

知能ソフトウェア特論

Intelligent Software

項書換え系 (1)  
代数的仕様と項書換え

---

Term Rewriting Systems(1)  
Algebraic Specification and  
Term Rewriting

# 項書換え系入門：基本的なアイデア

(Introduction to term rewriting systems: Basic idea)

等式 (equation)  仕様 (specification)

$$x \times (y + z) = x \times y + x \times z$$

書換え規則 (rewrite rule)  プログラム (program)

$$x \times (y + z) \rightarrow x \times y + x \times z$$

書換え (rewriting)  計算 (computation)

$$1 + \underbrace{(2 \times (a + b))}_{\text{left instance}} \rightarrow 1 + \underbrace{(2 \times a + 2 \times b)}_{\text{right instance}}$$

書換え規則の左辺のインスタンス

(an instance of the left-hand side  
of a rewrite rule)

対応する右辺のインスタンス

(the corresponding instance  
of the right-hand side)

# 項書換え系入門：応用

(Applications)

記号計算

(symbolic computation)

定理証明

(theorem proving)

ソフトウェアの代数的仕様記述

(algebraic specification of software)

ソフトウェアの自動検証

(automated verification of software)

# 1. 項書換え系の構文論 (1/2)

(Syntax of term rewriting systems)

基本的な記号は次の3種類

●変数  $x, y, z, \dots$

●定数  $0, 1, \dots, a, b, c, \dots$

●関数記号  $f, g, h, \dots$

関数記号ごとにアリティ (引数の個数)  
が固定している

項

- 1) 変数および定数は項である.
- 2)  $f$  がアリティ  $n$  の関数記号,  
 $t_1, \dots, t_n$  が項ならば,  
 $f(t_1, \dots, t_n)$  は項である.

Atomic symbols used in term rewriting systems are classified into variables ( $x, y, z, \dots$ ), constants ( $0, 1, \dots, a, b, c, \dots$ ), and function symbols ( $f, g, h, \dots$ ) with fixed arity, the number of arguments they take.

A term is constructed as follows.

- 1) A variable and a constant are terms.
- 2) If  $f$  is a function symbol of arity  $n$  and if  $t_1, \dots, t_n$  are terms, then  $f(t_1, \dots, t_n)$  is a term.

# 1. 項書換え系の構文論 (2/2)

**書換え規則** 2つの項 (左辺  $l$ , 右辺  $r$ ) の順序対  $l \rightarrow r$  のこと.

左辺のインスタンスを対応する右辺のインスタンスに書換え可能

A rewrite rule is an ordered pair  $l \rightarrow r$  of terms (the left-hand side  $l$  and the right-hand side  $r$ ).

Any instance of  $l$  can be rewritten to the corresponding instance of  $r$ .

**項書換え系** 書換え規則の集合  $R$ .

A term rewriting system (TRS)  $R$  is a set of rewrite rules.

$$\text{Example : } R = \left\{ \begin{array}{ll} f(x, g(y)) \rightarrow g(f(x, y)), & g(g(x)) \rightarrow x \\ f(x, a) \rightarrow x, & g(a) \rightarrow b \\ f(x, b) \rightarrow g(x), & g(b) \rightarrow a \end{array} \right\}$$

## 2. 項書換え系の操作的意味論 (1/5)

(Operational semantics of term rewriting)

書換え可能  $s \rightarrow_R t$  ( $s \rightarrow t$ )

$R$ に含まれる1つの書換え規則を  
項  $s$  の部分項に1回適用して  
項  $t$  が得られる.

A term  $s$  is reducible to a term  $t$ ,  
notation  $s \rightarrow_R t$  (or  $s \rightarrow t$ ), if  $t$   
can be obtained by applying a  
rewrite rule in  $R$  once to a  
subterm (a part) of  $s$ .

