2. 計算可能性入門

計算とは何か？

- 「計算できる」ことと「計算できない」ことの違い
- 「計算」の基本要素(今回)
- 「計算できない」ことの証明…対角線論法(次回)

2.1. 帰納的関数論概観

帰納的関数論(recursive function theory)

① “計算”とは何かについての研究
② 計算不可能性の証明
③ 計算不可能な関数のクラスの構造的研究
④ 他の数学との関連分野

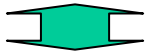
2. 計算可能性入門

① 計算とは何かについての研究

「何をもって計算可能な関数というか？」

・クリーネが定義した帰納的関数(recursive function)
・チューリングが考えたチューリング機械(Turing machine)

➔帰納関数全体＝チューリング機械で計算可能な関数全体



計算可能性の定義…チャーチの提唱（Church’s Thesis)

Chapter 2: Introduction to Computability

2.1. Studies on recursive functions

recursive function theory

(1) studies on what is "computation"

(2) proof of incomputability

(3) structural studies on a class of incomputable functions

(4) related mathematics fields

(1) Studies on what is computation.

"When do we call a function computable?"

- recursive function theory by Kleene
- Turing machine theory by Turing

➔the whole set of recursive functions

＝the whole set of functions computable by Turing machines

Church's Thesis on the definition of computability

② 計算不可能性の証明

- 計算可能性の証明ではプログラムを作ればよい
- 計算不可能性の証明では
  - どんなプログラムも作れないことの証明:
  - 「対角線論法」
  - 「帰納的還元性」

③ 計算不可能な関数のクラスの構造的研究

難しさに応じて階層化されたクラス
→構造的研究

④ 他の数学との関連分野

数理論理学(mathematical logic)など

(2) Proof of incomputability

- Proof of computability is easy: just give a program
- to prove incomputability
  must prove that no program exists : difficult
  proof tools: diagonalization
  recursive reducucibility

(3) Structural studies on a class of incomputable functions

hierarchical class depending of hardness
→structural studies

(4) Related mathematics fields

mathematical logic

2.2. 計算の基本要素

「データ」や「プログラム」を最小限の資源で表現
…対象を絞ることで議論を単純化する

2.2.1. データ表現のための基本要素

データ表現のためには文字列型だけで十分.
構造型などを含め,
すべてのデータ(型)は Σ (={0,1})上の文字列型で代用可能

補題2.1.すべての基本データ型はΣ^*型と構造型で実現できる.

自然数型, 整数型, 実数型, 論理値型, 文字列型

例2.1. 自然数→2進数表記(Σ^*型)

変数宣言 : var n,m: num; ➔ var n_s, m_s: Σ^*;
式など : n:=m*4+n;➔n_s := plus(mult(m_s, 100), n_s);

自然数の基本演算(加減乗除, 大小比較)に
対応するΣ^*上での関数が必要.

2.2. Elements of Computation

String data type suffices to represent data. All data types can including structured type be represented by strings on Σ.

Lemma 2.1: All elementary data types can be represented by Σ^* types and structured type.

types for natural numbers, integers, reals, truth values, strings

Ex. 2.1: Natural number→binary representation（Σ^*type）

declaration: var n,m: num; ➔ var n_s, m_s: Σ^*;

expressions: n:=m*4+n;➔n_s := plus(mult(m_s, 100), n_s);

functions on Σ^* are required for elementary operations on natural numbers (plus, minus, multiply, divide, compare)

```
prog plus(input x, y: Σ*): Σ*;
var a,b,c,d,z: Σ*;                                  c:キャリー, d:和
begin
  c:=0; z:= ε;
  while  x ≠ ε ∨ y ≠ ε  do
    if  x ≠ ε  then a:=tail(x); x:=left(x) else a:=0 end-if;
    if  y ≠ ε  then b:=tail(y); y:=left(y) else b:=0 end-if;
    case (a,b,c) of
      (1,1,1):  c:=1; d:=1;
      (0,0,0):  c:=0; d:=0;
      (0,1,1), (1,0,1), (1,1,0):  c:=1; d:=0;
      (0,0,1), (0,1,0), (1,0,0):  c:=0; d:=1;
    end-case;
    z:=d # z;
  end-while;
  if c=1 then z:=1 # z;
  halt(z);
end.
```

