計算理論　第 3 回
チャーチ・チューリングの定立

岡本 吉央
okamotoy@uec.ac.jp

電気通信大学

2020 年 10 月 15 日

最終更新：2020 年 10 月 19 日　08:31

概要

この講義の主題

計算理論 (Theory of Computation)

- 計算可能性理論　(Computability Theory)
- 計算複雑性理論 (計算量理論)　(Complexity Theory)

講義の進め方

- 前半：計算可能性理論 (担当：岡本)
- 後半：計算複雑性理論 (担当：垂井先生)

スケジュール 前半 (予定)

1. 計算とは何か？　(10/1)
2. 計算モデル　(10/8)
3. チャーチ・チューリングの定立　(10/15)
- ⋆ 休み (体育祭)　(10/22)
4. コード化　(10/29)
5. 計算可能性　(11/5)
6. 停止性問題　(11/12)
7. 再帰定理　(11/19)
8. 前半のまとめ　(11/26)

注意：予定の変更もありうる

計算モデル

計算モデルとは？ (直感的な定義)

「計算主体」を数学的に抽象化したもの

代表的な計算モデル

- チューリング機械
- ランダム・アクセス機械
- カウンタ機械
- ポインタ機械
- タグ・システム
- 帰納的関数 (μ 再帰関数)
- ラムダ計算
- マルコフ・アルゴリズム

この講義 (の前半) では

- ある「単純化したプログラミング言語」を計算モデルとして用いる
  - WHILE プログラム　(前回紹介した)
  - LOOP プログラム　(今回紹介する)
  - GOTO プログラム　(今回紹介する)

目次

計算可能性の等価性 (1)

計算モデルとは？ (直感的な定義)

「計算主体」を数学的に抽象化したもの

代表的な計算モデル

- チューリング機械
- ランダム・アクセス機械
- カウンタ機械
- ポインタ機械
- タグ・システム
- 帰納的関数 (μ 再帰関数)
- ラムダ計算
- マルコフ・アルゴリズム

数学的事実

これらの計算モデルにおける計算可能性は等価

これらの計算モデルで計算できることは変わらない (と証明できる)

計算可能性の等価性 (2)

計算モデルとは？ (直感的な定義)

「計算主体」を数学的に抽象化したもの

代表的な計算モデル

- チューリング機械
- ランダム・アクセス機械
- カウンタ機械
- ポインタ機械
- タグ・システム
- 帰納的関数 (μ 再帰関数)
- ラムダ計算
- マルコフ・アルゴリズム

数学的事実 (2)

これらの計算モデルにおける計算可能性は WHILE 計算可能性と等価

つまり，WHILE プログラムはこれらの計算モデルと同等に強力

チャーチ・チューリングの定立 (提唱，テーゼ)

チャーチ・チューリングの定立

我々が直観的に考える「計算できる」ことを
前のページで挙げた計算モデルによって「計算可能である」ことと見なす

注

- これは「数学的事実」ではない (証明できない)
- チャーチ・チューリングの定立を認めれば
  我々が直観的に考える「計算できる」ことは
  「WHILE 計算可能である」ことと見なせる

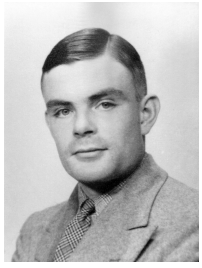
用語の定義：チューリング完全

計算モデル M がチューリング完全であるとは
チューリング機械で計算可能であることを M が計算可能であること

- つまり，前のページで挙げた計算モデルはすべてチューリング完全
- ほとんどのプログラミング言語はチューリング完全

チャーチとチューリング

Alonzo Church
(1903–1995)



Alan Turing
(1912–1954)

https://mathshistory.st-andrews.ac.uk/Biographies/Church/
https://www.britannica.com/biography/Alan-Turing

今日の目標

今日の目標

「計算可能性が等価である」ことの証明ができるようになる

- 簡単なプログラミング言語をあらたに 2 つ紹介する
  - LOOP プログラム
  - GOTO プログラム
- この 2 つと WHILE プログラムの計算可能性を比較する

重要な概念

- チューリング完全
- 構造的帰納法
- 計算の模倣

復習：WHILE プログラム (1)

