探索アルゴリズムのための非決定性オペレータの実装

金子ちひろ　(指導教員：浅井健一)

1　はじめに

非決定性とは，次の動作が一意に定まらないような計算のモデルであり，複数の選択肢の中から正しい動作を推測することにより，解を求めることができる．非決定性オペレータ choice と fail を用いると，このような非決定性のモデルを，直接プログラムとして記述することができる．オペレータはバックトラックによる解の探索を行い，字句解析や構文解析のほか，多くの探索問題を扱うことができる．

しかし，一般に探索問題では，目的とする解や効率のために，適切な探索アルゴリズムを選ぶことが重要となる．そこで，本研究ではさまざまな探索を実行する非決定性オペレータを，限定継続命令 shift/reset を用いて実装した．

2　非決定性オペレータの動作

オペレータは，非決定的な選択をする choice と，選択の失敗を意味する fail からなる．以下では，choice は選択肢のリストを受け取り要素一つを返す関数，fail は引数を持たない関数とする．

これらの関数を用いた非決定性のプログラムは，choice の選択によって，解または fail のどちらかに到達するように記述する．例えば，二つのリストに共通する要素を返すプログラムは，非決定性オペレータを使って次のように記述できる．

```
let f lst1 lst2 =
  let a = choice lst1 in
  let b = choice lst2 in
  if a = b then a
           else fail ()
```

このプログラムでは，二つのリストから要素を choice によって選び，それらが等しければその要素（解）を返し，そうでなければ fail とする．

このようなプログラムに対して，オペレータは自動的にバックトラックを行うことで，fail に到達しないような解を探索する．すなわち，choice は選択肢の中から適当な一つを実行し，それが fail に到達した場合には別の選択肢を選んで再実行する．これを繰り返すことで，最終的に解を出力することができる．

3　継続による実装

非決定性オペレータが行うバックトラックを，継続を使って実装する方法が知られている [1]．ここで継続とは，計算のある時点における残りの計算を表す概念である．非決定性オペレータは，選択肢を実行する際に現在の継続を保存しておき，その選択が失敗した場合に，保存した継続を別の選択肢で実行することでバックトラックを実現する．

継続を扱うには，プログラムを CPS（継続渡し形式）で書くなど，いくつかの方法がある．本研究では，限定継続命令である shift/reset[2] を用いた．shift/reset は継続を切り取る shift と，継続の範囲を限定する reset からなる．また，本研究では shift/reset を扱える言語環境として，Caml Light の拡張 [3] を用いた．

4　探索アルゴリズム

本研究では，通常の非決定性オペレータが行う探索として，深さ優先探索，幅優先探索，最良優先探索の三つのアルゴリズムを実装した．また，特殊な非決定性オペレータとして，ゲーム木探索に特化した，アルファベータ法を行うものを実装した．

5　ライブラリ

実装した各アルゴリズムについて，非決定性プログラムの実行に必要な関数をまとめたライブラリを作成した．ライブラリの一覧を以下に示す．

Depth	深さ優先探索
Collect	深さ優先探索（すべての解）
Breadth	幅優先探索
Best	最良優先探索
Game	アルファベータ法

6　深さ優先，幅優先，最良優先

深さ優先，幅優先，最良優先のライブラリは，次の三つの関数で構成される．

`choice`	非決定的な選択をする
`fail`	選択の失敗を表す
`start`	探索の初期化をする

6.1　深さ優先（Depth, Collect）

深さ優先探索は，深い分岐点を優先して探索する最も基本的な手法であり，継続によるバックトラックで実装できる．

Depth のプログラムを 図 1 に示す．このプログラムは二つの継続を扱っており，`k` はバックトラックに必要な各分岐点の継続，`cont` はそれらを含む探索全体の継続である．ここでは，前者を shift，後者を CPS により扱っている．文献 [1] では，`k` は success continuation, `cont` は failure continuation と呼ばれる．

また，Depth の代わりに Collect を使うと，全ての解をリストにして返すことができる．これは，ライブラリが内部でリストを用意しておき，解を見つけるたびにそれを付け加えていくことで実現する．

6.2　幅優先（Breadth）

幅優先は，分岐の浅い点から順に探索する手法である．一般に，深さ優先と幅優先は，未探索点を保存するデータ構造をスタックとするかキューとするかによって切り替えることができる．Depth のプログラムにおいて，スタックに相当するものは `cont` であり，この `cont` をキューに変更することで幅優先を実現できる．