【Example】 From  $f(x, g(y)) \rightarrow g(f(x, y))$ , we see

$$g(\underline{f(g(a), g(b))}) \rightarrow_R g(g(f(g(a), b)))$$

部分項

subterm

左辺のインスタンス  
(instance: 実例)

an instance of the left-  
hand side

= リデックス  
(reducible expression)

a redex  
(reducible expression)

## 2. 項書換え系の操作的意味論 (2/5)

### 書換え列

$$t_0 \rightarrow_R t_1 \rightarrow_R \cdots \rightarrow_R t_n \rightarrow_R \cdots$$

「計算」を表現

A rewrite sequence represents a computation.

### 正規形

$t_n$  をそれ以上書換えられないとき  $t_n$  は初期項  $t_0$  の正規形

「計算結果」を表現

A term  $t_n$  is a normal form of the initial term  $t_0$  if  $t_n$  cannot be rewritten any more.

The normal form represents the result of the computation.

## 2. 項書換え系の操作的意味論 (3/5)

### 書換え戦略

$s$  から書換え可能な  $t$  は一般には何通りもある (非決定性).

いずれを選ぶか.

**Rewrite strategy:** In general, the computation is non-deterministic, i.e., there are many  $t$ 's to which  $s$  is reducible. A rewrite strategy determines which one to choose.

**最内最左戦略** (最も内側のリデックスのうち最も左側を選択)

$f(\underline{g(a)}, g(g(a)))$

$\rightarrow f(b, \underline{g(g(a))})$

$\rightarrow f(b, \underline{g(b)})$

$\rightarrow \underline{f(b, a)}$

$\rightarrow b$

$$R = \left\{ \begin{array}{ll} f(x, g(y)) \rightarrow g(f(x, y)), & g(g(x)) \rightarrow x \\ f(x, a) \rightarrow x, & g(a) \rightarrow b \\ f(x, b) \rightarrow g(x), & g(b) \rightarrow a \end{array} \right\}$$

**Leftmost-innermost strategy** chooses the leftmost redex from the innermost ones. Here is an example.



## 2. 項書換え系の操作的意味論 (4/5)

最外最左戦略 (最も外側のリデックスのうち最も左側を選択)

$$\underline{f(g(a), g(g(a)))}$$

$$\rightarrow g(\underline{f(g(a), g(a))})$$

$$\rightarrow g(\underline{g(f(g(a), a))})$$

$$\rightarrow \underline{f(g(a), a)}$$

$$\rightarrow \underline{g(a)}$$

$$\rightarrow b$$

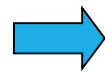
Leftmost-outermost strategy  
chooses the leftmost redex  
from the outermost ones.

$$R = \left\{ \begin{array}{ll} f(x, g(y)) \rightarrow g(f(x, y)), & g(g(x)) \rightarrow x \\ f(x, a) \rightarrow x, & g(a) \rightarrow b \\ f(x, b) \rightarrow g(x), & g(b) \rightarrow a \end{array} \right\}$$

## 2. 項書換え系の操作的意味論 (5/5)

### 停止性

無限の書換え列  $t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \dots$  が存在しない  
(計算は必ず停止)



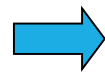
第2回で扱う

discuss it in the second lecture.

A TRS is **terminating** (or has a **termination property**) if there exists no infinite rewrite sequence., i.e., the computation will terminate definitely .

### 合流性

書換え戦略によらず正規形は (もしあれば) 唯一  
(計算結果は高々 1 つ)



第3回で扱う

discuss it in the third lecture.

A TRS is **confluent** (or has a **confluence property**) if there exists no or unique normal form, i.e., there exists at most one result of the computation.

# 3. ソフトウェアの代数的仕様記述 (1/13)

(Algebraic specification of software)

## 代数的仕様

関数間の関係を等式の集合で記述することにより

**抽象データ型**を定義するもの.

Algebraic specifications define abstract data types by describing relationships among functions in a set of equations.

## 直接実行

等式  $l = r$  を  $l \rightarrow r$  に向き付けて

項書換え系として実行.

Direct execution of algebraic specifications as TRSs are possible by directing equations  $l = r$  to  $l \rightarrow r$ .