機能をそぎ落として，本質的な部分だけを残したプログラミング言語

データ型

- 自然数 $0, 1, 2, \ldots$ ($\mathbb{N}$ の要素)

構成要素

変数

- $x_0, x_1, x_2, \ldots$

記号

- $:=$, ;, WHILE, DO, END
- $\neq 0$, $+1$, -1

WHILE プログラムの例

```
WHILE x_1 ≠ 0 DO
  x_0 := x_0 + 1;
  x_1 := x_1 − 1
END;
WHILE x_2 ≠ 0 DO
  x_0 := x_0 + 1;
  x_2 := x_2 − 1
END
```


復習：WHILE プログラム (2)：文法 (構文論)

WHILE プログラムの文法 (構文論)

WHILE プログラムは再帰的に定義される

1. 変数 x に対して，
   「$x := x + 1$」は WHILE プログラム
2. 変数 x に対して，
   「$x := x - 1$」は WHILE プログラム
3. WHILE プログラム P_1, P_2 に対して，
   「$P_1; P_2$」は WHILE プログラム
4. 変数 x と WHILE プログラム P に対して，
   「WHILE $x \neq 0$ DO P END」は
   WHILE プログラム

```
WHILE x_1 ≠ 0 DO
  x_0 := x_0 + 1;
  x_1 := x_1 − 1
END;
WHILE x_2 ≠ 0 DO
  x_0 := x_0 + 1;
  x_2 := x_2 − 1
END
```


復習：WHILE プログラム (3)：意味論

$x := x + 1$
$\rightsquigarrow$ x の値を 1 だけ増やす

$x := x - 1$
$\rightsquigarrow$ x の値を 1 だけ減らす
　x の値が 0 であるとき，x の値を 0 のままにする

$P_1; P_2$
$\rightsquigarrow$ P_1 を実行してから，P_2 を実行する

WHILE $x \neq 0$ DO P END
$\rightsquigarrow$ x が 0 でない限り，P の実行を続ける

注

- 変数 $x_1, \ldots, x_k$ は入力の値で初期化される
- その他の変数は 0 で初期化される
- 変数 x_0 が出力を表す

目次

LOOP プログラム (1)

データ型

- 自然数 $0, 1, 2, \ldots$ ($\mathbb{N}$ の要素)

構成要素

変数

- $x_0, x_1, x_2, \ldots$

記号

- $:=$, ;, LOOP, DO, END
- $+1$, -1

LOOP プログラムの例

```
LOOP x_1 DO
  x_0 := x_0 + 1
END;
LOOP x_2 DO
  x_0 := x_0 + 1
END
```


LOOP プログラム (2)：文法 (構文論)

LOOP プログラムの文法 (構文論)

LOOP プログラムは再帰的に定義される

1. 変数 x に対して，
   「$x := x + 1$」は LOOP プログラム
2. 変数 x に対して，
   「$x := x - 1$」は LOOP プログラム
3. LOOP プログラム P_1, P_2 に対して，
   「$P_1; P_2$」は LOOP プログラム
4. 変数 x と LOOP プログラム P に対して
   (ただし，P の中に x は現れない)，
   「LOOP x DO P END」は LOOP プログラム

```
LOOP x_1 DO
  x_0 := x_0 + 1
END;
LOOP x_2 DO
  x_0 := x_0 + 1
END
```

LOOP プログラム (3)：意味論

$x := x + 1$
- $\rightsquigarrow$ x の値を 1 だけ増やす

$x := x - 1$
- $\rightsquigarrow$ x の値を 1 だけ減らす
  x の値が 0 であるとき，x の値を 0 のままにする

$P_1; P_2$
- $\rightsquigarrow$ P_1 を実行してから，P_2 を実行する

LOOP x DO P END
- $\rightsquigarrow$ P を x の値の回数だけ実行する

注
- 変数の扱いは WHILE プログラムと同じ

LOOP 計算可能性

部分関数 $f \colon \mathbb{N}^k \rightharpoonup \mathbb{N}$

定義：LOOP 計算可能性

f が LOOP 計算可能であるとは，
次を満たす k 入力 LOOP プログラム P が存在すること

$$\begin{array}{lcl} f(x_1, \ldots, x_k)\downarrow & \Rightarrow & P \text{ の出力は } f(x_1, \ldots, x_k) \\ f(x_1, \ldots, x_k)\uparrow & \Rightarrow & P \text{ は停止しない} \end{array}$$