自然数型なので負の数は考えない.

```
prog plus(input x, y: Σ*): Σ*;
var a,b,c,d,z: Σ*;                                  c:carry, d:sum
begin
  c:=0; z:= ε;
  while  x ≠ ε ∨ y ≠ ε  do
     if  x ≠ ε  then a:=tail(x); x:=left(x) else a:=0 end-if;
     if  y ≠ ε  then b:=tail(y); y:=left(y) else b:=0 end-if;
     case (a,b,c) of
        (1,1,1):  c:=1; d:=1;
        (0,0,0):  c:=0; d:=0;
        (0,1,1), (1,0,1), (1,1,0):  c:=1; d:=0;
        (0,0,1), (0,1,0), (1,0,0):  c:=0; d:=1;
     end-case;
     z:=d # z;
  end-while;
  if c=1 then z:=1 # z;
  halt(z);
end.
```

No negative numbers are considered since we only deal with natural numbers

自然数の1進表記

自然数n → 0を n 個並べる

$\lceil n \rceil$: 自然数 n の2進表記　$\lceil 4 \rceil$ ➔ 100

$\overline{n}$: 自然数 n の1進表記　$\overline{4}$ ➔ 0000

例2.2. 一般の文字列(Γ上の文字列)もΣ上の文字列で表現可能.

e.g. 8ビットの2進列でのコード化(ASCIIコードなど)

➔right()を模倣するプログラム right_str

```
prog right_str(input x: Σ*): Σ*;
var y: Σ*;
begin
  y:=right(x); y:=right(y); y:=right(y); y:=right(y);
  y:=right(y); y:=right(y); y:=right(y); y:=right(y);
  halt(y)
end.
```

Unary representation of a natural number

natural number n → sequence of n 0s

$\lceil n \rceil$: binary representation $\lceil 4 \rceil$ ➔ 100

$\overline{n}$: unary representation $\overline{4}$ ➔ 0000

Ex. 2.2: Ordinary letters are also represented by binary strings

e.g. each letter is coded in 8 bits

➔we need a function right_str() to simulate right()

```
prog right_str(input x: Σ*): Σ*;
var y: Σ*;
begin
  y:=right(x); y:=right(y); y:=right(y); y:=right(y);
  y:=right(y); y:=right(y); y:=right(y); y:=right(y);
  halt(y)
end.
```

例2.3. 整数型→ Σ^*型と構造型 （絶対値と符号）

var n_s: record sign: Σ % *n* の符号（負＝0, 正＝1）
abs: Σ^* % *n* の絶対値の2進数表現
end-record;

レコード型, 配列型, ポインタ型などの構造型もΣ^*型で表現可能

補題2.2. すべての構造型はΣ^*型で表現できる.

準備: データペアの扱い方(符号化)
(011, 101)➔100 000 111 111 010 111 000 111 001
(0 1 1 , 1 0 1)

<011, 101>: (011, 101)を表している上記の文字列。つまり
<011, 101> = 100000111111010111000111001

Ex.2.3: integer→ Σ^* type and structure type
（absolute value and sign)

var n_s: record sign: Σ % sign of *n*（negative＝0, positive＝1）
abs: Σ^* % binary representation of the absolute of *n*
end-record;

Structure types such as record type, array type and pointer type are represented by Σ^* type.

Lemma 2.2. All structure types are represented by Σ^* type.

(011, 101)➔100 000 111 111 010 111 000 111 001
(0 1 1 , 1 0 1)

<011, 101>: the above string representing (011, 101), that is,
<011, 101> = 100000111111010111000111001

順序対と対応する2進列との間の相互変換は容易.
実際，次の関数を計算するプログラムが存在する.

各$a,b,c \in \Sigma^*$に対し,
pair(a, b)≡ <a,b>
1st(c) = x, ∃x,y[c=pair(x,y)]のとき,
= ?, その他のとき.
2nd(c) = y, ∃x,y[c=pair(x,y)]のとき,
= ?, その他のとき.

型エラーの場合は
記号?が値となる。

a=011, b=101のとき,
pair(a,b) = <a,b> = <011, 101>
= 100 000 111 111 010 111 000 111 001
c= 100 000 111 111 010 111 000 111 001のとき,
1st(c)=011, 2nd(c)=101

上記の考え方は三つ組，四つ組，... にも拡張可能

Conversion between ordered pairs and corresponding binary sequences is easy.

