

# 基本データ型上の関数

胡 振江

東京大学 計数工学科

2008 年 10 月 27 日

Copyright © 2008 Zhenjiang Hu, All Right Reserved.

## 論理型

論理型 (Bool) は True と False だけを含む。

```
data Bool = False | True
```

## 論理型上の関数

### 論理否定

$$\begin{aligned} \text{not} & \quad :: \quad \text{Bool} \rightarrow \text{Bool} \\ \text{not False} & = \text{True} \\ \text{not True} & = \text{False} \end{aligned}$$

Remark:

- パターンマッチング (Pattern matching) による定義
- 書き換え規則 (Rewriting rules)
- $\text{not } \perp = \perp$

## 論理型上の関数

### 論理積・論理和

$$(\wedge), (\vee) \quad :: \quad Bool \rightarrow Bool \rightarrow Bool$$

$$False \wedge x = False$$

$$True \wedge x = x$$

$$False \vee x = x$$

$$True \vee x = True$$

Remark:

- $\perp \wedge False = \perp$
- $False \wedge \perp = False$
- $True \wedge \perp = \perp$
- Haskell では、" $\wedge$ "  $\Rightarrow$  "&&", " $\vee$ "  $\Rightarrow$  "||"

## 論理型上の関数

### 論理積の別の定義

$$\begin{aligned}(\wedge) & \quad :: \text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool} \\ \text{False} \wedge \text{False} & = \text{False} \\ \text{False} \wedge \text{True} & = \text{False} \\ \text{True} \wedge \text{True} & = \text{True} \\ \text{True} \wedge \text{False} & = \text{False}\end{aligned}$$

Remark:

- $\perp \wedge \text{False} = \perp$
- $\text{False} \wedge \perp = \perp$
- $\text{True} \wedge \perp = \perp$

## 論理型上の関数

### 比較演算子

$$\begin{aligned} (==) & \quad :: \text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool} \\ x == y & = (x \wedge y) \vee (\text{not } x \wedge \text{not } y) \end{aligned}$$

$$\begin{aligned} (\neq) & \quad :: \text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool} \\ x \neq y & = \text{not } (x == y) \end{aligned}$$

Remark:

- Haskell では、" $\neq$ "  $\Rightarrow$  " $\neq$ "
- 同値性を定義したい型には論理型だけではなく、他に多くの型がある。論理型上の定義はその一例に過ぎない。

## 論理型上の関数

比較演算子の定義：class/instance

```
class Eq  $\alpha$  where  
  (==), (≠) ::  $\alpha \rightarrow \alpha \rightarrow Bool$   
   $x \neq y = not (x == y)$ 
```

```
instance Eq Bool where  
   $x == y = (x \wedge y) \vee (not\ x \wedge not\ y)$ 
```

## 論理型上の関数

### その他の比較演算子の定義

```
class Eq  $\alpha \Rightarrow$  Ord  $\alpha$  where  
(<), (<=), (>), (>=) ::  $\alpha \rightarrow \alpha \rightarrow Bool$   
 $x \leq y = (x < y) \vee (x == y)$   
 $x > y = not (x \leq y)$   
 $x \geq y = (x > y) \vee (x == y)$ 
```

```
instance Ord Bool where  
False < False = False  
False < True = True  
True < False = False  
True < True = False
```

## 例

xor

$$\begin{aligned} \text{xor} &:: \text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool} \\ \text{xor } p \ q &= (p \wedge \text{not } q) \vee (\text{not } p \wedge q) \end{aligned}$$

imply

$$\begin{aligned} \text{imply} &:: \text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool} \\ \text{imply } p \ q &= \text{not } p \vee q \end{aligned}$$

leap: 閏年を判定する関数

$$\begin{aligned} \text{leap} &:: \text{Int} \rightarrow \text{Bool} \\ \text{leap } y &= y \text{ 'mod' } 4 == 0 \wedge \\ &\quad \text{imply } (y \text{ 'mod' } 100 == 0) (y \text{ 'mod' } 400 == 0) \end{aligned}$$

## 自然数型

自然数型はすべての自然数 (0, 1, 2, ...) 含む。

```
data Nat = Zero | Succ Nat
```

Remark:

- *Nat* は再帰的に定義されている。
- *Zero*, *Succ* はデータ構成子である。
- 例: *Zero*, *Succ Zero*, *Succ (Succ Zero)*

## 自然数上の関数定義

### 加算

$$\begin{aligned} (+) & \quad :: \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat} \\ m + \text{Zero} & \quad = m \\ m + (\text{Succ } n) & \quad = \text{Succ } (m + n) \end{aligned}$$

練習問題：  $\text{Zero} + \text{Succ } (\text{Succ } \text{Zero})$  を評価列を示せ。



## 自然数上の関数定義

### 比較演算

**instance Eq Nat where**

$Zero == Zero = True$

$Zero == Succ\ n = False$

$Succ\ m == Zero = False$

$Succ\ m == Succ\ n = m == n$

**instance Ord Nat where**

$Zero < Zero = False$

$Zero < Succ\ n = True$

$Succ\ m < Zero = False$

$Succ\ m < Succ\ n = m < n$

## 自然数上の関数定義

### 比較演算

自然数を次のように定義すれば、比較演算子の定義が自動的に導出される。

```
data Nat = Zero | Succ Nat
deriving (Eq, Ord)
```



## 自然数上の関数定義

### 階乗

$$\begin{aligned} fact &:: Nat \rightarrow Nat \\ fact\ Zero &= Succ\ Zero \\ fact\ (Succ\ n) &= Succ\ n \times fact\ n \end{aligned}$$

