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)

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

書換え (rewriting) — 計算 (computation)

$$1 + (2 \times (a + b)) \to 1 + (2 \times a + 2 \times b)$$

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)

関数型プログラミング (functional programming)

代数的仕様記述 (algebraic specification)

自動検証 (automated verification)

記号計算 (symbolic computation)

定理証明 (theorem proving)

1．項書換え系の構文論 (1/2)

(Syntax of term rewriting systems)

基本的な記号

- 変数　$x, y, z, \ldots$
- 定数　$0, 1, \ldots, a, b, c, \ldots$
- 関数記号　$f, g, h, \ldots$
  アリティ（引数の個数）は記号ごとに固定

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

項

- 変数および定数は項
- f がアリティ n の関数記号，
  $t_1, \ldots, t_n$ が項ならば
  $f(t_1, \ldots, t_n)$ は項

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 ,⋯, t_n are terms, then $f(t_1,\cdots,t_n)$ 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 $l \to r$ 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 $l \to r$ 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.

Example :

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

参考：ブール代数で，
f =XOR, g =NOT, a =0, b=1
と解釈する.

Each equation is true if you interpret f = XOR, g = NOT, a = 0, b = 1 on Boolean algebra.

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　$s \to_R t$ （$s \to t$）

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 $s \to_R t$ (or $s \to t$), if *t* can be obtained by applying a rewrite rule in *R* once to a subterm (a part) of *s*.

Example： From $f(x, g(y)) \to g(f(x, y))$, we see

$$g(f(g(a), g(b))) \to_R g(g(f(g(a), b)))$$

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　$t_0 \to_R t_1 \to_R \cdots \to_R t_n \to_R \cdots$

「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

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

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

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

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

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

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

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

2．項書換え系の操作的意味論 (5/5)

停止性

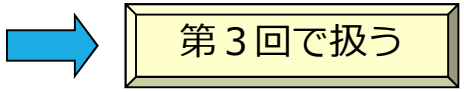
無限の書換え列 $t_0 \to t_1 \to t_2 \to \cdots$ が存在しない
（計算は必ず停止）

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）



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 $l \to r$ 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 $l \to r$.

3．ソフトウェアの代数的仕様記述 (2/13)

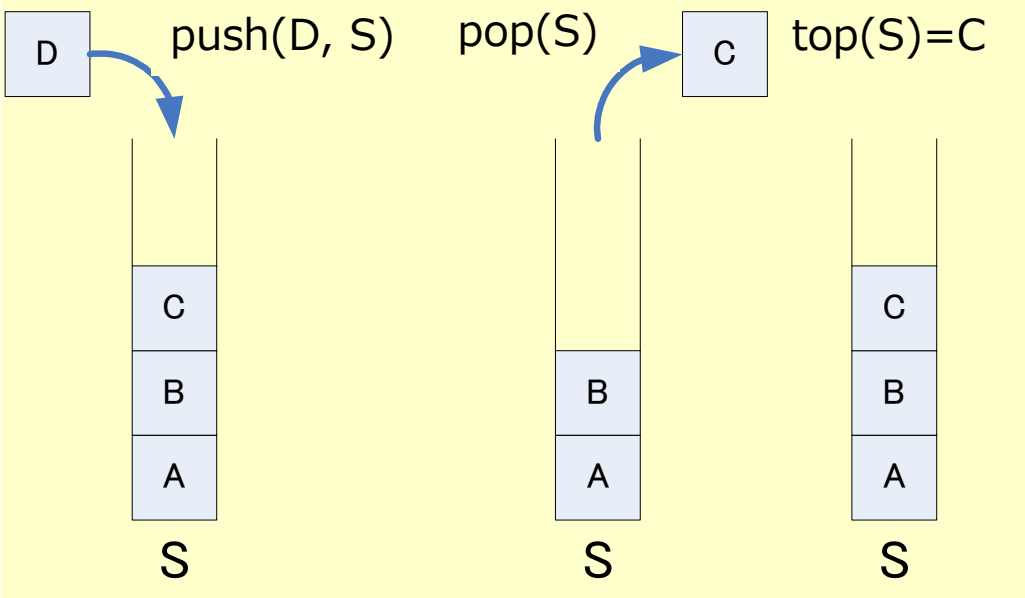
例題１　スタック　(Example 1: Stack)

push: Element × Stack → Stack

pop: Stack → Stack

top: Stack → Element

emptyStack: Stack



3．ソフトウェアの代数的仕様記述 (3/13)

pop(push(x,y)) = y
top(push(x,y)) = x

【直接実行の例】 (Example of direct execution)

top(pop(push(A,push(B,pop(push(C,push(D,emptyStack)))))))
→ top(pop(push(A,push(B,push(D,emptyStack)))))
→ top(push(B,push(D,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)),…
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)),…