このとき，P は f を計算するという

先の例より，次の (部分) 関数 $f \colon \mathbb{N}^2 \rightharpoonup \mathbb{N}$ は LOOP 計算可能

$$f(x_1, x_2) = x_1 + x_2$$

目次

GOTO プログラム (1)

データ型
- 自然数 $0, 1, 2, \ldots$ ($\mathbb{N}$ の要素)

構成要素

変数
- $x_0, x_1, x_2, \ldots$

ラベル
- $L_1, L_2, L_3, \ldots,$

記号
- $:=,\ = 0,\ ;,\ :,\ +1,\ -1$
- IF, THEN, GOTO, HALT

GOTO プログラムの例

L_1: IF $x_1 = 0$ THEN GOTO L_5;
L_2: $x_0 := x_0 + 1$;
L_3: $x_1 := x_1 - 1$;
L_4: GOTO L_1;
L_5: IF $x_2 = 0$ THEN GOTO L_9;
L_6: $x_0 := x_0 + 1$;
L_7: $x_2 := x_2 - 1$;
L_8: GOTO L_5;
L_9: HALT

GOTO プログラム (2)：文法 (構文論)

GOTO プログラムの文法 (構文論)

GOTO プログラムは次の形をしている

L_1: S_1;
L_2: S_2;
$\vdots$
L_n: S_n

- $L_1, L_2, \ldots, L_n$ はラベル
- $S_1, S_2, \ldots, S_n$ は文 (次のページで定義される)

GOTO プログラム (2)：文法 (構文論) 続き

GOTO プログラムの文法 (構文論) 続き

GOTO プログラムの文は次のいずれか

1. 変数 x に対して，
   「$x := x + 1$」
2. 変数 x に対して，
   「$x := x - 1$」
3. ラベル L に対して，
   「GOTO L」
4. 変数 x とラベル L に対して，
   「IF $x = 0$ THEN GOTO L」
5. 「HALT」

L_1: IF $x_1 = 0$ THEN GOTO L_5;
L_2: $x_0 := x_0 + 1$;
L_3: $x_1 := x_1 - 1$;
L_4: GOTO L_1;
L_5: IF $x_2 = 0$ THEN GOTO L_9;
L_6: $x_0 := x_0 + 1$;
L_7: $x_2 := x_2 - 1$;
L_8: GOTO L_5;
L_9: HALT

GOTO プログラム (4)：意味論

基本
- $\rightsquigarrow$ ラベル $L_1, L_2, \ldots$ の順に文を実行する

$x := x + 1$
- $\rightsquigarrow$ x の値を 1 だけ増やす

$x := x - 1$
- $\rightsquigarrow$ x の値を 1 だけ減らす
  x の値が 0 であるとき，x の値を 0 のままにする

GOTO L
- $\rightsquigarrow$ ラベル L に次の実行を移す

IF $x = 0$ THEN GOTO L
- $\rightsquigarrow$ x が 0 であるとき，ラベル L に次の実行を移す

HALT
- $\rightsquigarrow$ 実行を終了する

注
- 変数の扱いは WHILE プログラム，LOOP プログラムと同じ

GOTO プログラム (6)：例の実行

入力 $x_1 = 2, x_2 = 1$ として，実行してみる

L_1: IF $x_1 = 0$ THEN GOTO L_5;
L_2: $x_0 := x_0 + 1$;
L_3: $x_1 := x_1 - 1$;
L_4: GOTO L_1;
L_5: IF $x_2 = 0$ THEN GOTO L_9;
L_6: $x_0 := x_0 + 1$;
L_7: $x_2 := x_2 - 1$;
L_8: GOTO L_5;
L_9: HALT

- $x_0 = 0$
- $x_1 = 2$
- $x_2 = 1$

$\therefore$ 出力は 3

GOTO 計算可能性

部分関数 $f\colon \mathbb{N}^k \rightharpoonup \mathbb{N}$

定義：GOTO 計算可能性

f が GOTO 計算可能であるとは，
次を満たす k 入力 GOTO プログラム P が存在すること

$$\begin{array}{ll} f(x_1,\ldots,x_k)\downarrow & \Rightarrow \quad P \text{ の出力は } f(x_1,\ldots,x_k) \\ f(x_1,\ldots,x_k)\uparrow & \Rightarrow \quad P \text{ は停止しない} \end{array}$$