In fact, there is a program computing the following function.

for each a,b,c $\in \Sigma^*$

pair(a, b) $\equiv$ <a,b>

1st(c) = x, if $\exists$x,y[c=pair(x,y)],

= ?, otherwise.

2nd(c) = y, if $\exists$x,y[c=pair(x,y)],

= ?, otherwise.

（the symbol ? is returned for type error)

for a=011, b=101,

pair(a,b) = <a,b> = <011, 101>

= 100 000 111 111 010 111 000 111 001

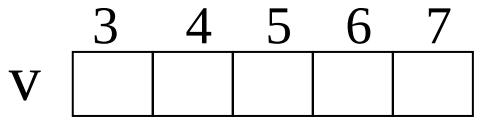
for c= 100 000 111 111 010 111 000 111 001,

1st(c)=011, 2nd(c)=101

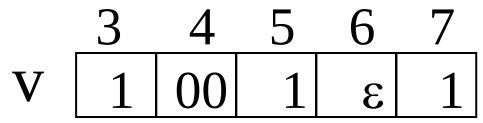
The above idea can be extended into triples, four tuples, etc.

補題の証明： 配列型をΣ^*型で実現する方法
特に，array ... of Σ^*型だけ考えればよい．

例2.4. 変数 v の型がarray[3..7] of Σ^*型だったとして，この変数をΣ^*型の変数 v_s で代用する方法を考える．



5個のΣ^*の元をもつ配列➔文字列 v_s



v_s=<1, 00, 1, ε, 1>
=100 111 010 000 000 010 111 010 010 111 001

```
prog ...
var v: array[3..7] of Σ*;
begin
   :
  v[4] := v[6] # 000;
  :
end.
```

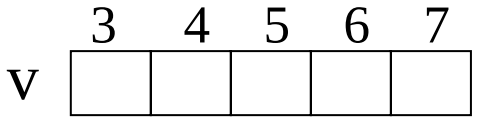
```
prog ...
var v_s, tmp: ;
begin
  v_s:=create(7-3+1);
  :
  tmp:=get(v_s, 6-3+1);
  tmp:=tmp # 000;
  v_s:=put(v_s, 4-3+1, tmp);
  :
end.
```

< ε, ε, ε, ε, ε >
配列の初期化

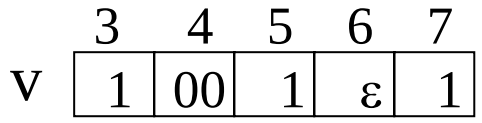
Proof of the lemma: realizing an array by a Σ^*type variable

It suffices to consider an array ... of Σ^*.

Ex.2.4: Suppose a variable v is of type array[3..7] of Σ^*. How to represent this variable by a variable v_s of type Σ^*.



array of 5 elements in Σ^*➔a string v_s



v_s=<1, 00, 1, ε, 1>

=100 111 010 000 000 010 111 010 010 111 001

```
prog ...
var v: array[3..7] of Σ*;
begin
   :
  v[4] := v[6] # 000;
   :
end.
```

```
prog ...
var v_s, tmp: ;
begin
   v_s:=create(7-3+1);
   :
   tmp:=get(v_s, 6-3+1);
   tmp:=tmp # 000;
   v_s:=put(v_s, 4-3+1, tmp);
   :
end.
```

< ε, ε, ε, ε, ε > initialization of an array

定理2.3. われわれのプログラミング言語のすべてのデータ型とその上の基本演算はΣ^*型とその上の基本演算だけで実現できる.

「われわれのコード化法」

$\lceil x \rceil$: データ x を表すΣ^*の元(x のコード)

$\lfloor w \rfloor$: Σ^*の元 w が表しているデータ

例2.5. 自然数, 文字列, 整数などのコード化法は前述の通り.

述語の真偽値, あるいはBoolean型のデータは次のようにコード化する.

$$\lceil true \rceil \equiv 1, \quad \lceil false \rceil \equiv 0$$

例2.6. プログラムも(改行コード入りの)文字列と見なしてコード化.

```
prog A ...        A  = 0111000  01110010  01101111 ....
begin                        p         r         o ....
  :
end.                    01100101  01101110  00101110 ...
                             e         n         d
```

もっと使いやすいコード化もあるが, 当面はこれで.