# 3. ソフトウェアの代数的仕様記述 (2/13)

## 例題1 スタック

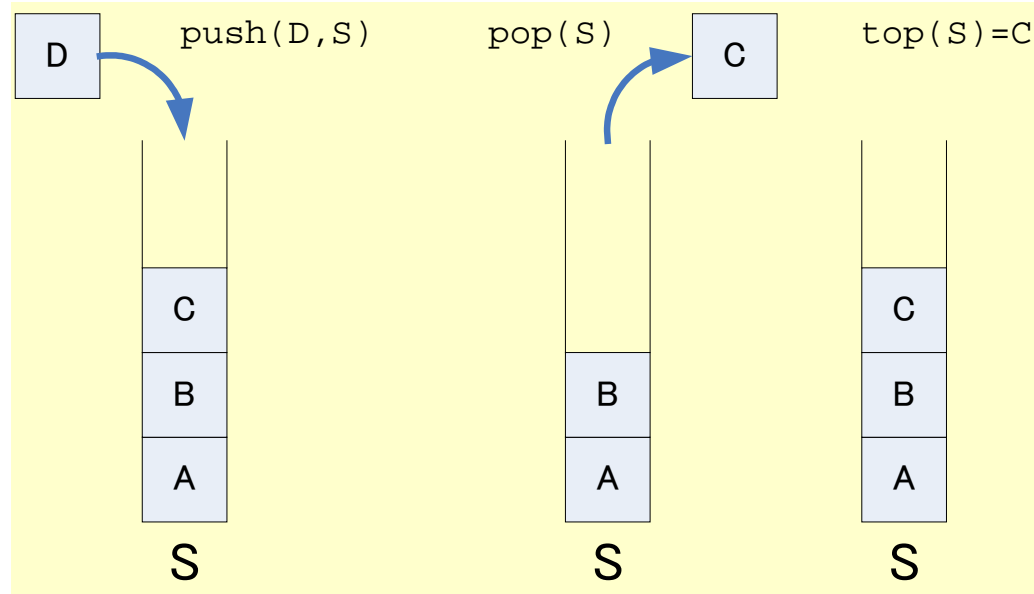
(Example 1: Stack)

$\text{push}: \text{Element} \times \text{Stack} \rightarrow \text{Stack}$

$\text{pop}: \text{Stack} \rightarrow \text{Stack}$

$\text{top}: \text{Stack} \rightarrow \text{Element}$

$\text{emptyStack}: \text{Stack}$



### 3. ソフトウェアの代数的仕様記述 (3/13)

$\text{pop}(\text{push}(x, y)) = y$

$\text{top}(\text{push}(x, y)) = x$

【直接実行の例】

(Example of direct execution)

$\text{top}(\text{pop}(\text{push}(A, \text{push}(B, \text{pop}(\text{push}(C, \text{push}(D, \text{emptyStack})))))))$

→  $\text{top}(\text{pop}(\text{push}(A, \text{push}(B, \text{push}(D, \text{emptyStack})))))$

→  $\text{top}(\text{push}(B, \text{push}(D, \text{emptyStack})))$

→ B

### 3. ソフトウェアの代数的仕様記述 (4/13)

#### 例題 2 : 自然数の加算

(Example 2: Addition of natural numbers)

後者関数

$s(x) = x + 1$ により自然数を  
項  $0, s(0), s(s(0)), \dots$  で表現.

The successor function  $s(x) = x + 1$  allows us to represent the natural numbers as terms  $0, s(0), s(s(0)), \dots$

```
s: Nat → Nat
```

```
plus: Nat × Nat → Nat
```

```
plus(0, x) = x
```

```
plus(s(x), y) = s(plus(x, y))
```

パターン  $0$  と  $s(x)$  で第1引数に来るすべての自然数について場合を尽くしている

The patterns  $0$  and  $s(x)$  cover all the cases for possible natural numbers for the first argument.

