プログラミングAIII

2024年度講義資料 (5)

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

多相性

```
# fun selfPair x = (x,x);
val selfPair = fn : ['a. 'a -> 'a * 'a]
# selfPair 1;
val it = (1, 1) : int * int
# selfPair true;
val it = (true, true) : bool * bool
# selfPair "aaa";
val it = ("aaa", "aaa") : string * string
#
```

- ここで定義した関数 selfPair は引数がどのような型であっても計算できる.
- このように，関数などが複数の型に対応できることを多相性とよぶ.

パラメトリックな多相性(教3.2節)

```
# fun selfPair x = (x,x);
val selfPair = fn : ['a. 'a -> 'a * 'a]
```

- selfPairの型は，ある程度決まっていて，引数の型が 'a なら，返り値の型は 'a * 'a となる．つまり型がパラメータ化されている． 'aは，型を表わす変数なので，型変数とよばれる.
- このように，関数の型が型変数を使って表わされ，変数が具体化されることで，関数がさまざまな型に対応するような多相性を，パラメトリックな多相性とよぶ.
- 型変数を使って表わされる型を多相型とよぶが， SMLでは，例外や参照型といった機能と併用するために，多相型をもつ関数定義に制約がある(教3.5節).

型推論と型チェック(教3.1節)

- SMLでは，関数定義式から最も一般的な型を計算し，コンパイル時に関数の型として返してくれる.
- このように自動的に，(最も一般的な)型を与える仕組みは， 型推論とよばれる.
- 一方で，式の型に整合性がとれなければ，型エラーを報告してくれる．コンパイル時に型の整合性をチェックする仕組みは， 静的型チェックとよばれる.
- 静的型チェックや型推論は，モダンなプログラミング言語における重要な仕組み.

```
# fun getLeft (x,y) = x;
val getLeft = fn : ['a, 'b. 'a * 'b -> 'a]
# fun round (x,y,z) = (y,x,z);
val round = fn : ['a, 'b, 'c. 'a * 'b * 'c -> 'b * 'a * 'c]
```


等値演算を使う型(教3.6節)

SMLの型には，等値性判定が使える型(整数など)と等値性判定が使えない型(実数など)があった． 以下の関数 isEqual の引数は，等値性判定(=)が使える型という制約が必要．制約は，'a#eqの#eqの部分で表わされている.

```
# fun isEqual (x,y) = if x = y then 1 else 0;
val isEqual = fn : ['a#eq. 'a * 'a -> int]
# isEqual (0,1);
val it = 0 : int
# isEqual ("aa","aa");
val it = 1 : int
# isEqual (0.0, 1.0);  (* 実数には利用できない *)
(interactive):9.0-9.17(195) Error:
  (type inference 016) operator and operand don't agree
  operator domain: 'PJB#eq * 'PJB#eq
          operand: 'PIY::{real, real32} * 'PJA::{real, real32}
```


明示的な型宣言(教3.3節)

- 最も一般的な型で使いたくない場合は，関数に型制約をつけることができる.
- 型制約は変数の後ろに「：型」という形で書く.
- 以下の例では，以前に定義した getLeft や round に型制約をつけている．型制約を付けなかった場合と得られる型を較べてみよ.

```
# fun getLeft (x,y:int) = x;
val getLeft = fn : ['a. 'a * int -> 'a]
# fun round (x:'t, y:'t, z) = (y,z,x);
val round = fn : ['a, 'b. 'a * 'a * 'b -> 'a * 'b * 'a]
#
```

関数の多重定義(教3.4節)

+ は， real 型に対しても， int 型に対しても定義されている．このように，関数がさまざまな型に対して多重に定義されているようなケースも，多相性の一種である．

```
# 1 + 1;
val it = 2 : int
# 1.0 + 1.0;
val it = 2.0 : real
# op +;
val it = fn : ['a::{int, ...}. 'a * 'a -> 'a]
#
```

もっとも，このような多相性は多くのプログラミング言語で導入されている．

目次

リスト(教6.1節，教6.2節)