### Fibonacci 関数

$$\begin{aligned} fib &:: Nat \rightarrow Nat \\ fib\ Zero &= Zero \\ fib\ (Succ\ Zero) &= Succ\ Zero \\ fib\ (Succ\ (Succ\ n)) &= fib\ (Succ\ n) + fib\ n \end{aligned}$$

# 文字型

文字型 *Char* は ASCII (American Standard Code for Information Interchange) 文字の集まりである。

← 下は、7ビットで0000000~1111111(2進数)の128のコードを0X00~0X7F(16進数)で表記している

上位3ビット→ ↓下位4ビット	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	ˆ	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAC	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF/NL	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

## 文字を操作する関数

- `ord :: Char → Int`: 文字を対応する ASCII 符号の整数に変換

$$\text{ord 'b'} \Rightarrow 98$$

- `chr :: Int → Char`: ASCII 符号の整数を対応する文字に変換

$$\text{chr 98} \Rightarrow \text{'b'}$$

- 関係演算子：文字の間は比較できる.

```
instance Eq Char where  
x == y = ord x == ord y
```

```
instance Ord Char where  
x < y = ord x < ord y
```

## 文字型上の関数の定義

- `isDigit`: 文字が数字であることを判定する関数.

$$\begin{aligned} \text{isDigit} &:: \text{Char} \rightarrow \text{Bool} \\ \text{isDigit } x &= '0' \leq x \wedge x \leq '9' \end{aligned}$$

- `capitalise`: 小文字を大文字に変える関数.

$$\begin{aligned} \text{capitalise} &:: \text{Char} \rightarrow \text{Char} \\ \text{capitalise } x & \mid \text{isLower } x = \text{chr}(\text{offset} + \text{ord } x) \\ & \mid \text{otherwise} = x \\ & \text{where } \text{offset} = \text{ord } 'A' - \text{ord } 'a' \end{aligned}$$

## 評価例

$$\begin{aligned} & \text{capitalise 'a'} \\ = & \quad \{ \text{definition and } \text{isLower 'a'} = \text{True} \} \\ & \text{chr}(\text{offset} + \text{ord 'a'}) \\ = & \quad \{ \text{definition of offset} \} \\ & \text{chr}(\text{ord 'A'} - \text{ord 'a'} + \text{ord 'a'}) \\ = & \quad \{ \text{arithmetic} \} \\ & \text{chr}(\text{ord 'A'}) \\ = & \quad \{ \text{since } \text{chr}(\text{ord } c) = c \text{ for all } c \} \\ & \text{'A'} \end{aligned}$$

## 文字列型

文字列型 *String* は文字の列の集まりである.

```
{ " ", "hello", "This is a string.", ... }
```

```
type String = [Char]
```

```
? "a"
```

```
"a"
```

```
? "Hello World"
```

```
"Hello World"
```

```
? putStr "Hello World"
```

```
Hello World
```

## 文字列上の関数

- $show :: a \rightarrow String$ : 任意の型のデータを文字列に変換

$show\ 100 \Rightarrow "100"$

$show\ True \Rightarrow "True"$

$show\ (show\ 100) \Rightarrow "\\\"100\\\""$

- $++ :: String \rightarrow String \rightarrow String$ : 二つの文字列をつなぐ接続演算子

$"hello" ++ " " ++ "world" \Rightarrow "helloworld"$

- 比較演算子：文字列の比較は通常の辞書式順に従う

## 整数型とその上の関数

整数型はすべての整数から構成されている。

`Int` : single precision integer

`Integer` : arbitrary precision integer

算術演算子	使用例
<code>+</code> (加算)	$2 + 3 \Rightarrow 5$
<code>-</code> (減算)	$2 - 3 \Rightarrow -1$
<code>*</code> (乗算)	$2 * 3 \Rightarrow 6$
<code>/</code> (除算)	$3/2 \Rightarrow 1.5$
<code>^</code> (べき乗算)	$2^3 \Rightarrow 8$
<code>div</code> (整数除算)	$div\ 3\ 2 \Rightarrow 1$
	$3\ 'div'\ 2 \Rightarrow 1$
<code>mod</code> (整数除余)	$mod\ 5\ 3 \Rightarrow 2$
	$5\ 'mod'\ 3 \Rightarrow 2$

## 関数定義

- 階乗

$$fact \quad :: \quad Integer \rightarrow Integer$$
$$fact \ 0 \quad = \ 1$$
$$fact \ (n + 1) \ = \ (n + 1) * fact \ n$$

- 整数の符号を計算する関数

$$sign \quad :: \quad Int \rightarrow Int$$
$$sign \ n \quad | \quad n > 0 \quad = \ 1$$
$$| \quad n == 0 \quad = \ 0$$
$$| \quad n < 0 \quad = \ -1$$

## 浮動小数点数型とその上の関数

浮動小数点数型はすべての浮動小数点数から構成されている。

**Float** : single precision floating-point numbers

**Double** : arbitrary precision floating-point numbers

演算子	使用例
+ (加算)	$2.3 + 3.3 \Rightarrow 5.6$
- (減算)	$2.5 - 3 \Rightarrow -0.5$
* (乗算)	$2.5 * 2.5 \Rightarrow 6.25$
/ (除算)	$3.2/2 \Rightarrow 1.6$

## 数型上の関数の定義

例：数の絶対値を返す関数 `abs`.

$$\begin{aligned} \text{abs} &:: \text{Num } a \Rightarrow a \rightarrow a \\ \text{abs } x &= \text{if } x < 0 \text{ then } -x \text{ else } x \end{aligned}$$

読みやすいために、次のように書いてもよい.

$$\begin{array}{l|l} \text{abs } x & x < 0 & = -x \\ & \text{otherwise} & = x \end{array}$$

## 宿題

- Hugs システムを使って、基本型上の関数をテストする。
- 教科書の第二章を復習し、教科書中の練習問題を解く。