### 3. ソフトウェアの代数的仕様記述 (5/13)

```
plus(0,x) = x
```

```
plus(s(x),y) = s(plus(x,y))
```

【直接実行の例】

(Example of direct execution)

```
plus(s(s(0)),s(s(0)))
```

```
→ s(plus(s(0),s(s(0))))
```

```
→ s(s(plus(0,s(s(0)))))
```

```
→ s(s(s(s(0))))
```

2+2 → → → 4

## 【補足】 階乗

(Supplementary note: Factorial)

```
factorial: Nat → Nat
factorial(0) = s(0)
factorial(s(x)) = times(s(x), factorial(x))
```

$\text{times}(a, b)$  はかけ算  $a \times b$ . 別途定義してあるとする.

$\text{times}(a, b)$   
means  $a \times b$ ,  
defined  
elsewhere.

関数型プログラミング言語 Haskell による実装

```
factorial    :: Int → Int
factorial 0  = 1
factorial (n+1) = (n+1) * factorial n
```

$n+1$  は  $s(n)$  の略記.  
ここに任意の数式を書けるわけではない.

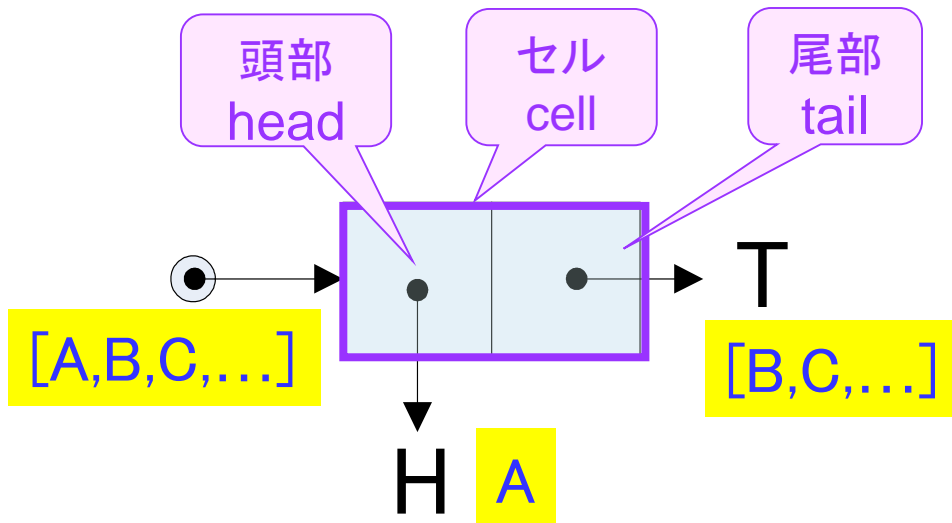
Implementation in  
Haskell, a functional  
programming language

$n+1$  is an abbreviation of  
 $s(n)$ . We cannot write an  
arbitrary expression here.



## 【補足】 リスト構造 (1/3)

リスト構造 : データの列  $[A, B, C, \dots]$   
を表現するデータ構造



これを, 項  $\mathbf{cons(H, T)}$  で表現

簡易記法:  $\mathbf{H:T}$

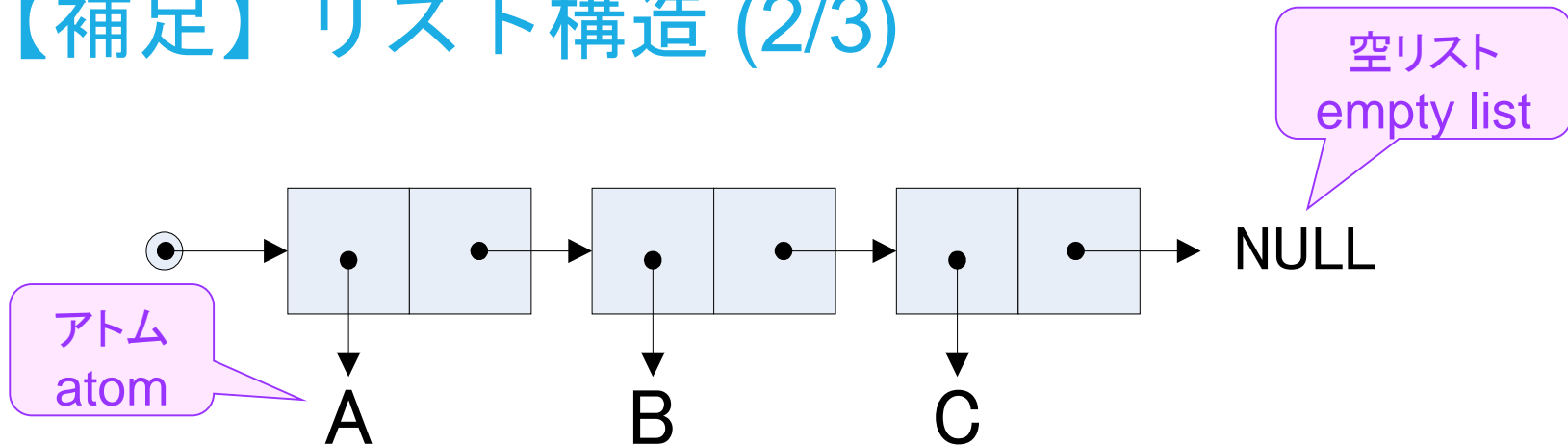
(Supplementary note:  
List structure)