Breadth の実装では，効率上，キューの代わりにスタックを二つ用いることで幅優先を実現した．

```
let choice lst =      (* 非決定的に選択する *)
  shift (fun k -> fun cont ->
    let rec loop lst =
      match lst with
          [] -> cont ()
        | a :: rest -> k a (fun () -> loop rest)
    in loop lst);;
let fail () =         (* 失敗，バックトラック *)
  shift (fun k -> fun cont -> cont ());;
let start f =         (* 初期化など *)
  (reset (fun () -> let r = f () in fun _ -> r))
    (fun () -> raise Not_found);;
```

図 1: 深さ優先探索（Depth）

6.3 最良優先（Best）

最良優先探索とは，各分岐点で解までの推定コストを求め，コストの低い分岐点から順に探索する手法である．応用に，A*アルゴリズムなどがある．

Best ではこの動作を実現するため，cont のデータ構造を，各分岐点における継続とコストの組のリストとした．また，この際，コストは choice の第二引数として受け取ってくるものとした．このリストをライブラリが内部でソートすることで，コストの低いものから順に継続を実行する．

7 アルファベータ法

7.1 ミニマックス法とアルファベータ法

ゲーム探索とは，先手と後手が交互に手を打つようなゲームにおいて，ゲームの展開をあらかじめ先読みすることで，有利となる手を求める手法である．

ミニマックス法では，まず先読みする手数 n を定め，n 手先の各局面の評価値を求める．この値に対して，$(n-1)$ 手以下の局面では，先手は最大，後手は最小をとるように次の手を選ぶことで，双方の利害を考慮したゲームの展開を実現する．アルファベータ法は，ミニマックス法の最大，最小を交互にとる性質に関して，枝刈りを導入することで効率化したものである．

7.2 実装（Game）

アルファベータ法を実現するため，Game ライブラリでは，以下の関数を用意した．

choice_max	評価値が最大となるものを選ぶ
choice_min	評価値が最小となるものを選ぶ
return	解に評価値を与えて返す
start	探索の初期化をする

ここで，choice_max と choice_min が非決定性のオペレータである．また，fail に相当するものはなく，代わりにすべての結果を return で返す必要がある．これらを用いた非決定性プログラムは，先手を choice_max，後手を choice_min によって選び，n 手先で return に解と評価値を与えるように記述する．

Game のプログラムを 図 2 に示す．各オペレータは継続を用いて選択肢を順に探索し，評価値が最大または最小となるものを求める（ミニマックス）．ただし，途中で枝刈りの条件を満たした場合には，探索を中断して適当な値を返す（アルファベータ）．

```
let return sol v =      (* 評価値を与える *)
  shift (fun k -> fun _ -> (v, Some sol));;
let getv (v, _) = v;;   (* 補助関数 *)
let choice_max lst =    (* 最大を選択する *)
  shift (fun k -> fun beta ->
  let rec loop lst max = match lst with
    [] -> max
  | first :: rest ->
    let a = k first (getv max) in
      if (getv a) > (getv max)
      then if (getv a) >= beta then a (* cut *)
      else loop rest a
      else loop rest max
  in loop lst (min_int, None));;
let choice_min lst =    (* 最小を選択する *)
  shift(fun k -> fun alpha ->
  let rec loop lst min = match lst with
    [] -> min
  | first :: rest ->
    let a = k first (getv min) in
      if (getv a) < (getv min)
      then if (getv a) <= alpha then a (* cut *)
      else loop rest a
      else loop rest min
  in loop lst (max_int, None));;
let start f =           (* 初期化など *)
  let top = (reset(fun () -> let r = f () in
              fun _ -> (0, Some r))) max_int
  in match top with
      (_, Some r) -> r
    | (_, None) -> raise Not_found;;
```

図 2: アルファベータ法（Game）

8 まとめと今後の課題

探索問題を非決定的に記述するためのオペレータを提案，実装した．オペレータが探索アルゴリズムを分離することにより，非決定性プログラムには問題の本質である，作用素の適用や，解の判定のみを記述することができる．また，これによりアルゴリズムの変更なども容易となる．

一方で，探索問題では，実際の探索順序に従う処理を記述したい場合がある．このような処理として，例えば，ある変数に探索した状態をすべて記録していき，同じ状態を探索しないといった処理があげられる．今後はこのような処理を自然に記述する方法や，その他の処理の組み込みについて考察する．

参考文献

[1] Harold Abelson and Gerald Jay Sussman with Julie Sussman. *Structure and Interpretation of Computer Programs*. Mit Press, Cambridge, 1996.

[2] Olivier Danvy and Andrzej Filinski. Abstracting Control. In *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pp. 151–160, June 1990.

[3] 増子萌, 浅井健一. shift/reset による Caml Light の拡張に向けて. 第 12 回プログラミングおよびプログラミング言語ワークショップ, pp. 115–129, 2010.