「プログラミング言語」
SICP 第4章
～超言語的抽象～
その6

五十嵐 淳

igarashi@kuis.kyoto-u.ac.jp

京都大学

July 18, 2012

今日のメニュー

- 4.3: Variations on a Scheme – Non-deterministic Computing
  - 4.3.1: **特殊形式 amb と探索**
  - 4.3.2: **非決定的プログラムの例**
  - 4.3.3: **amb インタプリタの実装**

Scheme 変奏曲 – 非決定的計算

非決定的計算 $\stackrel{\text{def}}{=}$ 結果が複数あるような計算

- **計算過程の多世界 (パラレルワールド) 解釈？**
- “generate and test” **方式で (探索) 問題を解くのに向いている**

$\Rightarrow$ **リスト，ストリームを使って散々 (？) やったこと**

例題:

与えられたふたつの自然数の集合から，和が素数になる要素の組を列挙する非決定的プログラム:

```
(define (prime-sum-pair list1 list2)
  (let ((a (an-element-of list1))
        (b (an-element-of list2)))
    (require (prime? (+ a b)))
    (list a b)))
```

- an-element-of: **リストから要素をひとつ非決定的に選択(し，計算過程を分岐させる)**
- require: **引数式が真である時のみ先に進む**

```
(prime-sum-pair '(2 3 6) '(4 5))
```

を実行すると

a	b	(prime? (+ a b))	**結果**
2	4	#f	—
2	5	#t	(2 5)
3	4	#t	(3 4)
3	5	#f	—
6	4	#f	—
6	5	#t	(6 5)

3 **通りのうちからどれかが結果になる**

4.3.1: 特殊形式 amb と探索

特殊形式 amb

$$(\texttt{amb}\ e_1\ \ldots\ e_n)$$

- e_i を非決定的に選択し評価する
- 結果は n 通り
- $n = 0$ でもよい!
  - 計算結果がない場合 (探索の失敗など) を表す

require も an-element-of も amb で書ける

```
(define (require p)
  (if (not p) (amb)))

(define (an-element-of items)
  (require (not (null? items)))
  (amb (car items)
       (an-element-of (cdr items)))))

;; 選択肢は無限にあってもよい!
(define (an-integer-starting-from n)
  (amb n (an-integer-starting-from (+ n 1))))
```

amb による系統的探索

- ひとつの (この実装では先頭の) 式を選んで計算を続ける
- 残りの計算が終了したらそれを全体の結果とする
- 残りの計算が途中で ((amb) を実行して) 失敗したら，次の式を選んで再挑戦
- 失敗が続いて選択肢が尽きたら全体を失敗扱い

amb インタプリタの REPL

- 式を入力するとひとつめの答えが出る
- 直後に `try-again` と入力すると次の答えが出る
- 答えが尽きたら知らせてくれる

4.3.2: 非決定的プログラムの例

- 論理パズル
- 自然言語の構文解析

論理パズル

[Dinesman 1968] より

- Baker, Cooper, Fletcher, Miller, Smith **が五階建ての建物の，それぞれ別の階に住んでいます．**
- Baker **は最上階には住んでいません．**
- Cooper **は最下階には住んでいません**
- Fletcher **は最上階にも最下階にも住んでいません．**
- Miller **は** Cooper **よりも高い階に住んでいます．**
- Smith **は** Fletcher **の隣りの階には住んでいません．**
- Fletcher **は** Cooper **の隣りの階には住んでいません．**
- **それぞれ何階に住んでいるでしょう？**

```
(define (multiple-dwelling)
(let ((baker (amb 1 2 3 4 5)) ;; 解候補列挙
      ... (smith (amb 1 2 3 4 5)))
  ;; 条件のチェック
   (require
     (distinct? (list baker ... smith)))
   (require (not (= baker 5))
   ...
   (require (not (= (abs (- fletcher cooper))
                    1)))
   ;; 全てのチェックを通過したらそれは解
   (list (list 'baker baker)
         ...
         (list 'smith smith)))))
```

自然言語の構文解析

- 構文解析: 単語列から文法的構造を取りだす
- 文法の例:

$$
\begin{aligned}
\langle \text{名詞} \rangle &::= \text{student} \mid \text{professor} \mid \text{cat} \mid \text{class} \\
\langle \text{動詞} \rangle &::= \text{studies} \mid \text{lectures} \mid \text{eats} \mid \text{sleeps} \\
\langle \text{冠詞} \rangle &::= \text{the} \mid \text{a} \\
\langle \text{文} \rangle &::= \langle \text{名詞句} \rangle \langle \text{動詞} \rangle \\
\langle \text{名詞句} \rangle &::= \langle \text{冠詞} \rangle \langle \text{名詞} \rangle
\end{aligned}
$$

- 構文木による文法的構造の表現

単語の構文解析

```
(define articles '(article the a))

;; *unparsed* -- まだ解析していない単語列
(define (parse-word word-list)
  (require (not (null? *unparsed*)))
  (require (memq (car *unparsed*)
                 (cdr word-list)))
  (let ((found-word (car *unparsed*)))
    (set! *unparsed* (cdr *unparsed*))
    (list (car word-list) found-word)))
```

文と名詞句の構文解析

```
(define (parse-sentence)
  (list 'sentence
    (parse-noun-phrase) (parse-word verbs)))

(define (parse-noun-phrase)
  (list 'noun-phrase
    (parse-word articles) (parse-word nouns)))

(define (parse input)
  (set! *unparsed* input)
  (let ((sent (parse-sentence)))
    (require (null? *unparsed*))
    sent))
```