このとき，P は f を計算するという

先の例より，次の (部分) 関数 $f\colon \mathbb{N}^2 \rightharpoonup \mathbb{N}$ は GOTO 計算可能

$$f(x_1,x_2) = x_1 + x_2$$

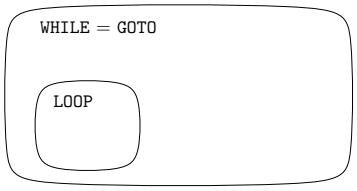
目次

ここからの目標

目標

3 つの「計算可能性」を比較する
- WHILE 計算可能性
- LOOP 計算可能性
- GOTO 計算可能性

先に結論



LOOP 計算可能 ⇒ WHILE 計算可能

自然数 k，部分関数 $f\colon \mathbb{N}^k \rightharpoonup \mathbb{N}$

定理 3.1

f が LOOP 計算可能 ⇒ f は WHILE 計算可能

証明のアイディア：
- LOOP プログラムを WHILE プログラムに書き換える
- そのためには「LOOP x DO P END」を書き換えれば十分

用語：計算の模倣

LOOP プログラム P を WHILE プログラム P' に書き換えることを
P' で P を模倣するという

模倣 = simulation (シミュレーション)

LOOP 計算可能 ⇒ WHILE 計算可能

証明：「LOOP x DO P END」を次のように書き換えればよい □

```
y := x;
WHILE y ≠ 0 DO
  P;
  y := y − 1
END
```

ただし，y はどこにも使われていない変数とする．

注：
- 代入 $y := x$ は糖衣構文なので，
  本来は WHILE プログラムに書き換える必要がある

WHILE 計算可能 ⇏ LOOP 計算可能

逆は成り立たない

定理 3.2

LOOP 計算可能ではない WHILE 計算可能部分関数が存在する

証明のアイディア：
- 停止しない WHILE プログラムが存在する (前回の例：infloop)
- LOOP プログラムは必ず停止する (今から証明すること)
- ∴ 停止しない WHILE プログラムは LOOP 計算可能ではない

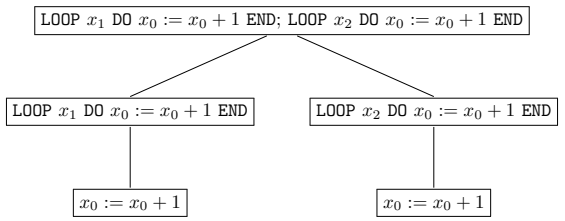
つまり，
- LOOP プログラムでは，全域関数しか計算できない

LOOP プログラムは必ず停止する (1)

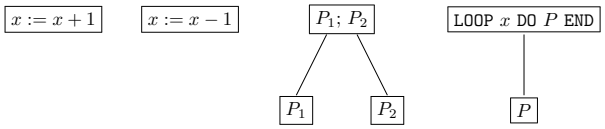
補題

LOOP プログラムは必ず停止する
(つまり，LOOP 計算可能な部分関数は 必ず 全域関数である)

証明のアイディア：構造的帰納法で証明する



LOOP プログラムの構造



LOOP プログラムは必ず停止する (2)

補題の証明：LOOP プログラム P を考える

- P がある変数 x に対して「$x := x + 1$」であるとき，
  x の値を 1 だけ増加して，P は停止する
- P がある変数 x に対して「$x := x - 1$」であるとき，
  x の値を 1 だけ減少して (x の値が 0 のときはそのままにして)，
  P は停止する
- P はある LOOP プログラム P_1, P_2 に対して「$P_1; P_2$」であるとき，
  P_1, P_2 が停止するならば，
  P は P_1 と P_2 を順に実行して停止する
- P がある変数 x と LOOP プログラム P' に対して
  「LOOP x DO P' END 」であるとき，
  P' が停止するならば，
  P は x の値の回数だけ P' を繰り返し実行して停止する

つまり，どの場合でも P は停止する □

目次

GOTO 計算可能 ⇒ WHILE 計算可能

自然数 k，部分関数 $f \colon \mathbb{N}^k \nrightarrow \mathbb{N}$

定理 3.3

f が GOTO 計算可能 ⇒ f は WHILE 計算可能

つまり，

- GOTO プログラムを WHILE プログラムに書き換えられる
  (WHILE プログラムで GOTO プログラムを模倣できる)

