# ミニマムなグラフ書き換え言語のコンパイラと仮想機械

https://github.com/sano-jin/lmn-alpha

Jin SANO

# 概要



flat lmntal の **ミニマム** な
コンパイラと仮想機械を OCaml で実装した

- 全部合わせても **1 KLOC** 以内
- モジュール分割を行い，**拡張性**・**再利用性** も高い

# 成果物



ソースコード：https://github.com/sano-jin/lmn-alpha

ドキュメント：https://sano-jin.github.io/lmn-alpha/

1. **導入方法と実行例**
2. 既存のツールとの比較
3. 実装の概要
4. 今後の課題
5. まとめ

# 導入方法：README.md より抜粋



```
$git clone https://github.com/sano-jin/lmn-alpha
$cd lmn-alpha
$opam install .
$dune build
$./run example/append_dot.lmn -t
```

事前に **opam** が必要

# コンパイラが出力するコード例の抜粋



```
ret = append([a, b], [c]).

append_cons @@
R = append([H | T], L)
:- R = [H | append(T, L)].

append_nil @@
R = append([], L)
:- R = L.
```

example/append_dot.lmn

```
append_cons @@
    alloc [5]
--head:
    PeakAtom     [ 0 'append'_3 ]
    DerefAtom    [ 1 0 0 2 ]
    CheckFunctor [ 1 '.'_3 ]
--body:
    PushAtom     [ 3 '.'_3 ]
    PushAtom     [ 4 'append'_3 ]
    ReLink       [ 3 0 1 0 ]
    SetLink      [ 3 1 4 2 ]
    ReLink       [ 3 2 0 2 ]
    ReLink       [ 4 0 1 1 ]
    ReLink       [ 4 1 0 1 ]
    SetLink      [ 4 2 3 1 ]
    FreeAtom     [ 1 ]
    FreeAtom     [ 0 ]

append_nil @@
    alloc [3]
--head:
    PeakAtom     [ 0 'append'_3 ]
    DerefAtom    [ 1 0 0 0 ]
    CheckFunctor [ 1 '[]'_1 ]
--body:
    Connect      [ 0 2 0 1 ]
    FreeAtom     [ 1 ]
    FreeAtom     [ 0 ]
```

```
$./run example/append_dot.lmn -c
```

-c (compile only) オプションは,
中間命令列を出力する

# 解釈実行の例



```
$./run example/append_dot.lmn -t
0: ret(append([a, b], [c]))
----> append_cons
1: ret([a | append([b], [c])])
----> append_cons
2: ret([a, b | append([], [c])])
----> append_nil
3: ret([a, b, c])
Final state:
ret([a, b, c])
```

-t (Trace) オプションは,
実行途中結果をトレースする

本物の実行結果に
色をつけ加えている

1. 導入方法と実行例
2. **既存のツールとの比較**
3. 実装の概要
4. 今後の課題
5. まとめ

# lmntal-compiler・slim との LOC 比較



| | 使用言語 | KLOC |
|---|---|---|
| slim[*1] | C++ | 39.5 |
| lmntal-compiler[*2] | Java | 7.6 |
| runtime in java | Java | 3.4 |
| ocamntal | OCaml | 2.3 |
| **lmn-alpha** | **OCaml** | **0.9** |

*1. src/ 内のレキサ・パーザジェネレータにより生成されたファイルを除いた LOC

*2. src/compiler/ 内のレキサ・パーザジェネレータにより生成されたファイルを除いた LOC

# lmntal-compiler の問題点



```
compiler/
 Compactor.java
 CompileException.java
 Grouping.java
 GuardCompiler.java
 HeadCompiler.java
 LHSCompiler.java
 Module.java
 Optimizer.java
 RuleCompiler.java
 RulesetCompiler.java
 parser/
 structure/
 util/
```

**モジュール化という概念を感じさせない** 構成

- 意味解析器・命令列生成器…とかが
  （実装上は別れているけど）ごちゃまぜ

RuleCompiler.java は **1.5 KLOC**

- モジュール化できていないので
  トップレベルが肥大化

# Imntal-compiler のバグ



```
a(X, X).
a(X, Y), a(Y, X) :- .
```

```
spec            [1, 5]
findatom        [1, 0, 'a'_2]
deref           [3, 1, 1, 0]
deref           [2, 1, 0, 1]
func            [2, 'a'_2]
eqatom          [3, 2]
commit          ["_aXYa", 0]
```

循環グラフに健全でなくパターンマッチする
中間命令を出力してしまう可能性がある

deref 時の neqatom 命令の出し忘れ

# Imn-alpha のコンパイラ概要



```
compiler/
 compiler.ml
 parser/
 corelang/
 analyzer/
 generator/
 test/
```

各フェーズ毎に **モジュール化**

- 各モジュールは **約 50 ~ 170 LOC**
- エントリポイントは **実質一行**
  - 各フェーズの関数を合成しているだけ
- **各モジュール毎にテスト** を用意