Theorem 2.3. All the data types and elementary operations in our programming language can be realized on Σ^*.

"Our encoding method"

$\lceil x \rceil$: an element of Σ^* representing a data x (a code of x)

$\lfloor w \rfloor$: a data represented by an element w of Σ^*

Ex.2.5. Natural numbers, strings, integers are coded as before

Truth value of a predicate or data of Boolean type are coded:

$$\lceil true \rceil \equiv 1, \quad \lceil false \rceil \equiv 0$$

Ex.2.6. Programs are also coded by considering them as strings

```
prog A ...        A  = 0111000  01110010  01101111 ....
  begin                     p         r          o ....
    :
  end.                01100101  01101110  00101110 ...
                            e         n          d
```

We could use a different coding method, but ...

2.2. 計算の基本要素

「データ」や「プログラム」を最小限の資源で表現
…対象を絞ることで議論を単純化する

2.2.2. 制御機構のための基本要素

補題2.4. 関数プログラム(関数定義と関数呼び出し)は,
すべてif文とgoto文によって実現できる.

（略証）

フローチャート → if文とgoto文
再帰呼び出し → スタックを用いて書きなおす

補題2.5. すべての制御構造はif文とgoto文によって実現できる.

定理2.6. すべての制御構造はif文とwhile文によって実現できる.
（例に基づいて証明）

プログラムの「標準形」

2.2.2. Elements for Control Mechanism

Lemma 2.4: A function (definition and call of function) can be implemented by if and goto statements.

（Proof sketch）

flowchart → if statement and goto statement

recursive call → can be rewritten using a stack

Lemma 2.5. All the control mechanisms can be realized by if and goto statements.

Theorem 2.6. All the control structures can be realized by if and while statements.

（Proof based on examples）

```
% xが0*かどうかを判定するプログラム
prog A(input x: Σ*): Σ*;
label LOOP;  var a: Σ*;
begin
LOOP: if x= ε then halt(1) end-if;
          a:=head(x); x:=right(x);
          if a=1 then halt(0) else goto LOOP end-if
end.
```

これを次のように変形する.

(1) プログラムの各行は次のいずれか.
  (a) 代入文とgoto文
  (b) if Σ*上の比較 then goto ... else goto ... end-if
  (c) halt（変数）

(2) プログラム本体の各行には，L1から始まり，L2, L3,...と順にラベルづけされている.

(3) ただし，(c)の形の行はプログラムの最後に1箇所しか現れず，それはL0とラベル付けされている.

```
% program to determine whether x is 0* or not
prog A(input x: Σ*): Σ*;
label LOOP;  var a: Σ*;
begin
LOOP: if x= ε then halt(1) end-if;
        a:=head(x); x:=right(x);
        if a=1 then halt(0) else goto LOOP end-if
end.
```

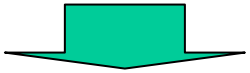
Convert it as follows.

(1) Each line of a program is one of the followings:
   (a) substitution, goto statement
   (b) if comparison on Σ^* then goto ... else goto ... end-if
   (c) halt（variable）

(2) Each line in the program body is labeled as L1, L2, ...

(3) The line of the form (c) above appears only once in the program and it is labeled as L0.

```
prog A(input x: Σ*): Σ*;
label LOOP;  var a: Σ*;
begin
LOOP: if x= ε then halt(1) end-if;
        a:=head(x); x:=right(x);
        if a=1 then halt(0) else goto LOOP end-if
end.
```



```
prog B(input x: Σ*): Σ*;
label L0, L1, L2, L3, L4, L5, L6;
var a,c: Σ*;
begin
L1: if x= ε then goto L5 else goto L2 end-if;
L2: a:=head(x); goto L3;
L3: x:=right(x); goto L4;
L4: if a=1 then goto L6 else goto L1 end-if;
L5: c:=1; goto L0;
L6: c:=0; goto L0;
L0: halt(c)
end.
```

