

1/19

## 2. 計算可能性入門

計算とは何か？

- 「計算できる」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)

- 「計算」とは何かについての研究
- 計算不可能性の証明
- 計算不可能な関数のクラスの構造的な研究
- 他の数学との関連分野

1/19

## Chapter 2: Introduction to Computability

What "Computation" is...

- Difference between "computable" and "incomputable"
  - Basic factor of a "computation" (Today)
  - Proof of "incomputable"... diagonalization (Next)

2.1. Studies on recursive functions

recursive function theory

- studies on what is "computation"
- proof of incomputability
- structural studies on a class of incomputable functions
- related mathematics fields

2/19

## 2. 計算可能性入門

① 計算とは何かについての研究

「何をもち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)

→ 帰納的関数全体 = 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)

2/19

## Chapter 2: Introduction to Computability

(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"

3/19

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」

難しい

③ 計算不可能な関数のクラスの構造的な研究

難しさに応じて階層化されたクラス  
→ 構造的な研究

④ 他の数学との関連分野

数理論理学(mathematical logic)など

3/19

(2) Proof of incomputability

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

Difficult!

(3) Structural studies on a class of incomputable functions

hierarchical class depending of hardness  
→ structural studies

(4) Related mathematics fields

mathematical logic

4/19

**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  $\Sigma (= \{0,1\})$  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  $\Sigma^*$  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  $\rightarrow$  2進数表記 ( $\Sigma^*$ 型)  
変数宣言:  $\text{var } n,m:\text{num}; \rightarrow \text{var } n\_s, m\_s: \Sigma^*$ ;  
式など:  $n:=m*4+n; \rightarrow n\_s := \text{plus}(\text{mult}(m\_s, 100), n\_s)$ ;

自然数の基本演算(加減乗除, 大小比較)に対応する  $\Sigma^*$  上での関数が必要。

4/19

**2.2. Elements of Computation**

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

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

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

**Ex. 2.1: Natural number  $\rightarrow$  binary representation ( $\Sigma^*$ type)**  
declaration:  $\text{var } n,m:\text{num}; \rightarrow \text{var } n\_s, m\_s: \Sigma^*$ ;  
expressions:  $n:=m*4+n; \rightarrow n\_s := \text{plus}(\text{mult}(m\_s, 100), n\_s)$ ;  
functions on  $\Sigma^*$  are required for elementary operations on natural numbers (plus, minus, multiply, divide, compare)

5/19

```

prog plus(input x, y:  $\Sigma^*$ ):  $\Sigma^*$ ;
var a,b,c,d,z:  $\Sigma^*$ ;          c:キャリー, d:和
begin
  c:=0; z:= $\epsilon$ ;
  while  $x \neq \epsilon \vee y \neq \epsilon$  do
    if  $x \neq \epsilon$  then a:=tail(x); x:=left(x) else a:=0 end-if;
    if  $y \neq \epsilon$  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;          自然数型なので負の数は考えない。
  end-while;
  if c=1 then z:=1 # z;
  halt(z);
end.

```

5/19

```

prog plus(input x, y:  $\Sigma^*$ ):  $\Sigma^*$ ;
var a,b,c,d,z:  $\Sigma^*$ ;          c:carry, d:sum
begin
  c:=0; z:= $\epsilon$ ;
  while  $x \neq \epsilon \vee y \neq \epsilon$  do
    if  $x \neq \epsilon$  then a:=tail(x); x:=left(x) else a:=0 end-if;
    if  $y \neq \epsilon$  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;          No negative numbers are considered since we only deal with natural numbers
  end-while;
  if c=1 then z:=1 # z;
  halt(z);
end.

```

6/19

**自然数の1進表記**  
自然数  $n \rightarrow 0$  を  $n$  個並べる  
 $\lceil n \rceil$ : 自然数  $n$  の2進表記  $\lceil 4 \rceil \rightarrow 100$   
 $\bar{n}$ : 自然数  $n$  の1進表記  $\bar{4} \rightarrow 0000$

例2.2. 一般の文字列 ( $\Gamma$  上の文字列) も  $\Sigma$  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)

$\rightarrow$  right()を模倣するプログラム right\_str

```

prog right_str(input x:  $\Sigma^*$ ):  $\Sigma^*$ ;
var y:  $\Sigma^*$ ;
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.