**中間データを取り出して活用可能**

- パージング・略記法の解消などのみでも OK

1. 導入方法と実行例
2. 既存のツールとの比較
3. **実装の概要**
4. 今後の課題
5. まとめ

# ocamldoc によるドキュメント



https://sano-jin.github.io/lmn-alpha/ocamldoc/lmn

# コンパイラ概要



## 546 LOC

- lmntal-compiler の 1/10 以下

## 100% 純粋（破壊的代入はなし）

- fold_left_map で状態の更新をしながら map している
- コンビネータの嵐（かなり関数型スタイル）

# コンパイラ構成



### parse: 169 LOC

- 字句解析・構文解析を行う

### corelang: 139 LOC

- 糖衣構文の解消・リンクのチェック

### analyzer: 42 LOC

- 意味解析：ポート情報の付加

### generator: 169 LOC

- 中間命令列 を生成する

# 中間命令列ドキュメント



https://sano-jin.github.io/lmn-alpha/ocamldoc/lmn/Generator　/Instruction

# コンパイラが出力するコード例の抜粋



```
append_cons @@
    alloc [5]
--head:
    PeakAtom     [ 0 'append'_3 ]
    DerefAtom    [ 1 0 0 2 ]
    CheckFunctor [ 1 '.'_3 ]
--body:
    PushAtom     [ 3 '.'_3 ]
    PushAtom     [ 4 'append'_3 ]
    ReLink       [ 3 0 1 0 ]
    SetLink      [ 3 1 4 2 ]
    ReLink       [ 3 2 0 2 ]
    ReLink       [ 4 0 1 1 ]
    ReLink       [ 4 1 0 1 ]
    SetLink      [ 4 2 3 1 ]
    FreeAtom     [ 1 ]
    FreeAtom     [ 0 ]

append_nil @@
    alloc [3]
--head:
    PeakAtom     [ 0 'append'_3 ]
    DerefAtom    [ 1 0 0 0 ]
    CheckFunctor [ 1 '[]'_1 ]
--body:
    Connect      [ 0 2 0 1 ]
    FreeAtom     [ 1 ]
    FreeAtom     [ 0 ]
```

```
$./run example/append_dot.lmn -c
```

-c (compile only) オプションは,
中間命令列を出力する

# 実行時処理系の概要



## 280 LOC

- ocamntal の 1/8 以下

## 破壊的代入を行っている

- （仮想マシンは）手続き型のスタイルで実装した

# 実行時処理系構成



**vm: 137 LOC**

- 1 step 簡約する
- アトムリストを「dump」する機能をもつ

**pretty: 109 LOC**

- dump されたアトムリストを pretty print する

# アトムリストの「dump」



まずは，
アトムに id を振って，リンクはその id と引数番号の組にした
**中間データ構造を「dump」する**

→ Pretty print が他のアプリケーションからも使えるように

（ただし現状 dump されたデータをパーズする機能はない）

# Pretty print



アトムリストを

1. まずファンクタの種類と arity でソート
2. Topological sort を行い,
   できるだけ木の子よりも親をリストの先頭側にする
3. リストを木構造のリスト（＝森）へ変換する
4. 木構造を文字列に整形する（抽象構文の pretty print）
5. 文字列でまたソートする
6. 最後にリストをドットで繋げる

# 解釈実行の例



```
$./run example/append_dot.lmn -t
0: ret(append([a, b], [c]))
----> append_cons
1: ret([a | append([b], [c])])
----> append_cons
2: ret([a, b | append([], [c])])
----> append_nil
3: ret([a, b, c])
Final state:
ret([a, b, c])
```

-t (Trace) オプションは,
実行途中結果をトレースする

本物の実行結果に
色をつけ加えている

# Util モジュール：118 LOC



```
let (<.) f g = fun x -> f (g x)

let compile =
    gen_ic <. sem_graph_of_process <. corelang_of_ast <. parse
```

OCaml には合成関数が標準で用意されていない

→ 自前で定義できる（楽しい）

# 実装に関する感想



簡潔で構造的なのは,
単に関数型言語で実装したから

関数型のスタイルで実装するすると自動的に
手続き型やオブジェクト指向の 1/5 以下になる

lmn-alpha は関数型での実装にしては，むしろ若干？冗長かも
↔ 簡単な関数型言語の評価器なら 100 行あれば書ける

# ただし，



実行性能は，．．．悪い．

slim と 10 倍以上の性能差がある

- そもそもオーダーが悪い
- 恐らくはアトムの削除で
  アトムリストを全探索（filter）しているのが問題

1. 導入方法と実行例
2. 既存のツールとの比較
3. 実装の概要
4. **今後の課題**
5. まとめ

# 今後の課題（本当にやるかどうかは別）




1. 対応する構文を増やす（算術式など）
2. slim 対応の中間命令列を出力する
3. --use-swaplink と同等（以上）の命令を出力する
4. データアトム，ガードの実装
5. （素朴な）CSLMNtal 対応
6. 性能↑
7. 膜対応
8. 非決定実行対応

1. 導入方法と実行例
2. 既存のツールとの比較
3. 実装の概要
4. 今後の課題
5. **まとめ**

# まとめ



多くの人（> 1）が関わるプロジェクトなので
**コードの肥大化は仕方がない**

- 機能の追加をしてくれる人は少ないけどいても
- 機能の削除・全体の設計の見直しを好んでする人はまずいない

ただし，言語の（基礎理論的な）研究をするなら
**ミニマニズムに心惹かれるのも必然**だよね？

# 参考文献・コード



1. https://github.com/sano-jin/lmn-alpha
2. https://github.com/lmntal/lmntal-compiler
3. https://github.com/lmntal/slim