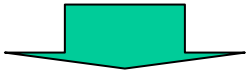
(3-2) goto 文で次に実行する行に移動

(3-1) 通常の処理＋次に実行する行を決める

(2) haltの値を設定

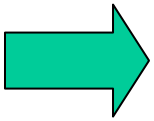
(1) halt文を追加

```
prog A(input x: Σ*): Σ*;
label LOOP;  var a: Σ*;
begin
LOOP: if x= ε then halt(1) end-if;
        a:=head(x); x:=right(x);
        if a=1 then halt(0) else goto LOOP end-if
end.
```



```
prog B(input x: Σ*): Σ*;
label L0, L1, L2, L3, L4, L5, L6;
var a,c: Σ*;
begin
L1: if x= ε then goto L5 else goto L2 end-if;
L2: a:=head(x); goto L3;
L3: x:=right(x); goto L4;
L4: if a=1 then goto L6 else goto L1 end-if;
L5: c:=1; goto L0;
L6: c:=0; goto L0;
L0: halt(c)
end.
```

```
prog B(input x: Σ*): Σ*;
label L0, L1, L2, L3, L4, L5, L6;
var a,c: Σ*;
begin
L1: if x= ε then goto L5 else goto L2 end-if;
L2: a:=head(x); goto L3;
L3: x:=right(x); goto L4;
L4: if a=1 then goto L6 else goto L1 end-if;
L5: c:=1; goto L0;
L6: c:=0; goto L0;
L0: halt(c)
end.
```

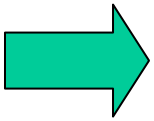


```
prog C(input x: Σ*): Σ*;
var pc: num; a,c:Σ*;
begin
  pc:=1;
  while pc != 0 do
     case pc of
     1:  if x= ε then pc:=5 else pc:=2 end-if;
     2: a:=head(x); pc:=3;
     3: x:=right(x); pc:=4;
     4: if a=1 then pc:=6 else pc:=1 end-if;
     5: c:=1; pc:=0;
     6: c:=0; pc:=0;
     end-case;
  end-while;
  halt(c)
end.
```

goto Lk ➔ pc:=k;

ただし，case文は
実際にはif文の
組み合わせで実現.

```
prog B(input x: Σ*): Σ*;
label L0, L1, L2, L3, L4, L5, L6;
var a,c: Σ*;
begin
L1: if x= ε then goto L5 else goto L2 end-if;
L2: a:=head(x); goto L3;
L3: x:=right(x); goto L4;
L4: if a=1 then goto L6 else goto L1 end-if;
L5: c:=1; goto L0;
L6: c:=0; goto L0;
L0: halt(c)
end.
```



```
prog C(input x: Σ*): Σ*;
var pc: num; a,c:Σ*;
begin
  pc:=1;
  while pc != 0 do
     case pc of
     1:  if x= ε then pc:=5 else pc:=2 end-if;
     2: a:=head(x); pc:=3;
     3: x:=right(x); pc:=4;
     4: if a=1 then pc:=6 else pc:=1 end-if;
     5: c:=1; pc:=0;
     6: c:=0; pc:=0;
     end-case;
  end-while;
  halt(c)
end.
```

goto Lk ➔ pc:=k;

Remark: case statement is realized by combination of if and goto

```
case pc of
1: if x= ε then pc:=5 else pc:=2 end-if;
2: a:=head(x); pc:=3;
```

```
if pc=1 then
    if x= ε then pc:=5 else pc:=2 end-if;
else if pc=2 then
   a:=head(x); pc:=3;
end-if;
```

```
case pc of
1: if x= ε then pc:=5 else pc:=2 end-if;
2: a:=head(x); pc:=3;
```

```
if pc=1 then
    if x= ε then pc:=5 else pc:=2 end-if;
else if pc=2 then
   a:=head(x); pc:=3;
end-if;
```

単純プログラム: 下の要素のみで構成されるプログラム

データ型: Σ上の文字列型(Σ型, Σ*型)

基本演算: 文字列型の基本演算

実行文: 代入文, if文(case文), while文, halt文

定理2.7. どんなプログラムもそれと同値な単純プログラムに書換えることができる. しかも次のような標準形プログラムに書き直せる

```
prog プログラム名(input ...) ;
var pc: Σ*; ... Σ; ... Σ*; %pcの値は自然数の2進表記
begin
  pc:=1;
  while pc != 0 do
      case pc of
      1: (文) ;
      2: (文) ;
        :
      k: (文) ;
     end-case
   end-while;
  halt(c)
end.
```

