プログラミングAIII

2025年度講義資料 (7)

新潟大学　工学部工学科　知能情報システムプログラム

青戸等人

目次

目次

case 式の例(1)

パターンマッチによる場合分けを，式の中の任意の場所でも用いることができる.
式中の場合分けにはcase 式を用いる．以下の関数は同じ動作をする.

```
fun moreThanTwoElems []   = false
  | moreThanTwoElems [x] = false
  | moreThanTwoElems  _   = true;

fun moreThanTwoElems' xs =
    case xs of
       []  => false
     | [x] => false
     | _   => true;
```

case 式(教6.3節)

case を用いた式

```
case expr of
    pat_1  =>  expr_1
  | pat_2  =>  expr_2
    ..........
  | pat_N  =>  expr_N
```

は，式 $expr$ の値をまず評価し，その値がどのパターンにマッチするが上から順番にみていく．そして，その値が最初にマッチするパターンがpat_iのとき， $expr_i$を評価した値を返す．

- 式の一部として，式が書ける場所なら，どの場所にも case 式を用いた式を書くことができる．
- 改行は可読性のために入れているだけで，入れなくても問題ない．

case 式の例(2)

```
fun isSum0or2 (x,y) =
  case x + y of 0 => true | 2 => true | _ => false;

fun appBothNonEmptyPairs [] = []
  | appBothNonEmptyPairs (zs::zss) =
    case zs of
       ([],_)  => appBothNonEmptyPairs zss
     | (_,[])  => appBothNonEmptyPairs zss
     | (xs,ys)  => (xs @ ys) :: appBothNonEmptyPairs zss;

fun foo (x,y) =
  x + (case x * y of 0 => 1 | _ => 0) + y;
```

実習課題 (1)

パターンマッチと case 式を併用して，以下の関数を定義せよ．
(前ページのappBothNonEmptyPairs関数を参考にせよ．)

1. リストのリストが与えられたときに，要素となるリストが空でなければ先頭要素を取り除く関数 dropHeads.

```
# dropHeads [[1,2],[3],[],[4,5,6]];
val it = [[2], [], [], [5, 6]] : int list list
```

2. リストのリストLが与えられたときに，長さが2以上のリストのみを要素として残す関数 takeMoreThanTwoElems.

$$[\ xs \mid xs \in L, |xs| \geq 2\ ]$$

```
# takeMoreThanTwoElems [[1,2],[3],[],[4,5,6]];
val it = [[1, 2], [4, 5, 6]] : int list list
```

実習課題 (1)

3. 整数のリスト$[x_0, x_1, \ldots, x_n]$に対して，リスト $[\sum_{i=0}^{n} x_i, \sum_{i=1}^{n} x_i, \ldots, \sum_{i=n}^{n} x_i]$を返す関数 subseqSums. (ヒント：再帰呼出しを case ... of の ... の部分に入れてみよ.)

```
# subseqSums [1,3,5,7];
val it = [16, 15, 12, 7] : int list
```

case 式の注意点 (1): パターンの変数による束縛

```
# fun test1 xs =
        case xs of
            []  => []
          | (x::xs) => xs;
val test1 = fn : ['a. 'a list -> 'a list]
# test1 [1,2,3];
val it = [2, 3] : int list
```

- パターンに変数が含まれる場合は，その変数の束縛が起きる．変数束縛のスコープは，そのパターンの右辺.
- 変数の定義の場合と同様に，同じ名前の変数がその前にあれば，前の変数束縛は隠されて見えなくなる.

case 式の注意点 (2)： case 式のネスト

case 式のネスト

- case 式のパターンは，パターンがネストしているときは，1番最後のcase式におけるパターンになる.
- 外側のcase 式のパターンになるようにするには，内側のcase 式の範囲を括弧で囲むことで明示する.

```
case expr1 of
   | pat1a => (case expr2 of
                 pat2a => ...
               | pat2b => ...)
   | pat1b => (case expr3 of
                 pat3a => ...
               | pat3b => ...)
   | pat1c => ...
```