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

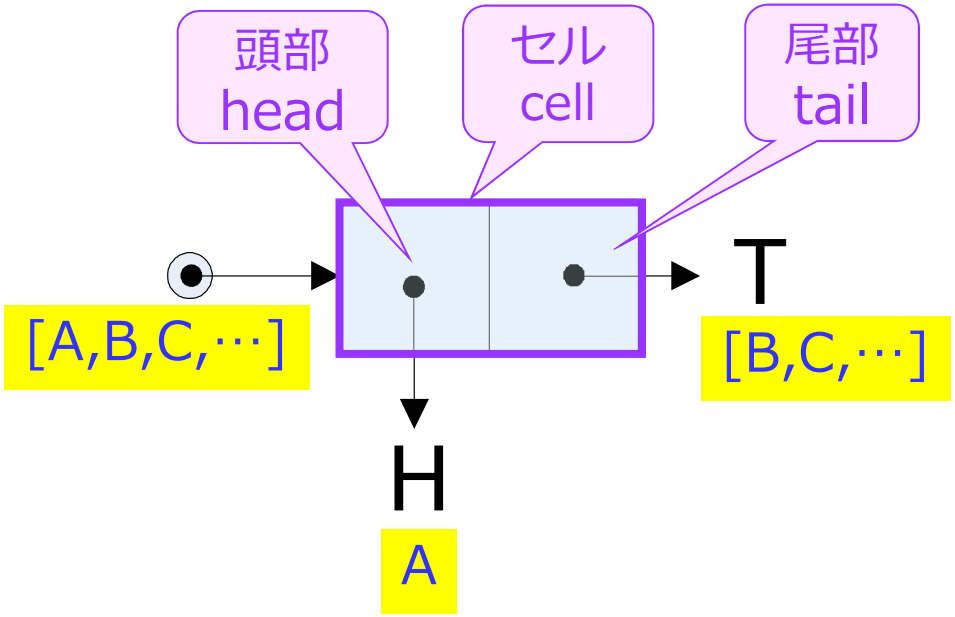
factorial(0) = s(0)
factorial(s(x)) = times(s(x),factorial(x))

times(a,b) は かけ算 a×b.
別途定義してあるとする.

times(a,b) means a×b,
defined elsewhere.

【補足】リスト構造 (1/3)

リスト構造:データの列　[A, B, C, …]
を表現するデータ構造



これを，項　cons(H,T)　で表現

簡易記法：　H:T

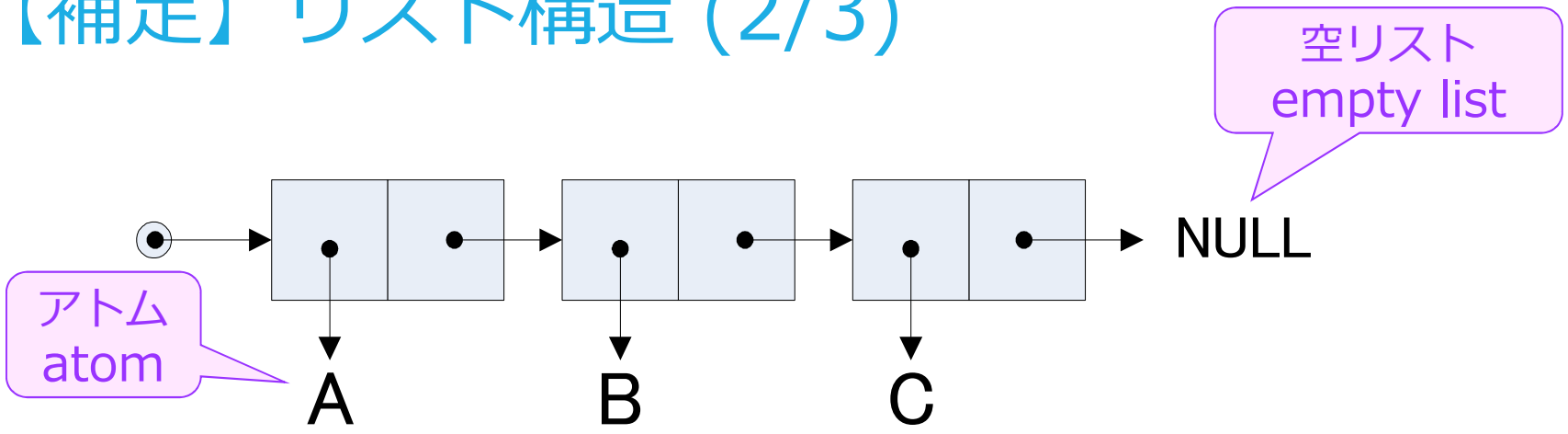
(Supplementary note:
List structure)

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

It is implemented as a cell consisting of two parts: the head for representing the first idem 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 cons(H,T) and abbreviated as H:T.