各(文)の形は

- if 比較文 then pc:=k1 else pc:=k2 end-if
- 代入文 ; pc:=k;

のいずれか

Simple program: a program consisting only of the following elements.

data type: string type on Σ（Σ type, Σ^* type）

elementary operations: elementary operations on strings

execution statements: substitution, if (case), while, halt

Theorem 2.7 Any program can be rewritten into its equivalent simple program of the following form:

```
prog  Program name(input ...) ;
var pc: Σ*; ... Σ; ... Σ*; % value of pc is a binary representation of an integer
begin
  pc:=1;
  while pc != 0 do
      case pc of
      1:（statement）;
      2:（statement）;
        :
      K:（statement）;
     end-case
    end-while;
  halt(c)
end.
```

each statement is one of the two:

- if comparison then pc:=k1 else pc:=k2 end-if
- substitution；pc:=k;

定理2.8. すべての計算可能関数に対し，
それを計算する標準形プログラムが存在する.

プログラムカウンタの働きを考えてみよう.

更なる制約（テキスト101ページ）

「各文は高々定数時間で実行できるものだけ」

u, u’: Σ型の変数，　v,v’: Σ*型の変数

c: Σ型の定数，　s: Σ*型の定数

（代入文）

(1) u:=c;　(2) u:=u’;

(3) u:=head(v);　(4) u:=tail(v);

(5) v:=s;　~~(6) v:=v’;~~ ?

(7) v:= right(v);　(8) v:=left(v);

(9) v:=u # v;　(10) v:=v # u;

（比較文）

(11) u=c　(12) v=s

Theorem2.8 For every computable function, there is a program in the standard form.

Consider a behavior of program counter.

Further constraints（refer to 101 page of the textbook）

"each statement must be implemented in constant time"

u, u': variables of Σ type, v,v': variables of Σ^* type

c: constant of Σ type, s: constant of Σ^* type

（Substitution）

（1）u:=c; (2) u:=u';

(3) u:=head(v); (4) u:=tail(v);

(5) v:=s; ~~(6) v:=v';~~ **?**

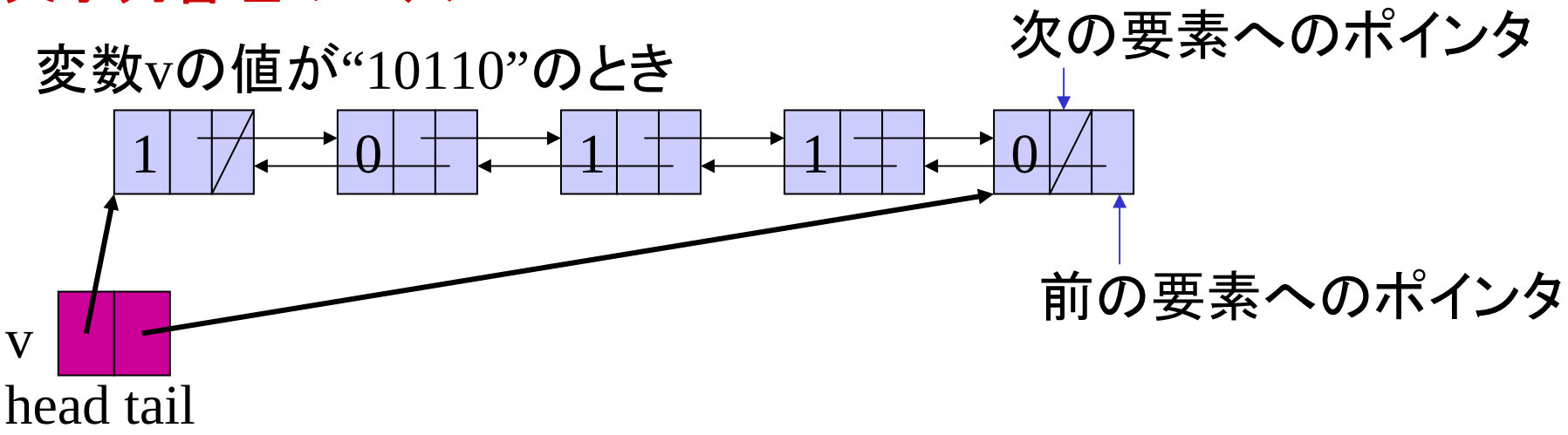
(7) v:= right(v); (8) v:=left(v);

