Algebraic Stepper for Simple Modules

浅井 健一、秋山 雛乃

お茶の水女子大学

PEPM 2025 （PPL 2025 用短縮日本語版）

デモページ：http://pllab.is.ocha.ac.jp/~asai/Stepper/demo/

ステッパとは

プログラムの実行の途中経過を全て表示するツール
（数式の変形や small-step 意味論のように）

$$
\begin{aligned}
& 4 + 5 \times (3 - 1) \\
= \; & 4 + 5 \times 2 \\
= \; & 4 + 10 \\
= \; & 14
\end{aligned}
$$

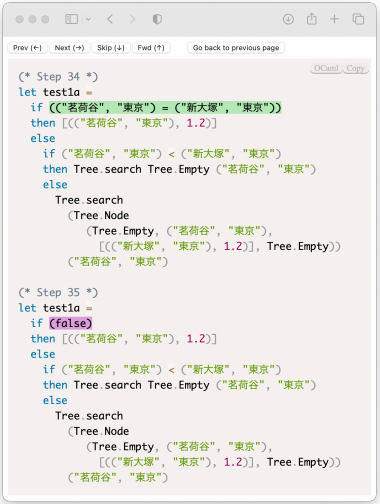
fac 5 $\to^*$ 5 * fac 4 $\to^*$ 5 * (4 * fac 3) $\to^*$ 5 * (4 * (3 * fac 2)) $\to^* \cdots$

$$(\lambda x.\, \lambda y.\, x + y)\, 3\, 4 \to (\lambda y.\, 3 + y)\, 4 \to 3 + 4 \to 7$$

OCaml のステッパ

お茶大では OCaml のステッパを作って関数型言語の授業で使っている。

- 基本的な構文（再帰、レコード、リスト、例外、文字列出力、セルなど）をサポート。
- Emacs で動く（VS code もサポートしようと努力中）。デモページあり。
- 授業で扱う構文のうち、サポートできていないのはモジュールのみ。



ステッパ ≠ Small-Step 意味論

変数の遅延代入

ステッパでは、変数は 1 ステップ使ってその値に変化する。
（変数の定義時に代入は行わない。）

プログラム：

```
let a = 10
let f x = a + x
let _ = f 100
```

ステップ実行：

```
    f 100
→   a + 100
→   10 + 100
→   110
```

Small-step 意味論：

```
let a = 10
let f x = 10 + x
let _ = (\x.10 + x) 100
    (\x.10 + x) 100
→   10 + 100
→   110
```

変数の値の管理：attribute を使う

変数には **レベル** と **値** が付与される。

ユーザ視点：

```
let a = 10
let f x = a + x
let _ = f 100
```

```
    f 100
→   a + 100
→   10 + 100
→   110
```

ステッパの実装内部：

```
let a = 10
let f x = a [@0] [@10] + x
let _ = f [@0] [@\x.a[@0][@10] + x] 100
```

```
    f [@0] [@\x.a[@0][@10] + x] 100
→   a [@0] [@10] + 100
→   10 + 100
→   110
```

(attribute **はユーザからは見えない**)

名前の衝突（が起きているように見える）

ユーザ視点：

```
let a = 10
let f x = a + x
let a = 20
let _ = f 100
```

```
    f 100
→   a + 100
→   10 + 100
→   110
```

ステッパの実装内部：

```
let a = 10
let f x = a [@0] [@10] + x
let a = 20
let _ = f [@0] [@\x.a[@0][@10] + x] 100
```

```
    f [@0] [@\x.a[@0][@10] + x] 100
→   a [@0] [@10] + 100
→   10 + 100
→   110
```

OCaml のモジュール

```
let a = 10
let f x = a + x
let _ = f 100

module X = struct
  let a = 20
  let g x = f x + a
  let _ = g 200
end

let _ = X.g 300
```