List structure is a data structure used to represent a sequence  $[A, B, C, \dots]$  of data.

It is implemented as a cell consisting of two parts: the head for representing the first item of the list and the tail for the subsequence starting from the second item.

The cell consisting of the head  $H$  and the tail  $T$  is represented by the term  $\mathbf{cons(H, T)}$  and abbreviated as  $\mathbf{H:T}$ .

## 【補足】 リスト構造 (2/3)



**A :** ( **B :** ( **C :** **NULL** ) )

= **A :** **B :** **C :** **NULL**

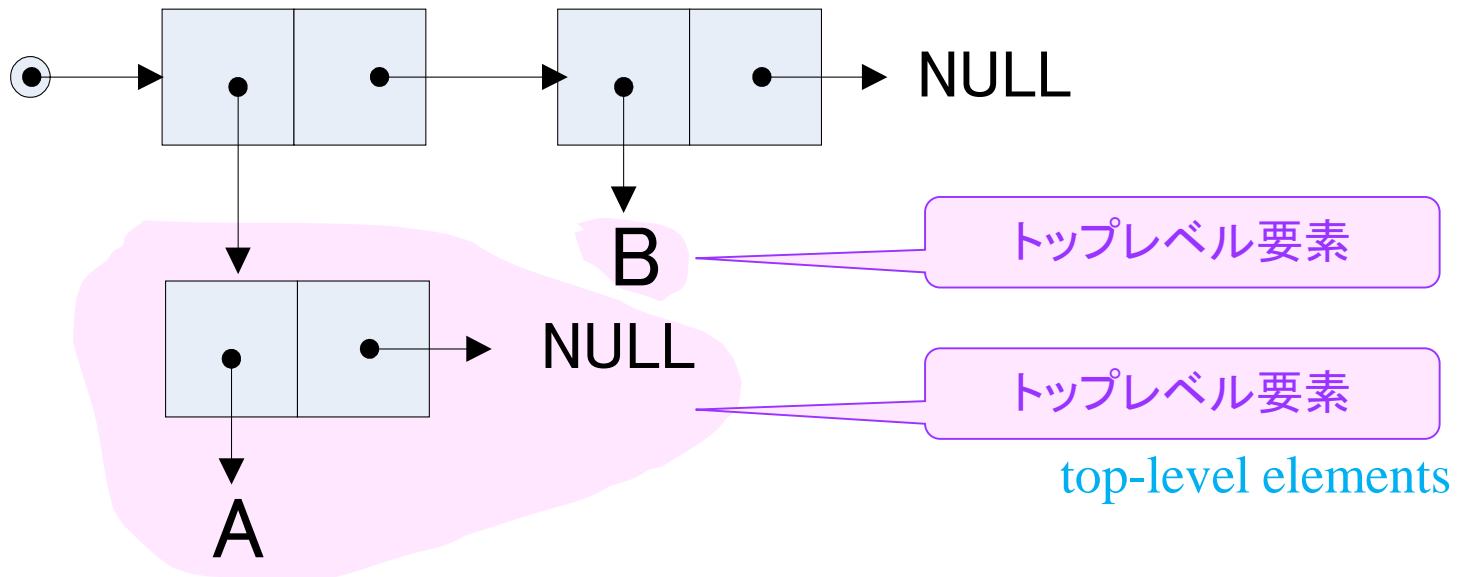
= **[A, B, C]**

簡易記法  
abbreviation

**: は右結合演算子**

The **:** is a right-associative operator.

