タイトル

名前　(指導教員：浅井 健一)

1　はじめに

ステッパとはプログラミング教育やデバッグのために使うツールであり、プログラムが代数的に書き換わる様子を出力することで実行過程を見せるものである。

これまでに Racket [?] や OCaml [?] のプログラミング教育用に制限した構文などを対象にステッパが作られてきたが、shift/reset [?] や algebraic effects and handlers [?] (以下、algebraic effects) といった、継続を明示的に扱うための機能を含む言語のステッパは作られていない。継続を扱うプログラムの挙動を理解するのは困難なので、そういった言語に対応したステッパを作ることが本研究の目的である。

ステッパは簡約のたびにその時点でのプログラム全体を出力するインタプリタなので、実行している部分式のコンテキスト (周りの式) の情報が常に必要になる。継続を扱うような複雑な機能を持つ言語を対象にしたステッパでは、コンテキストがどのような構造をしているかが自明でない。そこで、通常のインタプリタ関数をプログラム変換することで機械的にコンテキストの情報を保持させ、ステッパを実装する方法を示す。実際に型無し λ 計算 [?] と algebraic effects から成る言語を対象にしてステッパを実装する過程を説明する。

2　ステッパとは

ステッパ

3　対象言語とインタプリタ

本稿では、以下の定義で表される言語についてステッパを実装する方法を示す。

```
(* 値 *)
type v =
    Var of string      (* x *)
  | Num of int         (* n *)
  | Fun of string * e  (* fun x -> e *)
  | Handler of h       (* ハンドラ *)
  | Cont of string * (cont_in -> cont_in)
(* ハンドラ *)
and h =
  {return : (string * e) option;
   ops : (string * string * string * e) list}
(* 式 *)
and e = Val of v          (* v *)
      | App of e * e      (* e e *)
      | Op of string * e  (* op e *)
      | With of e * e (* with e handle e *)
(* 継続 *)
and cont_in = v -> a
```

4　ラムダ式

macro.tex に書いてあるような定義をあらかじめしておいて、それを使うのが良いです。

$$\text{型}\quad t \;:=\; \mathsf{int} \mid t \to t$$
$$\text{項}\quad e \;:=\; x \mid \lambda x.\, e \mid e\, e \mid \mathcal{S}k.\, e \mid \langle e \rangle$$

5　箇条書きの書き方など

箇条書き

- ひとつめ
- ふたつめ
- みっつめ

番号付き

1. ひとつめ
2. ふたつめ
3. みっつめ

見出し付き

継続　その後の計算をさす。

限定継続　継続のうち、その範囲が限定されているもの。

6　コード

コードを直接、書くには verbatim 環境が簡単です。

```
let rec fac n =
  if n = 0 then 1 else n * fac (n - 1)
```

verbatim 環境内で tex のコマンドを使いたいときは、alltt を usepackage して使います。上のコードは、どうも前後の文と間がきつすぎると思うときは、quote 環境に入れるというのはひとつの手です。

```
let rec fib n =
  if n < 2
  then n
  else fib (n - 1) + fib (n - 2)
```

これらの環境はタイプライタフォントなので、横幅をとりすぎる傾向にあります。なれてきたら、よりきれいにコードを書く方法を習得するのが良いかも知れません。

7　参考文献

bibtex を使うのが良いでしょう。paper.bib に型デバッガ [6] 関係の文献と限定継続 [1, 2] 関係の文献を入れておきました。また、本の例としてアルゴリズミックデバッギング [4] も入れました。

8　定理

定義 1 (CPS 変換 [3]) 項 e の CPS 変換 $[\![e]\!]$ は以下のように定義される。（中身は省略。）

命題 2 e に型がつくなら、その部分式にも型がつく。

補題 3 (代入補題 [5]) $x : t_1 \vdash e : t$ かつ $\vdash v : t_1$ なら $\vdash e[v/x] : t$ が成り立つ。

定理 4 e に型がついたら、e の実行中に型エラーは起きない。

9　証明木

証明木の例です。judgement もマクロとして定義するのが良いです。

$$\frac{\Gamma(x) = t}{\Gamma \vdash x : t}\ (\mathsf{TVar}) \qquad \frac{\Gamma, x : t_1 \vdash e : t_2}{\Gamma \vdash \lambda x.\, e : t_1 \to t_2}\ (\mathsf{TLam})$$

$$\frac{\Gamma \vdash e_1 : t_2 \to t_1 \quad \Gamma \vdash e_2 : t_2}{\Gamma \vdash e_1\, e_2 : t_1}\ (\mathsf{TApp})$$

10　長さ稼ぎ

あいうえおかきくけこさしすせそたちつてとなにぬねのはひふへほまみむめもやゆよらりるれろわをんABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstu

v
w
x
y
z
0
1
2
3
4
5
6
7
8
9

11　まとめ

最後に、まとめと今後の課題などを書きます。このサンプルは 1 ページで終わっていますが、必ず 2 ページを埋めます。（2 ページはすぐ埋まります。むしろすぐ足りなくなります。紙面が足りない場合、下のように参考文献は多少、小さくしても構いません。）

参考文献

[1] O. Danvy and A. Filinski. A Functional Abstraction of Typed Contexts. Technical Report 89/12, DIKU, University of Copenhagen, July 1989.

[2] O. Danvy and A. Filinski. Abstracting Control. In *Proc. 1990 ACM Conference on Lisp and Functional Programming*, pp. 151–160, 1990.

[3] G. D. Plotkin. Call-by-Name, Call-by-Value, and the λ-Calculus. *Theoretical Computer Science*, Vol. 1, pp. 125–159, 1975.

[4] E. Y. Shapiro. *Algorithmic Program Debugging*. Cambridge: MIT Press, 1983.

[5] A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, Vol. 115, No. 1, pp. 38–94, 1994.

[6] 対馬かなえ, 浅井健一. コンパイラの型推論を利用した型デバッグの手法の提案. コンピュータソフトウェア, Vol. 30, No. 1, pp. 180–186, 2013.