より複雑な文法へ

- 例の文法には単語レベル以外選択肢がない
- 選択肢のある文法

$$
\begin{aligned}
\langle \text{名詞} \rangle &::= \ldots \\
\langle \text{前置詞} \rangle &::= \text{for} \mid \text{to} \mid \text{in} \mid \text{by} \mid \text{with} \\
\langle \text{文} \rangle &::= \langle \text{名詞句} \rangle \langle \text{動詞句} \rangle \\
\langle \text{前置詞句} \rangle &::= \langle \text{前置詞} \rangle \langle \text{名詞句} \rangle \\
\langle \text{名詞句} \rangle &::= \langle \text{冠詞} \rangle \langle \text{名詞} \rangle \\
&\quad \mid \langle \text{名詞句} \rangle \langle \text{前置詞句} \rangle \\
\langle \text{動詞句} \rangle &::= \langle \text{動詞} \rangle \mid \langle \text{動詞句} \rangle \langle \text{前置詞句} \rangle
\end{aligned}
$$

```
(define prepositions '(prep for to in by with))

(define (parse-prepositional-phrase)
  (list 'prep-phrase
        (parse-word prepositions)
        (parse-noun-phrase)))

(define (parse-sentence)
  (list 'sentence
        (parse-noun-phrase)
        (parse-verb-phrase)))
```

```
(define (parse-verb-phrase)
  (define (maybe-extend verb-phrase)
    (amb
      verb-phrase
      (maybe-extend
        (list 'verb-phrase
              verb-phrase
              (parse-prepositional-phrase)))))
  (maybe-extend (parse-word verbs)))

;; parse-noun-phrase も同様
```

例

- The student with the cat sleeps in the class.
- The professor lectures to the student with the cat.

宿題： 7/25 午前8時 締切

- Exercise 4.35, 4.42, 4.45
- **レポートには**
  - **考え方の説明**
  - **プログラムリストと考え方の対応**
  - **実行例**

  を示すこと
- **レポート** (pdf) **とプログラムファイルを提出**
- **友達に教えてもらったら、その人の名前を明記**
- web **は出典を明記** (**「同じ」回答は減点**)

Ex. 4.35

Write a procedure an-integer-between that returns an integer between two given bounds. This can be used to implement a procedure that finds Pythagorean triples, i.e., triples of integers (i, j, k) between the given bounds such that $i < j$ and $i^2 + j^2 = k^2$, as follows:

```
(define (a-pythagorean-triple-between lo hi)
  (let ((i (an-integer-between lo hi)))
    (let ((j (an-integer-between i hi)))
      (let ((k (an-integer-between j hi)))
        (require (= (+ (* i i) (* j j))
                    (* k k)))
        (list i j k)))))
```

Ex. 4.42: 嘘つきパズル

Five schoolgirls sat for an examination. Their parents – so they thought – showed an undue degree of interest in the result. They therefore agreed that, in writing home about the examination, each girl should make one true statement and one untrue one. The following are the relevant passages from their letters:

- Betty: "Kitty was second in the examination. I was only third."
- Ethel: "You'll be glad to hear that I was on top. Joan was second."
- Joan: "I was third, and poor old Ethel was bottom."
- Kitty: "I came out second. Mary was only fourth."
- Mary: "I was fourth. Top place was taken by Betty."

What was the order in which the five girls were placed?

Ex. 4.45

With the grammar given above, the following sentence can be parsed in five different ways:

> *"The professor lectures to the student in the class with the cat."*

Give the five parses and explain the differences in shades of meaning among them.

今日のおまけ

- 4.3: Variations on a Scheme – Non-deterministic Computing
  - 4.3.1: **特殊形式 `amb` と探索**
  - 4.3.2: **非決定的プログラムの例**
  - 4.3.3: **`amb` インタプリタの実装**

  への前奏曲

amb と catch/throw

- **類似点**
  - **どちらも実行の「ジャンプ」を引き起こす**
  - amb **は** catch **と** throw **両方の役割をする**
- **相違点**
  - catch **が働く** (catch **する**) **有効範囲は本体式の実行中**
  - amb **の有効範囲は残りの計算全部**

amb は catch と throw 両方の役割をする

- (throw ...) ∼ (amb)
- (amb e1 e2): e1 の評価中に (amb) が実行されると，その評価が中断し e2 の評価から再開する

  (amb (+ 2 (amb) 3) 4)

  ⇒ 4

  - (amb) が実行されなければ，e1 の値がそのまま全体の値

catch と amb の有効範囲比較

```
(let ((x (catch 'a 2)))
  (if (even? x) (throw 'a true) x))
⇒ (uncaught exception)

(let ((x (amb 2 3)))
  (if (even? x) (amb) x))
⇒ 3
```

(amb ...) を抜けた後の (amb) の実行は，実行の巻き戻し (backtrack という) を起こす!

継続の言葉でいうと…

- `throw` は継続の一部を飛ばす $\Rightarrow$ 「早送り」
- (`amb`) は「早送り」することもあれば，既に終了して TODO リストから消された作業を復活させて「巻き戻す」ことも
  - 常に「最後に `amb` に実行が突入した時点での継続」が実行される

継続渡しインタプリタによる amb の実装

アイデア: 二種類の継続を引数とする eval

- 式の評価がふつうに終了した時にその値を渡す継続
- 式の評価中に（amb）が実行された時に呼び出す継続
  - 「最後に amb に実行が突入した時点での継続」