```

6/19

**Unary representation of a natural number**  
natural number  $n \rightarrow$  sequence of  $n$  0s  
 $\lceil n \rceil$ : binary representation  $\lceil 4 \rceil \rightarrow 100$   
 $\bar{n}$ : unary representation  $\bar{4} \rightarrow 0000$

Ex. 2.2: Ordinary letters are also represented by binary strings  
e.g. each letter is coded in 8 bits

$\rightarrow$  we need a function right\_str() to simulate right()

```

prog right_str(input x:  $\Sigma^*$ ):  $\Sigma^*$ ;
var y:  $\Sigma^*$ ;
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.

```

7/19

**例2.3. 整数型  $\rightarrow \Sigma^*$  型と構造型 (絶対値と符号)**

```

var n_s: record sign:  $\Sigma$  % nの符号(負=0, 正=1)
      abs:  $\Sigma^*$  % 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  $\Sigma^*$  型で表現可能

**補題2.2. すべての構造型は  $\Sigma^*$  型で表現できる.**

準備: データペアの扱い方(符号化)

(011, 101)  $\rightarrow$  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

7/19

**Ex.2.3: integer  $\rightarrow \Sigma^*$  type and structure type**  
 (absolute value and sign)

```

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

```

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

**Lemma 2.2. All structure types are represented by  $\Sigma^*$  type.**

(011, 101)  $\rightarrow$  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

8/19

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,  
 $\text{pair}(a, b) \equiv \langle a, b \rangle$   
 $\text{1st}(c) = x$ , if  $\exists x, y [c = \text{pair}(x, y)]$  のとき,  
           = ?, 其他のとき.  
 $\text{2nd}(c) = y$ , if  $\exists x, y [c = \text{pair}(x, y)]$  のとき,  
           = ?, 其他のとき.

$a=011, b=101$  のとき,  
 $\text{pair}(a, b) = \langle a, b \rangle = \langle 011, 101 \rangle$   
                   = 100 000 111 111 010 111 000 111 001  
 $c = 100 000 111 111 010 111 000 111 001$  のとき,  
 $\text{1st}(c) = 011, \text{2nd}(c) = 101$

上記の考え方は三つ組, 四つ組, ... にも拡張可能

8/19

Conversion between ordered pairs and corresponding binary sequences is easy.

In fact, there is a program computing the following function.  
 for each  $a, b, c \in \Sigma^*$   
 $\text{pair}(a, b) \equiv \langle a, b \rangle$   
 $\text{1st}(c) = x$ , if  $\exists x, y [c = \text{pair}(x, y)]$ ,  
           = ?, otherwise.  
 $\text{2nd}(c) = y$ , if  $\exists x, y [c = \text{pair}(x, y)]$ ,  
           = ?, otherwise.  
 (the symbol ? is returned for type error)  
 for  $a=011, b=101$ ,  
 $\text{pair}(a, b) = \langle a, b \rangle = \langle 011, 101 \rangle$   
                   = 100 000 111 111 010 111 000 111 001  
 for  $c = 100 000 111 111 010 111 000 111 001$ ,  
 $\text{1st}(c) = 011, \text{2nd}(c) = 101$   
 The above idea can be extended into triples, four tuples, etc.

9/19

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

**例2.4. 変数  $v$  の型が  $\text{array}[3..7]$  of  $\Sigma^*$  型だったとして, この変数を  $\Sigma^*$  型の変数  $v\_s$  で代用する方法を考える。**

$v$ 

3	4	5	6	7

 5個の  $\Sigma^*$  の元をもつ配列  $\rightarrow$  文字列  $v\_s$

$v$ 

3	4	5	6	7
1	00	1	ε	1

 $v\_s = \langle 1, 00, 1, \epsilon, 1 \rangle$   
 = 100 111 010 000 000 010 111 010 010 111 001

```

prog ...
var v: array[3..7] of  $\Sigma^*$ ;
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.

```

$\langle \epsilon, \epsilon, \epsilon, \epsilon, \epsilon \rangle$   
 initialization of an array

9/19

**Proof of the lemma: realizing an array by a  $\Sigma^*$  type variable**  
 It suffices to consider an array ... of  $\Sigma^*$ .

**Ex.2.4: Suppose a variable  $v$  is of type  $\text{array}[3..7]$  of  $\Sigma^*$ . How to represent this variable by a variable  $v\_s$  of type  $\Sigma^*$ .**

$v$ 

3	4	5	6	7

 array of 5 elements in  $\Sigma^* \rightarrow$  a string  $v\_s$

$v$ 

3	4	5	6	7
1	00	1	ε	1

 $v\_s = \langle 1, 00, 1, \epsilon, 1 \rangle$   
 = 100 111 010 000 000 010 111 010 010 111 001

