fast lean, ast-grep

Terminal Night #1 / fujitani sora

自己紹介

- **fujitani sora** / @_fs0414
- 👤 2001（24）
- 🏢 株式会社トリドリ・software engineer
- 🎤 技育CAMPの公式メンター
- 💪 TSKaigiの運営
- 💻 Terminal Use : NeoVim, WebTerm, ClaudeCode,
- 📱 Photo, Post 👌

最近のTips

CodeFormatterに凝っています。
Prettierのバグを直したり、自作のRust製Ruby Code Formatterを公開したり
https://github.com/fs0414/rfmt

コードベースから特定のパターンを検索したい

ex: Prettierのコードから、isNode関数の呼び出しを全て見つけたい

- 文字列で検索する
- 正規表現で検索する

🤔

文字列ベースの検索

```
$ grep "isNode" src/
```

- フォーマットに依存
- コメントや文字列リテラルも検出
- コードの構造は考慮しない

```
// すべてマッチする
isNode(node, ["type"])           // ← 検索対象
// isNode is deprecated          // ← コメント
const text = "isNode function"   // ← 文字列リテラル
```

正規表現検索

```
$ grep -E "isNode\(.*\[.*\]\)" src/
```

- 改行を跨ぐパターンには対応しづらい
  grep -Pzo などのoptionを使えば可能ではある
- ネストした構造の表現が複雑
- パターンが長くなる傾向

```
// マッチする
isNode(node, ["type1", "type2"])

// マッチしない
isNode(node, [
  "type1",
  "type2"
])
```

👀 コードをASTに変換しての検索

AST (Abstract Syntax Tree) = 抽象構文木

コードをASTにParse
→ ASTから「関数呼び出し」のNodeみを検索可能

```
// コードをASTとして解析
isNode(node, ["type"])
```

```
# AST表現（簡略化）
CallExpression:
  callee:
    name: "isNode"
  arguments:
    - Identifier: "node"
    - ArrayExpression:
        - "type"
```

ast-grep : ASTベースの検索ツール

ast-grepの特徴

- コードの構造検索・置換を行う
- 20種類以上のプログラミング言語に対応

> *構文解析機能を備えたgrep/sedのようなものと考えてください。*
> *AST（抽象構文木）に基づいてコードを検索・修正するためのパターンを記述でき、数千ものファイルに対してインタラクティブに操作を行うことが可能です。*

https://ast-grep.github.io/

ast-grepが理解するコード構造

```
// 検索対象コード
isNode(node, ["type1", "type2"])
```

```
# ast-grepのパターンマッチング
pattern: isNode($NODE, $TYPES)

# マッチする箇所
$NODE  → node
$TYPES → ["type1", "type2"]
```

実践例1 - フォーマット非依存検索

ref : github.com/prettier/prettier

Parse結果としての構文木は同一
WhiteSpaceなどの情報はParserの字句解析時点で除去される

```
// パターン1: 1行
isNode(node, ["sequence", "mapping"])

// パターン2: 複数行・インデント
isNode(node, [
  "documentHead",
  "documentBody",
  "flowMapping",
  "flowSequence",
])

// パターン3: スペースなし
isNode(node,["type"])
```

フォーマット非依存検索 - ast-grepの場合

```
$ ast-grep --lang js --pattern 'isNode($NODE, [$$$])' src/language-yaml/
```

すべてのフォーマットを検出

結果:

```
✅ isNode(node, ["sequence", "mapping"])
✅ isNode(node, [
     "documentHead",
     "documentBody",
     ...
   ])
✅ isNode(node,["type"])
```

YAMLルールで高度な検索

```
$ ast-grep scan --rule debug-rule.yml src/
```

- メッセージをカスタマイズ
- 重要度を設定
- 複数ルールを管理しやすい

```yaml
# debug-rule.yml
id: find-debug-console
language: js
rule:
  pattern: if ($DEBUG) console.$METHOD($$$)
message: Debug console statement found
severity: warning
```

実践例3 - リファクタリング支援

配列の最後の要素へのアクセス

prettierコードベースには両方が混在している

```
// 古いスタイル（ES5）
arr[arr.length - 1]
items[items.length - 1]

// 新しいスタイル（ES2022）
arr.at(-1)
items.at(-1)
```

リファクタリング候補の検出

実行:

```
$ ast-grep scan --rule modernize-array.yml src/language-yaml/utils.js
```

```
id: modernize-array-access
language: js
rule:
  any:
    - pattern: $ARR[$ARR.length - 1]
    - pattern: $ARR.slice(-1)[0]
message: Consider using modern array.at(-1) syntax
```

検出結果とリファクタリング

utils.js での検出結果

```
help[modernize-array-access]:
     ┌─ src/language-yaml/utils.js:190:7
     │
190  │ lines[lines.length - 1] = [...lines.at(-1), ...words];
     │ ^^^^^^^^^^^^^^^^^^^^^^^^
     │

help[modernize-array-access]:
     ┌─ src/language-yaml/utils.js:250:7
     │
250  │ lines[lines.length - 1] = [...lines.at(-1), ...words];
     │ ^^^^^^^^^^^^^^^^^^^^^^^^
     │

help[modernize-array-access]:
     ┌─ src/language-yaml/utils.js:261:9
     │
261  │ words[words.length - 1] += " " + word;
     │ ^^^^^^^^^^^^^^^^^^^^^^^^
     │
```

3箇所のリファクタリング候補を発見

自動置換の実行

基本の置換コマンド:

```
$ ast-grep --lang js \
  --pattern '$ARR[$ARR.length - 1]' \
  --rewrite '$ARR.at(-1)' \
  src/language-yaml/utils.js
```

重要なオプション:

- --pattern: 検索パターン
- --rewrite: 置換パターン
- --update-all: プレビューではなく実際に更新
- --interactive: 対話的に確認しながら置換

実際の置換結果（–update-all適用後）:

```
// before
const lastLine = lines[lines.length - 1];
return arr[arr.length - 1] || defaultValue;
const last = words[words.length - 1];
```

```
// after
const lastLine = lines.at(-1);
return arr.at(-1) || defaultValue;
const last = words.at(-1);
```

総じて好きなところ

- 検索パターンをymlで管理, 共有できる
  - 情報が永続化して読み取り可能であることはAgenticCodgingにとって重要だと思っている
- 構文木で一致するコードの一括置換が可能である
- 特定言語に依存しない汎用ツールである
  - 構文解析可能であれば
  - 個別のCustomLintを理解するよりも汎用的な仕組みかな？と思う
- astに関する知識の隠蔽, toolとしてのinterfaceが優れているなと思う

see you later