組型に続き，もっとも基本的な複合型の1つであるリスト型について学習する．

リスト

- リストとは，リスト構造によって実現されている同じ型の要素の列のことである．要素が型'aをもつとき，そのリストの型は'a listとなる．
- 列$a_1, \ldots, a_n$のリストを$[a_1, \ldots, a_n]$と書く．
- 空リストを[]と書く． (nilともよぶ．)
- リストxsの先頭に要素xをつけたリストを，$x::xs$と書く．
- 中置演算子::(consとよぶ)は右結合で，リスト$[a_1, \ldots, a_n]$は， $a_1::\cdots::a_n::[]$の略記である．

いろいろな型のリスト

```
# [1,2,3];
val it = [1, 2, 3] : int list
# [#"a",#"b"]; ["abc"];
val it = [#"a", #"b"] : char list
val it = ["abc"] : string list
# [1, "a"];  (* 異なる型の要素はリストに出来ない *)
(interactive):35.0-35.7(362) Error:
  (type inference 023) operator and operand don't agree
  operator domain: ...
          operand: ...
# [[1,2],[3,4,5],[6]]; [(1,"a"),(2,"b")];
val it = [[1, 2], [3, 4, 5], [6]] : int list list
val it = [(1, "a"), (2, "b")] : (int * string) list
```

注意). int list list は，(int list) list の省略

リストの評価とリストの等価性

```
# 1::2::[];
val it = [1, 2] : int list
# [1, 1+1, 1+1+1];  (* リストの値は，要素の式の値のリスト *)
val it = [1, 2, 3] : int list
# [1,2,3] = 1::[2,3]; [2,2+1] = [1+1,3];
val it = true : bool
val it = true : bool  (* リストの等しさは，その値の等しさ *)
# [1] = [1,1]; [1,2] = [2,1];
val it = false : bool (* 個数も順番も関係ある *)
val it = false : bool
# [1.0] = [1.0];  (* 等価性の判定できない要素のとき *)
(interactive):15.0-15.12(31) Error:
  (type inference 026) operator and operand don't agree
  operator domain: 'PXB#eq * 'PXB#eq
          operand: 'PWJ::{real, real32} list * 'PXA::{real, real
```


リストの基本関数(教6.4節) (1)

```
# hd [1,2,3]; tl [1,2,3];
val it = 1 : int
val it = [2, 3] : int list
# null []; null [1];
val it = true : bool
val it = false : bool
# [1,2]@[3];
val it = [1, 2, 3] : int list
# []@[1,2]; ["abc"]@[];
val it = [1, 2] : int list
val it = ["abc"] : string list
# length [1,2]; length [];
val it = 2 : int
val it = 0 : int
# rev [1,2,3];
val it = [3, 2, 1] : int list
```

- hdとtlはそれぞれ，先頭要素と先頭要素を取り除いた残りのリストを返す．空リストに対してはエラー(Empty例外をなげる)．
- nullは空リストかどうかを判定する関数．
- 中置演算@は2つのリストを繋げる．空リストを繋げても同じリストのまま．
- lengthはリストの長さを返す関数．
- revはリストを反転したリストを返す関数

リストの基本関数(教6.4節) (2)

```
# concat ["aa", "bb", "cc"];
val it = "aabbcc" : string
# implode;
val it = fn : char list -> string
# implode [#"a", #"b", #"c"];
val it = "abc" : string
# explode;
val it = fn : string -> char list
# explode "hello";
val it = [#"h", #"e", #"l", #"l", #"o"] : char list
#
```

- concatは文字列のリストを受けとり，リストの文字列を連結して得られる文字列を返す関数
- implodeは文字のリストを受けとり，それらを繋げて得られる文字列を返す．
- explodeは，文字列を受けとり，その文字列を分解した文字のリストを返す．

リスト関数の多相性

リスト関数の多くは，どのような要素の型をもつリストであっても，共通に用いることが出来る．

```
# rev [1,2,3,4];
val it = [4, 3, 2, 1] : int list
# rev ["aa","bb","cc"];
val it = ["cc", "bb", "aa"] : string list
#
```