そのために，糖衣構文を導入する

WHILE プログラムの糖衣構文：条件分岐 (3)

条件分岐の機能

WHILE プログラム P，変数 x に対して，

IF $x = 0$ THEN P END

と書いて，x の値が 0 であるならば P を実行する機能を実現したい

注：前回の条件分岐では，条件が「$x \neq 0$」だった

```
IF x = 0 THEN
  P
END
```
⇝
```
IF x ≠ 0 THEN
  y := y + 1
ELSE
  P
END
```

ただし，y はプログラムに出てこない変数

WHILE プログラムの糖衣構文：条件分岐 (4)

条件分岐の機能

WHILE プログラム P，変数 x に対して，

IF $x = 2$ THEN P END

と書いて，x の値が 2 であるならば P を実行する機能を実現したい

```
IF x = 2 THEN
  P
END
```
⇝
```
y := x; y' := 1;
IF y = 0 THEN y' := 0 END;
y := y − 1;
IF y = 0 THEN y' := 0 END;
y := y − 1;
IF y = 0 THEN
  IF y' ≠ 0 THEN P END
END
```

ただし，y, y' はプログラムに出てこない変数

WHILE プログラムの糖衣構文：条件分岐 (5)

条件分岐の機能

WHILE プログラム P_1, P_2，変数 x に対して，

IF $x = 2$ THEN P_1 ELSE P_2 END

と書いて，x の値が 2 であるならば P_1 を実行し，
そうでなければ，P_2 を実行する機能を実現したい

```
IF x = 2 THEN
  P1
ELSE
  P2 END
```
⇝
```
y := 1;
IF x = 2 THEN
  P1; y := 0
END;
IF y ≠ 0 THEN
  P2
END
```

ただし，y はプログラムに出てこない変数

GOTO プログラムを WHILE プログラムに書き換える (1)

GOTO プログラム

```
L1: S1;
L2: S2;
 ⋮
Ln: Sn
```
⇝

WHILE プログラム
```
y := 1;
WHILE y ≠ 0 DO
  IF y = 1 THEN P1 END;
  IF y = 2 THEN P2 END;
    ⋮
  IF y = n THEN Pn END
END
```

y はプログラムに出てこない変数

P_i は，次に説明する通り，S_i から作られる

GOTO プログラムを WHILE プログラムに書き換える (2)

S_i が 「$x := x \pm 1$」 のとき
⇝ P_i は「$x := x \pm 1; y := y + 1$」

S_i が 「GOTO L_j」 のとき
⇝ P_i は「$y := j$」

S_i が「IF $x = 0$ THEN GOTO L_j」 のとき
⇝ P_i は「IF $x = 0$ THEN $y := j$ ELSE $y := y + 1$ END」

S_i が「HALT」 のとき
⇝ P_i は「$y := 0$」

これで，GOTO プログラムを WHILE プログラムに書き換えられる □

GOTO プログラムを WHILE プログラムに書き換える：例

L_1: IF $x_1 = 0$ THEN GOTO L_5;
L_2: $x_0 := x_0 + 1$;
L_3: $x_1 := x_1 - 1$;
L_4: GOTO L_1;
L_5: IF $x_2 = 0$ THEN GOTO L_9;
L_6: $x_0 := x_0 + 1$;
L_7: $x_2 := x_2 - 1$;
L_8: GOTO L_5;
L_9: HALT

$\rightsquigarrow$

```
x3 := 1;
WHILE x3 ≠ 0 DO
  IF x3 = 1 THEN
    IF x1 = 0 THEN x3 := 5
    ELSE x3 := x3 + 1 END
  END;
  IF x3 = 2 THEN x0 := x0 + 1; x3 := x3 + 1 END;
  IF x3 = 3 THEN x1 := x1 − 1; x3 := x3 + 1 END;
  IF x3 = 4 THEN x3 := 1 END;
  IF x3 = 5 THEN
    IF x2 = 0 THEN x3 := 9
    ELSE x3 := x3 + 1 END
  END;
  IF x3 = 6 THEN x0 := x0 + 1; x3 := x3 + 1 END;
  IF x3 = 7 THEN x2 := x2 − 1; x3 := x3 + 1 END;
  IF x3 = 8 THEN x3 := 5 END;
  IF x3 = 9 THEN x3 := 0 END
END
```


