました

LLM に書かせる前提で設計された言語の設計について議論して。

… 回答を生成しています…



ところで

LLM 向け言語を作って



当然……

俺も作るぞ!!!!



ということで
作りました

ところで

ベンチマークをとって



Fun Facts: 生成される既存言語のコードはすでに高品質

LLM 向けの言語で爆コード生成!! と思っていたが

- ▶ 構文解析 → 解析・最適化 → コード生成 + 周辺ツール
- ▶ LLM に Rust で実装させたらすでにできちゃった!

おしまい

ご清聴ありがとうございました

Nexus

まあまあそう言わず.....

いいんじゃないですか、楽しいし

作って分かった:

- 生成品質は型エラーのようなdynamicなフィードバックや、
大量のコードベース(self-hostedな処理系)で向上
- AIに実装させたら数ヶ月で1x万溶けた
- 詳細はスライドをご覧ください

Nexus

- 名前..... AIに考えてもらった
- ロゴ..... AIに考えてもらった
- 構文..... おれ
- 意味論..... おれ
- 型システム..... 半々 🤔
- 実装..... AIにやってもらった



LLM 向け言語を作って



大まかな設計思想:

▶ リテラル上の情報

を増やしたい

- 全てラベル引数
- トップレベルは型を明示
- エフェクトシステム
 - 例外…継続破棄
 - 要求…継続 1 回
 - (mutli-shot は特別形式)
- 値の性質にマークをつける
 - linearity %
 - laziness @
 - 参照 &

▶ ツーリングを厚めに作る

- 処理系… 型エラー送出
- MCP
- LSP
- skills

▶ ランタイム安全性

- WASM ターゲット
main のエフェクト
→ permissions
- そのための高階エフェクト
Fs → PermFs → --dir .

Nexus

言語仕様として

- すべてラベル引数
- エフェクトシステム
- 線形型
- その他普通のやつ
 - パターンマッチング、多相.....

Nexus

処理系

- セルフホスティング
- wasm (wasmtime) のみサポート
 - main関数のエフェクトがランタイムの権限
- ツール同梱
 - LSP、fmt、lint、REPL、テストランナー、skills

エフェクトシステム

```
cap Mailer do
  fn send(addr: string, body: string) → unit
end

let signup = fn (email: string) → unit require { Mailer } do
  Mailer.send(addr: email, body: "Welcome!")
  return ()
end
```

エフェクトシステム

```
let console_mailer = handler Mailer require { Console } do
  fn send(addr: string, body: string) → unit do
    Console.println(val: "[mail] " ++ addr ++ ": " ++ body)
  return ()
end
end
```

```
let mock_mailer = handler Mailer do
  fn send(addr: string, body: string) → unit do
    return ()
  end
end
```

エフェクトシステム

- **capability**が仕様
- ハンドラで実装



```
cap Mailer do
  fn send(addr: string, body: string) → unit
end
```

```
let signup = fn (email: string) → unit require { Mailer } do
  Mailer.send(addr: email, body: "Welcome!")
  return ()
end
```



```
let console_mailer = handler Mailer require { Console } do
  fn send(addr: string, body: string) → unit do
    Console.println(val: "[mail] " ++ addr ++ ": " ++ body)
    return ()
  end
end
```

エフェクトシステム

- **capability**が仕様
- ハンドラで実装
- main関数のエフェクト
がランタイムの**権限**

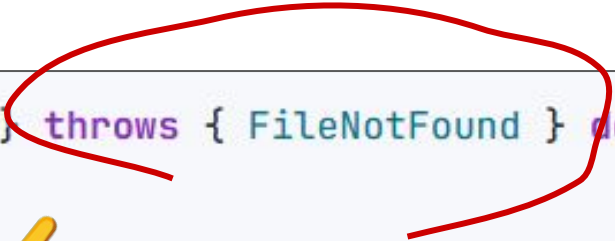
※ wasmtimeは標準入出力を
デフォルトで通すので
PermConsoleは特に制限なし

```
let main = fn () → unit require { PermConsole } do  
  inject stdio.system_handler do  
    inject console_mailer do  
      signup(email: "alice@example.com")  
    end  
  end  
  inject mock_mailer do  
    signup(email: "bob@example.com")  
  end  
  return ()  
end
```

[mail] alice@example.com: Welcome!