## 【補足】 リスト構造 (3/3)



**( A : NULL ) : B : NULL**

**[ [ A ] , B ]**

### 3. ソフトウェアの代数的仕様記述 (6/13)

例題 3  $n$  番目に小さな素数  
(先頭は 0 番目)

素数の集合 = {2,3,5,7,11,...}  
最外最左戦略で `prime(2) → ... → 5`

**Example 3:** The  $n$ th smallest prime number  
(where 0th is the first one)

The set of prime number is {2,3,5,7,11,...}.  
The leftmost-outermost strategy will reduce `prime(2)` to 5.

$n$  番目に小さな素数

`prime(n) = nth(primes(), n)`

`prime(n)` returns the  $n$ th smallest prime number.

先頭から  $n$  番目のデータを返す

素数が昇順に並ぶ無限リスト

`nth(L, n)` returns the  $n$ th element of the list  $L$ .

`primes()` returns the infinite list of the prime numbers arranged in the ascending order.

### 3. ソフトウェアの代数的仕様記述 (7/13)

**$\text{nth}(L, n)$**

はリスト  $L$  の  $n$  番目の要素

$$\text{nth}(x:y, 0) = x$$

$$\text{nth}(x:y, s(n)) = \text{nth}(y, n)$$

**$\text{nth}(L, n)$**

returns the  $n$ th  
element of the list  $L$

**The first (0th) element of  $x:y$  is  $x$ .**

**The  $(n+1)$ th element of  $x:y$  is the  $n$ th element of  $y$ .**

### 3. ソフトウェアの代数的仕様記述 (8/13)

素数が昇順に並ぶ無限リスト

```
primes() = sieve(ints(s(s(0))))
```

**primes()** returns the infinite list of the prime numbers in the ascending order.

自然数  $x$  以降の自然数の無限リスト

```
ints(x) = x:ints(s(x))
```

**ints( $x$ )** returns the infinite list of the natural numbers starting from the natural number  $x$  in the ascending order.

```
ints(s(s(0)))  
= [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, ...]
```

**sieve**

```
primes()  
= [2, 3, 5, 7, 11, 13, .....]
```

# 3. ソフトウェアの代数的仕様記述 (9/13)

エラトステネスのふるい

```
sieve(x:y) = x:sieve(filt(x, y))
```

`sieve(x:y)` returns the list of prime numbers starting from  $x$  by filtering out the non-prime numbers from the list  $y$ , based on the *Eratosthenes' sieve* algorithm.

sieve [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,...]



2 : filter [3,4,5,6,7,...] by 2, followed by sieve



2 : sieve [3,5,7,9,11,13,15,...]



2 : 3 : filter [5,7,9,11,13,15,...] by 3, followed by sieve



### 3. ソフトウェアの代数的仕様記述 (10/13)

**filt**( $x, L$ )は  $x$  の倍数をすべて  
リスト  $L$  から削除するフィルター

```
filt(x, y:z) = if(eq(mod(y,x), 0),  
                filt(x, z),  
                y:filt(x, z))
```

**filt**( $x, L$ ) returns the list  
obtained from the list  $L$  by  
filtering out all the multiples  
of  $x$ .

**eq**( $x, y$ ) means  $x=y$  and  
**mod**( $y, x$ ) means the  
remainder of  $y \div x$ .

(Define them in the Exercise.)