```

prog ...
var v: array[3..7] of  $\Sigma^*$ ;
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.

```

$\langle \epsilon, \epsilon, \epsilon, \epsilon, \epsilon \rangle$   
 initialization of an array

10/19

**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 $\Sigma^*$ 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」

$[x]$ : データ  $x$  を表す  $\Sigma^*$  の元 ( $x$  のコード)  
 $[w]$ :  $\Sigma^*$  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.  
 $[true] \equiv 1, [false] \equiv 0$

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 ...	もっと使いやすい
:		コード化もあるが,
end.	01100101 01101110 00101110 ...	当面はこれで.
	e n d	

10/19

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

“Our encoding method”

$[x]$ : an element of  $\Sigma^*$  representing a data  $x$  (a code of  $x$ )  
 $[w]$ : a data represented by an element  $w$  of  $\Sigma^*$

Ex.2.5. Natural numbers, strings, integers are coded as before  
 Truth value of a predicate or data of Boolean type are coded:  
 $[true] \equiv 1, [false] \equiv 0$

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

prog A ...	A = 0111000 01110010 01101111 ...	
begin	p r o ...	
:		We could use a
end.	01100101 01101110 00101110 ...	different coding
	e n d	method, but ...

11/19

**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」**

11/19

**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)

**“Standard Form” of a program**

12/19

```
% xが0*かどうかを判定するプログラム
prog A(input x:  $\Sigma^*$ ):  $\Sigma^*$ ;
label LOOP; var a:  $\Sigma^*$ ;
begin
LOOP: if x=  $\epsilon$  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.

- プログラムの各行は次のいずれか.
  - 代入文とgoto文
  - if  $\Sigma^*$ 上の比較 then goto ... else goto ... end-if
  - halt (変数)
- 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.
- 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.

12/19

```
% program to determine whether x is 0* or not
prog A(input x:  $\Sigma^*$ ):  $\Sigma^*$ ;
label LOOP; var a:  $\Sigma^*$ ;
begin
LOOP: if x=  $\epsilon$  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.

- Each line of a program is one of the followings:
  - substitution, goto statement
  - if comparison on  $\Sigma^*$  then goto ... else goto ... end-if
  - halt (variable)
- Each line in the program body is labeled as L1, L2, ...
- The line of the form (c) above appears only once in the program and it is labeled as L0.

13/19

```

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.

```

(1) halt文を追加

(2) haltの値を設定

(3-1) 通常の処理+次に実行する行を決める

(3-2) goto文で次に実行する行に移動

13/19

```

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.

```

(1) Add halt

(2) Set values of halt

(3-1) Usual process + goto next line

(3-2) Jump to the next line indicated by goto

14/19

```

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.

```

Program Counter

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。

14/19

```

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.

```

Program Counter

goto Lk → pc:=k;

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

15/19

```

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

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;
end-case;

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;

```

15/19

```

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

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;
end-case;

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;

```

16/19

**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:  $\Sigma$ 上の文字列型( $\Sigma$ 型,  $\Sigma^*$ 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等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:  $\Sigma^*$ ; ...  $\Sigma$ ; ...  $\Sigma^*$ ; %pcの値は自然数の2進表記
begin
  pc:=1;
  while pc != 0 do
    case pc of
      1: (文);           各(文)の形は
                        ・if 比較文 then pc:=k1 else pc:=k2 end-if
      2: (文);           ・代入文;pc:=k;
                        ;
      k: (文);           のいずれか
    end-case
  end-while;
  halt(c)
end.
  