エフェクトシステム

```
let peek = fn (path: string) → unit require { Fs } throws { FileNotFound } do  
  let %h = Fs.open_read(path: path)  
  return ()  
end
```



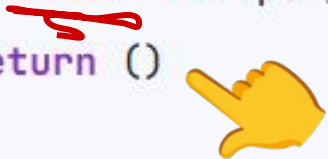
例外も別途トラック

線形型...

```
let peek = fn (path: string) → unit require { Fs } throws { FileNotFound } do
  let %h = Fs.open_read(path: path)
  return ()
end
```

線形型...

```
let peek = fn (path: string) → unit require { Fs } throws { FileNotFound } do  
  let %h = Fs.open_read(path: path)  
  return ()  
end
```



あ

linearity error: linear binding '%h' is never consumed [E2007]

線形型...?

```
let peek = fn (path: string) → unit require { Fs } throws { FileNotFound } do
  let %h = Fs.open_read(path: path)
  return ()
end
```

線形性

線形~~型~~性！

```
let peek = fn (path: string) → unit require { Fs } throws { FileNotFound } do
  let %h = Fs.open_read(path: path)
  return ()
end
```

sigils

値の性質をsigilで明示

%線形、移譲 &借用

```
let %b = Box(val: 41)
let x = peek(b: &b)
let y = consume(b: %b)
```

@遅延 @force

```
let @p = slow(n: 1)
let @q = slow(n: 2)
let r = @{ a: p, c: q }
```

sigils

- @遅延は線形でもある
- @遅延&線形借用はthunkに入れられない
- 関数引数でのみ &借用 可能

sigils

- @遅延は線形でもある
- @遅延&借用はthunkに入れられない
- 関数引数でのみ &借用 可能
- @遅延はエフェクトを持ち運べる！

?!

sigils

@遅延は
エフェクトを
持ち運べる！

- @forceした
箇所に
エフェクト

```
cap Db do
  fn fetch(key: string) -> string
end

// thunk を作るだけ: Db は要らない
let lookup = fn (key: string) -> @(string require { Db }) do
  let @v = Db.fetch(key)
  return v
end

// force する場所で Db を精算
let run = fn (th: @(string require { Db })) -> string require { Db } do
  return @th
end

// Db が無い場所で force -> E2001
let bad = fn (th: @(string require { Db })) -> string do
  return @th
end
```

仕様その他

FFIができるぞい

```
import { Console }, * as stdio from "std:stdio"  
import { from_i64 } from "std:str"  
import external "add.wasm"
```



```
external add = "add" : (a: i64, b: i64) -> i64
```

```
let main = fn () -> unit require { PermConsole } do  
  inject stdio.system_handler do  
    Console.println(val: "3 + 4 = " ++ from_i64(val: add(a: 3, b: 4)))  
  end  
  return ()  
end
```

```
(module  
  (func (export "add")  
    (param i64 i64) (result i64)  
    local.get 0  
    local.get 1  
    i64.add))
```

仕様その他

- 型システム
 - レコードのsubtypingとかgrouped exnなど
 - <https://nexus-llm-lang.github.io/latest/spec/type-system-formal/>
 - 鬼のようにKaTeXを書かせたら鬼重いページに
- 標準ライブラリ厚め
 - パーザー...JSON、YAML、TOML、HCL...
 - テストライブラリ...power-assert like, PBT

ランタイム

wasm(wasmtime)がターゲット

- mainのrequirementsがwasmtimeの権限
 - `PermFs` → `--dir=.`
 - `PermNet` → `-S http -S inherit-network`
- 出力形式(core module, component)は自動選択

ランタイム

wipが多い😅

- ~~AI「やりました」→できてへんやんけ!💢~~
- メモリ管理→それをしません
 - perceusにする途中
- wasmtimeの実験的な機能に依存
 - バージョンアップで消えたり不安定だったり
 - arm64版はstack switching対応してないのかよ
 - threads p3止まり→ v47で消えて並列計算がオワッタ

perceus... <https://www.microsoft.com/en-us/research/wp-content/uploads/2020/11/perceus-tr-v1.pdf>

アプリケーション



アプリケーション

コンパイラ

- フツーのパイプライン
 - 構文解析→型検査→最適化→コード生成
- その他ツール込み
 - LSPなどなど
- だいたい11万行
- ビルドに**24秒**！
- Apple Silicon M5、wasmtime 49.0.2 on Rosetta2

反省

- 覚えてない仕様が多い 😊
 - おれもAIも記憶が曖昧
 - ADRを書こう
- Gemini 2.8 → Opus 4.6 → Fable 5.0 → Opus 5.5
 - AIの進化すげ～
 - (相対的に)弱いモデルの書き散らしたコードの処遇

おわりに

みんなも最強の自作言語を作るぞ!!!