プログラム：モジュールの木構造

- モジュールには、型宣言、変数宣言、モジュール宣言を書ける。
- それらが出てきた順に１度だけ実行される。

変数の参照の仕方：

- 自分を含む親モジュールで宣言された変数は直接、参照可能。
- 子モジュールで宣言された変数にはモジュール名を付ける。

変数の値をモジュール内に伝播

```
let a = 10
let f x = a [@0] [@10] + x
let _ = f 100

module X = struct
  let a = 20
  let g x = f x + a
  let _ = g 200
end
```

- 変数の値は以降のプログラム中に伝播する。

変数の値をモジュール内に伝播

```
let a = 10
let f x = a [@0] [@10] + x
let _ = f 100

module X = struct
  let a = 20
  let g x = f x + a
  let _ = g 200
end
```

- 変数の値は以降のプログラム中に伝播する。
- 同じ変数名にぶつかったら、そこで終了。

変数の値をモジュール内に伝播

```
let a = 10
let f x = a [@0] [@10] + x
let _ = f [@0] [@\x. a[@0][@10] + x] 100

module X = struct
  let a = 20
  let g x = f [@1] [@\x. a[@1][@10] + x] x + a
  let _ = g 200
end
```

- 関数の値も以降のプログラム中に伝播する。
- モジュールに入るとレベルが 1 増える。

変数の値をモジュール内に伝播

```
let a = 10
let f x = a [@0] [@10] + x
let _ = f [@0] [@\x. a[@0][@10] + x] 100

module X = struct
  let a = 20
  let g x = f [@1] [@\x. a[@1][@10] + x] x + a [@0] [@20]
  let _ = g 200
end
```

- attribute の中の変数には伝播しない。

変数の値をモジュール内に伝播

```
let a = 10
let f x = a [@0] [@10] + x
let _ = f [@0] [@\x. a[@0][@10] + x] 100

module X = struct
  let a = 20
  let g x = f [@1] [@\x. a[@1][@10] + x] x + a [@0] [@20]
  let _ = g [@0] [@\x. f[@1][@...] x + a[@0][@20]] 200
end
```

$\rightarrow$ `f[@1][@\x. a[@1][@10] + x] 200 + a[@0][@20]`
$\rightarrow$ `f[@1][@\x. a[@1][@10] + x] 200 + 20`
$\rightarrow$ `(a[@1][@10] + 200) + 20`
$\rightarrow$ `(10 + 200) + 20`

モジュールの値を以降のプログラムに伝播

```
let a = 10
let f x = a [@0] [@10] + x

module X = struct
  let a = 20
  let g x = f [@1] [@\x. a[@1][@10] + x] x + a [@0] [@20]
end

let _ = X.g 300
```

- モジュールの実行が終わると、その情報が以降のプログラムに伝播する。

モジュールの値を以降のプログラムに伝播

```
let a = 10
let f x = a [@0] [@10] + x

module X = struct
  let a = 20
  let g x = f [@1] [@\x. a[@1][@10] + x] x + a [@0] [@20]
end

let _ = X.g [@0] [@\x. f[@0][@\x. a[@0][@10] + x] x
                    + X.a[@0][@20]] 300
```

- レベルは 1 減る。
- レベルがすでに 0 なら、モジュール名を付ける。

定式化：できます

論文では、ステッパ意味論、small-step 意味論の両方について

- 構文
- 簡約規則（式用）
- 実行規則（モジュール用）

を定義した後、以下の定理を証明

Theorem

ステッパで e_1 が e_2 に簡約されるなら、small-step 意味論でも e_1 は e_2 に（1ステップまたは0ステップで）簡約される。

注：簡約は対応するが、変数名が正しいことは言っていない。

現状とまとめ

- OCaml 4.14.2 にて実装。
- お茶大の関数型言語の授業で使用。

ステッパ $\neq$ small-step 意味論

変数の遅延代入があるので。

今後の課題

- 代数的エフェクトとハンドラをサポート？
- ファンクタ？
- Signature sealing はステッパとは相性が悪そう。