(9) v:=u # v; (10) v:=v # u;

（Comparison）

(11) u=c (12) v=s

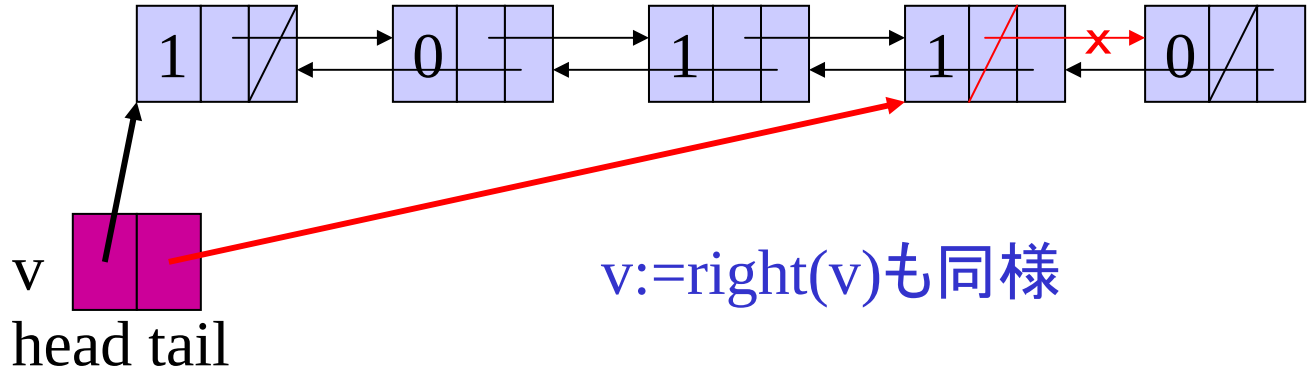
文字列管理のモデル



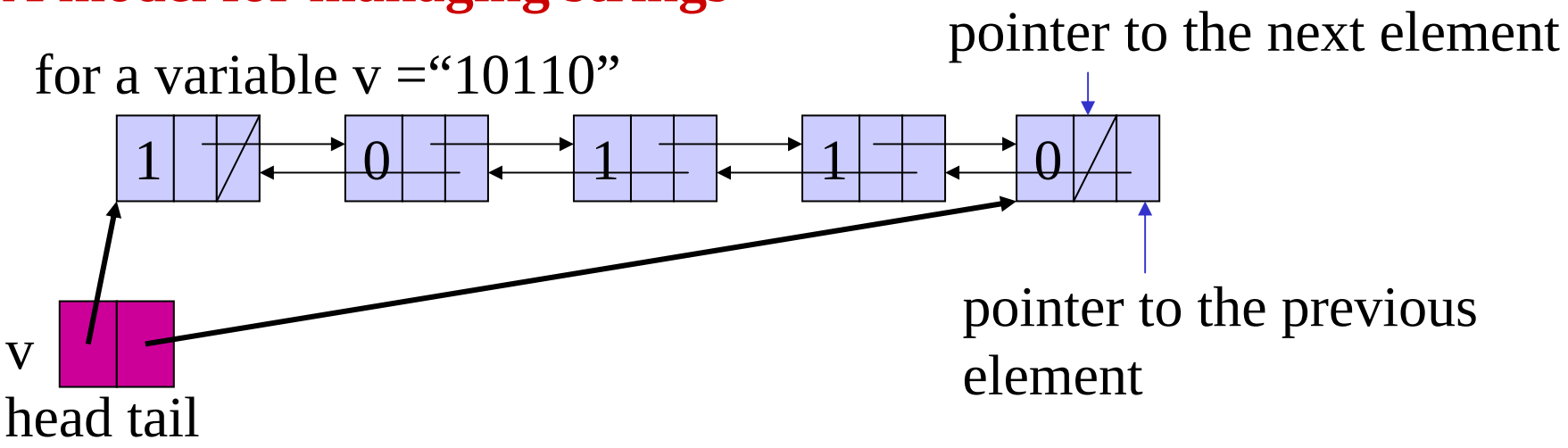
このようにしておくと，head(v), tail(v)の値はすぐに得られる.