If the first element  $y$  is divided by  $x$  (i.e.,  $y \bmod x$  equals 0),  
then ignore  $y$  and continue the filtering of  $z$ ,  
else save  $y$  for the head and continue the filtering of  $z$  for the tail.

```
if(true, x, y) = x  
if(false, x, y) = y
```

**if**( $C, x, y$ ) represents the  
conditional expression. It  
returns  $x$  if the condition  $C$   
is reduced to true; or  $y$  if  $C$   
is reduced to false.



### 3. ソフトウェアの代数的仕様記述 (11/13)

```
prime 2
```

```
→nth(primes(), 2)
```

```
→nth(sieve ints 2, 2)
```

```
→nth(sieve 2:ints 3, 2)
```

```
→nth(2:sieve f(2,ints 3), 2)
```

```
→nth(sieve f(2,ints 3), 1)
```

最外最左戦略

leftmost outermost  
reduction strategy

遅延評価

delayed evaluation

2 などは  $s(s(0))$  などの略記.

f は `filt` の略

1引数関数 `prime`, `ints`, `sieve` の引数を囲む括弧は省略

For the simplicity of the expressions:

- integers such as **2** represents the terms such as  $s(s(0))$ ,
- **f** is the abbreviation for **filt**,
- parentheses surrounding the argument of unary functions **prime**, **ints**, and **sieve** are omitted.

### 3. ソフトウェアの代数的仕様記述 (12/13)

```
nth(sieve f(2,ints 3),1)
```

```
→nth(sieve f(2,3:ints 4),1)
```

```
→nth(sieve 3:f(2,ints 4),1)
```

```
→nth(3:sieve f(3,f(2,ints 4)),1)
```

```
→nth(sieve f(3,f(2,ints 4)),0)
```

### 3. ソフトウェアの代数的仕様記述 (13/13)

```
nth(sieve f(3,f(2,ints 4)),0)
→nth(sieve f(3,f(2,4:ints 5)),0)
→+ nth(sieve f(3,f(2,ints 5)),0)
→nth(sieve f(3,f(2,5:ints 6)),0)
→nth(sieve f(3,5:f(2,ints 6)),0)
→+ nth(sieve 5:f(3,f(2,ints 6)),0)
→nth(5:sieve f(5,f(3,f(2,ints 6))),0)
→5
```

→+ は、1ステップ以上の書換えを表す

→+ denotes one or more steps of rewriting

# 演習問題 5

自然数を 0 と後者関数  $s$  を用いてコード化したとき、以下の関数を定義する代数的仕様を示せ。

(それにより例題 3 の記述が完結する。組込み演算子  $=$  は使えないものとする。)

- (1)  $eq(x, y)$ :  $x=y$  ならば true,  
                でなければ false
- (2)  $ge(x, y)$ :  $x \geq y$  ならば true,  
                でなければ false
- (3)  $minus(x, y)$ :  $x \geq y$  ならば  $x-y$ ,  
                    でなければ 0
- (4)  $mod(x, y)$ :  $x$  を  $y$  で割った余り  
( $x$  から  $y$  を可能な限り減じていく)

**[HINT]** You have to define the equality operator  $eq$  without built-in equality  $=$ . For example, your answer including an equation like

$eq(x, y) = \text{if}(x=y, \text{true}, \text{false})$   
is incorrect.

# Exercise 5

When the natural numbers are encoded by  $0$  and the successor function  $s$ , write the algebraic specifications of the following functions for completing the description for Example 3, assuming the built-in operator  $=$  is not available.

- (1)  $eq(x, y)$  returns true if  $x=y$ ;  
                    or false, otherwise.
- (2)  $ge(x, y)$  returns true if  $x \geq y$ ;  
                    or false, otherwise.
- (3)  $minus(x, y)$  returns  $x-y$  if  $x \geq y$ ;  
                    or 0, otherwise.
- (4)  $mod(x, y)$  returns the remainder for  $x \div y$ .  
(Subtract  $y$  from  $x$  as long as possible.)

A correct answer should include four equations with the left-hand sides

$eq(0, 0)$ ,  $eq(0, s(y))$ ,  $eq(s(x), 0)$ , and  
 $eq(s(x), s(y))$ .

Make sure that  $eq(s(0), s(s(s(0))))$  is rewritten to false.