WHILE 計算可能 $\Rightarrow$ GOTO 計算可能

自然数 k，部分関数 $f \colon \mathbb{N}^k \rightharpoonup \mathbb{N}$

定理 3.4

f が WHILE 計算可能 $\Rightarrow$ f は GOTO 計算可能

つまり，

- WHILE プログラムを GOTO プログラムに書き換えられる
  (GOTO プログラムで WHILE プログラムを模倣できる)

基本的な考え方

- WHILE プログラムは「;」でつないだ順に実行される
  - 例外は「WHILE $x \neq 0$ DO P END」だけ
- GOTO プログラムは前から順に実行される
  - 例外は「GOTO L」と「IF $x = 0$ THEN GOTO L」だけ

WHILE プログラムを GOTO プログラムに書き換える (1)

WHILE プログラム

P_1;
P_2;
$\vdots$
P_n

$\rightsquigarrow$

GOTO プログラム

L_1: S_1;
L_2: S_2;
$\vdots$
$\vdots$
L_m: HALT

$S_1, S_2, \ldots, S_{m-1}$ は，次に説明する通り，$P_1, P_2, \ldots, P_n$ から作られる
(S_i が P_i から作られるわけではない)

WHILE プログラムを GOTO プログラムに書き換える (2)

状況

既に，$P_1, \ldots, P_{i-1}$ から $S_1, \ldots, S_{j-1}$ が作られ，
次に，P_i を処理するときを考える (次のラベルは L_j)

P_i が「$x := x \pm 1$」のとき

$\rightsquigarrow$ L_j: $x := x \pm 1$;

P_i が「WHILE $x \neq 0$ DO P END」のとき

$\rightsquigarrow$ L_j: IF $x = 0$ THEN GOTO L_ℓ;
L_{j+1}: (P を書き換える)
$\vdots$
$L_{\ell-1}$: GOTO L_j;

これで，WHILE プログラムを GOTO プログラムに書き換えられる $\square$

注：WHILE プログラムのシンプルさの恩恵

WHILE プログラムを GOTO プログラムに書き換える：例

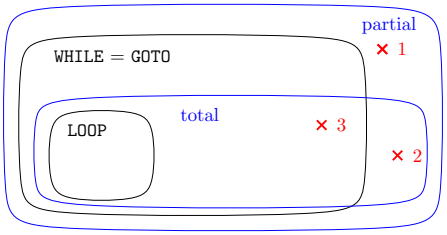
```
WHILE x1 ≠ 0 DO
  x0 := x0 + 1;
  x1 := x1 − 1
END;
WHILE x2 ≠ 0 DO
  x0 := x0 + 1;
  x2 := x2 − 1
END
```

$\rightsquigarrow$

L_1: IF $x_1 = 0$ THEN GOTO L_5;
L_2: $x_0 := x_0 + 1$;
L_3: $x_1 := x_1 - 1$;
L_4: GOTO L_1;
L_5: IF $x_2 = 0$ THEN GOTO L_9;
L_6: $x_0 := x_0 + 1$;
L_7: $x_2 := x_2 - 1$;
L_8: GOTO L_5;
L_9: HALT

目次

今日のまとめ：計算可能性の比較



疑問

1. WHILE 計算可能でない部分関数はあるか？ 具体例はあるか？
2. WHILE 計算可能でない全域関数はあるか？ 具体例はあるか？
3. LOOP 計算可能でない全域関数はあるか？ 具体例はあるか？

$\rightsquigarrow$ 後の講義で議論

今日のまとめ と 次回の予告

今日の目標

「計算可能性が等価である」ことの証明ができるようになる

- 簡単なプログラミング言語をあらたに 2 つ紹介する
  - LOOP プログラム
  - GOTO プログラム
- この 2 つと WHILE プログラムの計算可能性を比較する

重要な概念

- チューリング完全
- 構造的帰納法
- 計算の模倣

次回の予告

- 自然数以外のデータを扱う方法 $\rightsquigarrow$ コード化
- プログラム自体を入力とするプログラム