【補足】リスト構造 (2/3)



A: (B: (C: NULL))

= A: B: C: NULL

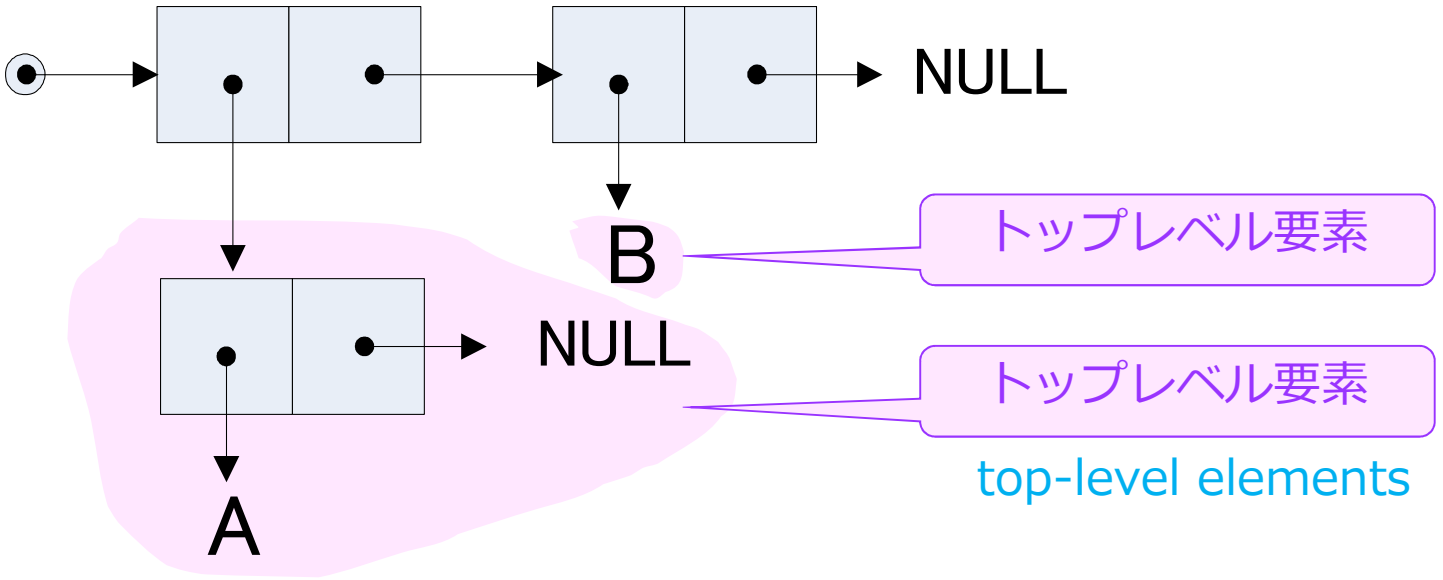
= [A,B,C]

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）

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

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

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*) returns the *n*th smallest prime number.

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

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)

nth(*L*, *n*)
はリスト*L* の*n* 番目の要素

nth(x : y, 0) = x
nth(x : y, s(n)) = nth(y, n)

nth(*L*, *n*)
returns the *n*th
element of the list *L*

The first (*0*th) 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,…]

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

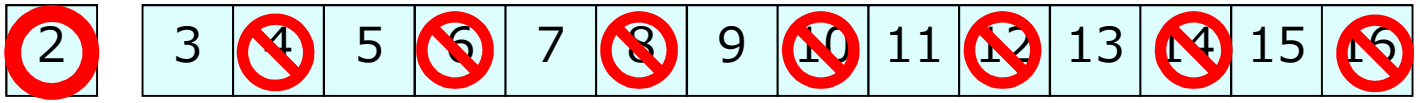
sieve(x : y) = x : sieve(filt(x, y))

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*÷*x*.
(Define them in the Exercise.)

If the first element *y* is divided by *x* (i.e., *y* mod *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≧yならばtrue,
でなければfalse
(3) minus(x,y): x≧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)

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*≧*y*;
or false, otherwise.
(3) minus(*x*, *y*) returns *x*-*y* if *x*≧*y*;
or 0, otherwise.
(4) mod(*x*, *y*) returns the remainder for *x*÷*y*. (Subtract *y* from *x* as long as possible.)

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

eq(*x*, *y*) = if(*x*=*y*, true, false)

is incorrect.

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.