1. 以下の関数で，false が返ってくる場合は，どのような場合か．未定義エラーになるのはどのような場合か．

```
fun test2 (x,y) =
        case x of
            0 => case y of
                     1 => true
                   | n => false;
```

2. 以下の関数定義はエラーになる．括弧を補って修正せよ．

```
fun test3 (x,y) =
        case x of
            0 => case y of
                     1 => true
                   | n => false
          | n => false;
```

目次

部分関数

- 通常の数学で扱う関数は，入力の値が定義域のどの値であっても関数によって移される先があるが，リストにおけるhdのように，場合によっては(空リストでない場合は)値を返すが，場合によっては(空リストの場合は)返す値がないような関数を考える方が自然な場合もある．(前者を全域関数，後者を部分関数とよんで区別する．)
- ここでは，そのような，場合によっては値があるが，場合によっては値がないような式を SMLで扱う方法として，オプション型を使う方法と例外を使う方法がある
- ここでは，オプション型について学ぶ．オプション型は，組型やリスト型と同じく，代表的な複合型である．

オプション型(教7.4節) (1)

```
# SOME "aa"; SOME #"a"; SOME (SOME 1);
val it = SOME "aa" : string option
val it = SOME #"a" : char option
val it = SOME (SOME 1) : int option option
# SOME;
val it = fn : ['a. 'a -> 'a option]
# NONE;
val it = NONE : ['a. 'a option]
```

- オプション型は，「NONE」という値と「SOME ...」という2種類の値をもつ．
- vの型が$'a$のとき，「SOME v」の型は「$'a$ option」．

オプション型(教7.4節) (2)

```
# isSome (SOME 3);
val it = true : bool
# isSome NONE;
val it = false : bool
# valOf (SOME 3);
val it = 3 : int
# SOME 3 = SOME (1+2); SOME 3 = SOME 4;
val it = true : bool
val it = false : bool
# SOME 3 = NONE; NONE = NONE;
val it = false : bool
val it = true : bool
#
```

- 「NONE」の形か「SOME ...」の形かは，isSome 関数を用いて判定できる．
- 「SOME v」の形からvを取り出すには，valOf 関数を用いる．
- SOME vとSOME wは，
  $v = w$のときは等しく，
  $v \neq w$のときは等しくない．
  SOME vとNONEは等しくない．

オプション型(教7.4節) (3)

```
# Int.fromString "3";
val it = SOME 3 : int option
# Int.fromString "aa";
val it = NONE : int option
# Int.fromString;
val it = fn : string -> int option
# fun averageInt (x,y) = if (x + y) mod 2 = 1 then NONE
>  else SOME ((x + y) div 2);
# val averageInt = fn : int * int -> int option
# averageInt (2,3);
val it = NONE : int option
# averageInt (9,3);
val it = SOME 6 : int option
```

返す値を持つときには「SOME 値」の形で返し，返す値がないときには「NONE」を返すようにする.

実習課題 (2)

1. 整数の組(x, y)を受けとって，$y \neq 0$であればSOME (x/y)を返し，$y = 0$であればNONEを返す関数divIntを与えよ．

```
# divInt (3,2); divInt (3,0);
val it = SOME 1.5 : real option
val it = NONE : real option
```

2. 文字列の対を受けとって，両方の文字列が整数を表わす文字列ならば，それらの整数を加算した値を表わす文字列をSOME 文字列の形で返し，そうでなければNONEを返す関数addIntStringを与えよ．

```
# addIntString ("12", "34");
val it = SOME "46" : string option
```

実習課題 (2)

3. 文字列が同じ文字列をつなげたwwの形であれば，SOME wを返し，そうでなければNONEを返す関数unfoldRepeatを与えよ．

```
# unfoldRepeat "nemunemu";
val it = SOME "nemu" : string option
# unfoldRepeat "nemunoki";
val it = NONE : string option
```

4. 2つの正整数k, nを受けとって，nがk以上n未満の約数をもてば，そのうちの1つを返し，なければNONEを返す関数isPrimeSubを与えよ．

```
# isPrimeSub (3,17447);
val it = SOME 73 : int option
```