Jacques Garrigue, 2007 年 12 月 5 日

11 数式の正規化 (続)

多項式

前回の資料では，多変数の多項式を作る方法を紹介した.
関数を使った式を変数と見立てれば，多項式に変換することで式を正規化できる.

```
type 'a poly = (('a * float) list * float) list

val expr_of_poly : expr poly -> expr

let map_poly f : 'a poly -> 'b poly =
  List.map (fun (p,x) -> (List.map (fun (e,y) -> (f e, y)) p, x))

let rec normalize e =
  match e with
    Add _ | Mul _ | CMul _ ->
      let p = poly_of_expr e in
      expr_of_poly (map_poly normalize p)
  | _ -> map normalize e

let show_float n k =
  let s = Printf.sprintf "%g" k in
  if n > 0 && k < 0. then "("^s^")" else s

let rec show n = function
  | X -> "x"
  | Cst k -> show_float n k
  | Sin e -> "sin(" ^ show 0 e ^")"
  | Cos e -> "cos(" ^ show 0 e ^")"
  | Exp e -> "exp(" ^ show 0 e ^")"
  | Log e -> "log(" ^ show 0 e ^")"
  | Pow (e, k) -> show 3 e ^ "^" ^ show_float 3 k
  | e when n > 2 -> "("^ show 0 e ^")"
  | Mul (e1, e2) -> show 2 e1 ^ " * " ^ show 2 e2
  | CMul (k, e) -> show_float n k ^ " " ^ show 2 e
  | e when n > 1 -> "("^ show 0 e ^")"
  | Add (e1, e2) -> show n e1 ^ " + " ^ show 1 e2

# let e1 = Add(Mul(Sin X,Mul(Pow(Cos X,-1.), Pow(Exp X,2.))),
               CMul(-0.5,Pow(Mul(Exp X,Sin X),2.))) ;;
# show 0 e1;;
- : string = "sin(x) * cos(x)^(-1) * exp(x)^2 + (-0.5) (exp(x) * sin(x))^2"
# make_poly e1;;
- : ((expr * float) list * float) list =
[([(Sin X, 1.); (Cos X, -1.); (Exp X, 2.)], 1.);
 ([(Sin X, 2.); (Exp X, 2.)], -0.5)]
# let e2 = Add(Cos X, CMul(2., Pow(Sin X,-1.)));;
# let e3 = normalize (Mul (e1, e2));;
```

```
# show 0 e3;;
- : string = "-0.5 sin(x)^2 * cos(x) * exp(x)^2 + 2 cos(x)^(-1) * exp(x)^2"
```

練習問題 11.1 上記の expr_of_poly を定義せよ．いくつかの補助関数が必要になる．

疑似実数 float には特殊な値が含まれている．

```
# 1./.0.;;
- : float = infinity
# -1./.0.;;
- : float = neg_infinity
# infinity = infinity;;
- : bool = true
# infinity+.infinity = infinity;;
- : bool = true
# -1./.infinity;;
- : float = -0.
# (-1./.infinity) = 0.;;
- : bool = true
# 0./.0.;;
- : float = nan
# nan = nan;;
- : bool = false
```

特に，最後の nan に関する不等性に気を付けなければならない．自分に等しく実数があるのだ!

12 論理式の正規化

数式を正規化しようとすると様々な問題が起こる．それに比べて，論理式はやりかたが似ているものの，問題が少なくて済む．

命題論理

ここでは命題論理を考える．

$$
\begin{array}{rcll}
A & ::= & a, b, c \ldots & \text{記号 (アトム)} \\
P & ::= & A & \text{命題} \\
 & | & \neg P & \text{否定} \\
 & | & P \wedge P & \text{連言} \\
 & | & P \vee P & \text{選言} \\
 & | & P \rightarrow P & \text{含意}
\end{array}
$$

論理式の値

論理式に含まれる全てのアトムに真偽値を割当てると，その論理式の真偽が計算できる．

```
type prop =
    Atom of string
  | Neg of prop
  | And of prop * prop
  | Or  of prop * prop
  | Imp of prop * prop

let rec eval v = function
```

```
    Atom a -> List.assoc a v
  | Neg p -> not (eval v p)
  | And (p1, p2) -> eval v p1 && eval v p2
  | Or  (p1, p2) -> eval v p1 || eval v p2
  | Imp (p1, p2) -> not (eval v p1) || eval v p2
val eval : (string * bool) list -> prop -> bool
```

ド・モルガンの法則

eval を見れば，含意が他の関係から定義できることが分かる．

$$P \to Q = \neg P \vee Q$$

そして，否定が対合的であることも導ける．

$$\neg\neg P = P$$

さらに，連言と選言の間に次の法則が成り立つ．

$$\neg(P \wedge Q) = \neg P \vee \neg Q$$

$$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$$

$$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$$

選言標準形　上の法則を使えば，任意の命題論理式を以下のような標準形に変換できる．

$$\bigvee_i (\bigwedge_j a_{ij} \wedge \bigwedge_k \neg b_{ik})$$

例えば，

$$\begin{aligned} \neg(a \to (b \wedge c)) &= \neg(\neg a \vee b \wedge c) \\ &= \neg\neg a \wedge (\neg b \vee \neg c) \\ &= (a \wedge \neg b) \vee (a \wedge \neg c) \end{aligned}$$

練習問題 12.1　*1.* 論理式を選言標準形に変換する関数を定義せよ．

2. 選言標準形を以下の型で表現するように変換関数を改めよ．

```
(string * bool) list list
```