v:=left(v); の計算

tail(v)の場所を求め，そこから前へのポインタをたどり，tail(v)のポインタ，および新たなtail(v)のポインタを書き換える



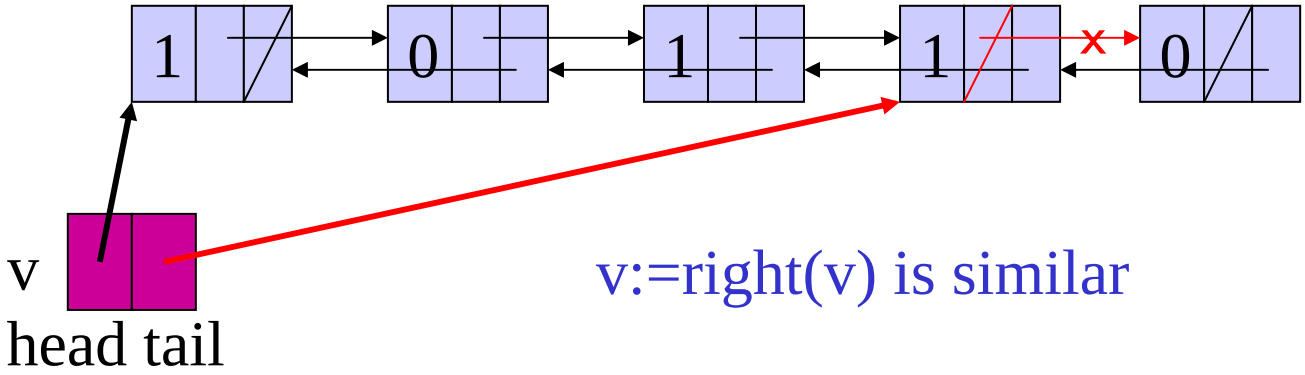
A model for managing strings



Using the points, head(v) and tail(v) are easily computed.

implementation of v:=left(v);

Find tail(v), and then trace back to its previous element to rewrite the pointer to tail(v) and modify the last pointer.



例4.1. Σ^* 型の変数 x, y の値が等しいかどうかを調べるプログラム

（考え方）

x, yの先頭の文字を比較

異なる➔reject (halt(0))

一致➔先頭文字を取り除いて繰り返し

x,yが共にε ➔accept (halt(1))

```
prog EQ_str(input x,y): var pc, out: Σ; a,b: Σ;
begin
  pc:=1;
  while pc != 0 do
    case pc of
    1: if x= ε then pc:=10 else pc:=2 end-if;
    2: if y= ε then pc:=12 else pc:=3 end-if;
    3: a:=head(x); pc:=4;
    4: b:=head(y); pc:=5;
    5: if a=0 then pc:=6 else pc:=7 end-if;
    6: if b=0 then pc:=8 else pc:=12 end-if;
    7: if b=1 then pc:=8 else pc:=12 end-if;
    8: x:=right(x); pc:=9;
    9: y:=right(y); pc:=1;
   10:  if y= ε then pc:=11 else
                pc:=12 end-if;
   11:  out:=1; pc:=0;
   12:  out:=0; pc:=0;
   end-case;
  end-while;
  halt(out)
end.
```

Ex.4.1 Program for checking whether variables *x* and *y* of Σ* type are equal to each other.

（Idea）

Compare the first letters of *x* and *y*

distinct➔reject (halt(0))

equal➔remove the first letters and iterate.

both of *x* and *y* become ε ➔accept (halt(1))

```
prog EQ_str(input x,y): var pc, out: Σ; a,b: Σ;
begin
  pc:=1;
  while pc != 0 do
    case pc of
    1: if x= ε then pc:=10 else pc:=2 end-if;
    2: if y= ε then pc:=12 else pc:=3 end-if;
    3: a:=head(x); pc:=4;
    4: b:=head(y); pc:=5;
    5: if a=0 then pc:=6 else pc:=7 end-if;
    6: if b=0 then pc:=8 else pc:=12 end-if;
    7: if b=1 then pc:=8 else pc:=12 end-if;
    8: x:=right(x); pc:=9;
    9: y:=right(y); pc:=1;
   10:  if y= ε then pc:=11 else
                pc:=12 end-if;
   11:  out:=1; pc:=0;
   12:  out:=0; pc:=0;
    end-case;
  end-while;
  halt(out)
end.
```