関数revの型を見てみると以下のようになっている：

```
# rev;
val it = fn : ['a. 'a list -> 'a list]
```

つまり，関数revは，どんな型の要素をもつリストに対しても使える．

実習課題(1)

1. 与えられたリストの最後の要素を返す関数last．ただし，空リストに対してはエラーとなってよい．

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

2. 自然数nとリストxsを受けとって，xsをn回連結したリストを返す関数 appendNtimes．（ヒント：再帰関数で定義する．）

```
# appendNtimes (3,[1,2]);
val it = [1, 2, 1, 2, 1, 2] : int list
```

3. 正の整数nを受けとって，リスト$[n, n-1, \ldots, 0]$を返す関数 natListDownFrom．

```
# natListDownFrom 7;
val it = [7, 6, 5, 4, 3, 2, 1, 0] : int list
```


実習課題(1)

4. 2つの整数n, m ($n \leq m$とする)をもらって，リスト$[n, n+1, \ldots, m]$ を返す関数 intListFromTo．

```
# intListFromTo (23,27);
val it = [23, 24, 25, 26, 27] : int list
```

5. 文字列が回文になっているかを真理値で返す関数 isPalindrome．

```
# isPalindrom "rotator"; isPalindrom "hello";
val it = true : bool
val it = false : bool
```


リストに関する組み込み関数

リストに関する組み込み関数は，Listストラクチャにある．

```
# List.last [1,2,3];
val it = 3 : int
# List.nth ([1,2,3],1);
val it = 2 : int
# List.nth ([1,2,3],2);
val it = 3 : int
# List.take ([1,2,3],1);
val it = [1] : int list
# List.drop ([1,2,3],1);
val it = [2, 3] : int list
```

lastに空リストを適用したり，nth，take，dropで指定がリストの範囲を越えていると，エラーとなる．

- List.lastは，リストの最後の要素を返す．
- List.nthは，リストxsと整数nを受けとり，xsのn番目の要素を返す．なお，先頭の要素を0番目と数える．
- List.takeは，リストxsと整数nを受けとり，xsの先頭からn個の要素からなるリストを返す．
- List.dropは，リストxsと整数nを受けとり，xsの先頭からn個の要素を取り除いたリストを返す．

対のリストに関する組み込み関数

対のリストに関する組み込み関数がListPairストラクチャに入っている．

```
# ListPair.zip ([1,2,3],[4,5,6]);
val it = [(1, 4), (2, 5), (3, 6)] : (int * int) list
# ListPair.unzip [(1,2),(3,4),(5,6)]
val it = ([1, 3, 5], [2, 4, 6]) : int list * int list
```

- ListPair.zipは，リストの対(xs, ys)を受けとって，xsとysの要素を先頭から順に対にしてできるリストを返す．片方がもう片方より長い場合は，残りの要素は捨てられる．
- ListPair.unzipは，対のリストを受けとって，対の第1要素からなるリストと，対の第2要素からなるリストを，対にして返す．

実習課題(2)

1. 整数nと要素aとリストxsを受けとって，リストxsのn番目の要素をaに置き替えたリストを返す関数 replaceNth．

```
# replaceNth (3,10,[1,2,3,4,5]);
val it = [1,2,3,10,5] : int list
```

2. 整数kとリスト $[a_1, a_2, \ldots, a_n]$ を受けとって，リスト $[a_{k+1}, a_{k+2}, \ldots, a_n, a_1, \ldots, a_k]$ を返す関数 rotate．

```
val rotate = fn : ['a. int * 'a list -> 'a list]
# rotate (3,[1,2,3,4,5]);
val it = [4, 5, 1, 2, 3] : int list
```

3. リスト $[a_1, a_2, \ldots, a_n]$ を受けとって，リスト $[(a_1, a_n), (a_2, a_{n-1}), \ldots, (a_n, a_1)]$ を返す関数 zipWithRev．

```
# zipWithRev [1,2,3];
val it = [(1, 3), (2, 2), (3, 1)] : (int * int) list
```