```

16/19

**Simple program:** a program consisting only of the following elements.  
 data type: string type on  $\Sigma$  ( $\Sigma$  type,  $\Sigma^*$  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:  $\Sigma^*$ ; ...  $\Sigma$ ; ...  $\Sigma^*$ ; % value of pc is a binary representation of an integer
begin
  pc:=1;
  while pc != 0 do
    case pc of
      1: (statement);   each statement is one of the two:
                        ・if comparison then pc:=k1 else pc:=k2 end-if
      2: (statement);   ・substitution;pc:=k;
                        ;
      k: (statement);
    end-case
  end-while;
  halt(c)
end.
  
```

17/19

**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」  
 $u, u'$ :  $\Sigma$ 型の変数,  $v, v'$ :  $\Sigma^*$ 型の変数  
 $c$ :  $\Sigma$ 型の定数,  $s$ :  $\Sigma^*$ 型の定数

(代入文)  
 (1)  $u:=c$ ; (2)  $u:=u'$ ;  
 (3)  $u:=\text{head}(v)$ ; (4)  $u:=\text{tail}(v)$ ;  
 (5)  $v:=s$ ; (6)  $v:=v' \leftarrow ?$   
 (7)  $v:=\text{right}(v)$ ; (8)  $v:=\text{left}(v)$ ;  
 (9)  $v:=u \# v$ ; (10)  $v:=v \# u$ ;

(比較文)  
 (11)  $u=c$  (12)  $v=s$

17/19

**Theorem 2.8** 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  $\Sigma$  type,  $v, v'$ : variables of  $\Sigma^*$  type  
 $c$ : constant of  $\Sigma$  type,  $s$ : constant of  $\Sigma^*$  type

(Substitution)  
 (1)  $u:=c$ ; (2)  $u:=u'$ ;  
 (3)  $u:=\text{head}(v)$ ; (4)  $u:=\text{tail}(v)$ ;  
 (5)  $v:=s$ ; (6)  $v:=v' \leftarrow ?$   
 (7)  $v:=\text{right}(v)$ ; (8)  $v:=\text{left}(v)$ ;  
 (9)  $v:=u \# v$ ; (10)  $v:=v \# u$ ;

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

18/19

**文字列管理のモデル**

変数 $v$ の値が"10110"のとき

次の要素へのポインタ  
 前の要素へのポインタ

このようにしておくと、 $\text{head}(v)$ ,  $\text{tail}(v)$ の値はすぐに得られる。

$v:=\text{left}(v)$ の計算  
 $\text{tail}(v)$ の場所を求め、そこから前へのポインタをたどり、 $\text{tail}(v)$ のポインタ、および新たな $\text{tail}(v)$ のポインタを書き換える

$v:=\text{right}(v)$ も同様

18/19

**A model for managing strings**

for a variable  $v = "10110"$

pointer to the next element  
 pointer to the previous element

Using the points,  $\text{head}(v)$  and  $\text{tail}(v)$  are easily computed.

implementation of  $v:=\text{left}(v)$ ;  
 Find  $\text{tail}(v)$ , and then trace back to its previous element to rewrite the pointer to  $\text{tail}(v)$  and modify the last pointer.

$v:=\text{right}(v)$  is similar

例4.1.  $\Sigma^*$ 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

19/19

(考え方)

$x, y$  の先頭の文字を比較  
異なる  $\rightarrow$  reject (halt(0))  
一致  $\rightarrow$  先頭文字を取り除いて繰り返し  
 $x, y$  が共に  $\varepsilon \rightarrow$  accept (halt(1))

```
prog EQ_str(input x,y): var pc, out:  $\Sigma$ ; a,b:  $\Sigma$ ;  
begin  
  pc:=1;  
  while pc != 0 do  
    case pc of  
      1: if x=  $\varepsilon$  then pc:=10 else pc:=2 end-if;  
      2: if y=  $\varepsilon$  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=  $\varepsilon$  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  $\Sigma^*$  type are equal to each other.

19/19

(Idea)

Compare the first letters of  $x$  and  $y$   
distinct  $\rightarrow$  reject (halt(0))  
equal  $\rightarrow$  remove the first letters and iterate.  
both of  $x$  and  $y$  become  $\varepsilon \rightarrow$  accept (halt(1))

```
prog EQ_str(input x,y): var pc, out:  $\Sigma$ ; a,b:  $\Sigma$ ;  
begin  
  pc:=1;  
  while pc != 0 do  
    case pc of  
      1: if x=  $\varepsilon$  then pc:=10 else pc:=2 end-if;  
      2: if y=  $\varepsilon$  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=  $\varepsilon$  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.
```