# Mizarによる計算量のためのアルゴリズムの形式化について

2017年12月7日

信州大学

岡崎　裕之

# 背景



◆形式検証、定理証明の工学への応用

- 数学だけではなく情報などの分野
  - ➢プログラム
  - ➢符号理論

◆暗号の安全性検証

- 暗号システムの安全性 (サービス)
- 暗号プロトコルの安全性検証（通信）
- 暗号方式の安全性検証（RSA,AES）
- (実装の安全性検証)(実際のソフト)

# 背景



## ◆我々の取り組み

形式手法による暗号の安全性証明自動検証システムの開発

- 暗号方式の安全性検証（RSA,AES）
  - ➢Mizar（定理証明系）

    岡崎,師玉(信州大),布田(東京工科大)
- 暗号プロトコルの安全性検証（通信）
  - ➢ProVerif（モデルチェッカー）

    荒井(長崎大)　（岡崎,布田）

# どんなことをやっていたか

◆共通鍵暗号に関するアーティクル
- DES暗号（DESCIP_1）、AES

◆数論及び代数系に関するアーティクル
- 素因数分解の一意性, 有限素体（INT_7）
- 有限素体上の楕円曲線(EC_PF_1,2)

◆アルゴリズムに関するアーティクル
- GCD,EXGCD,CRT(NTALGO_1)

◆確率に関するアーティクル
- 離散確率（DIST_1,2）
- 確率変数、確率測度（RANDOM_1,2）

◆計算量に関するアーティクル
- 多項式オーダー、無視可能関数(ASYMPT_2,3)

# アルゴリズムの形式化

◆アルゴリズムやプログラムの形式化にはいろいろな目的があるので形式化方針が異なる

- アルゴリズムの正当性（正常に終了するか？）
  チューリングマシン等
    伝統的な正当派？
- 実装の正しさまで検証する
    計算機のハードまで形式化してしまう
- 計算量を評価したい
  どれぐらいのコスト、ステップ数で終了するのか？

# アルゴリズムの形式化（Mizar）

◆Mizarでも既にあった

- 正当派？の形式化
- スタックマシンを定義して動作の検証を行う
- SCM* というシリーズ

◆例GCD

```
definition
  func GCD-Algorithm -> Program of SCMPDS equals
:: SCMP_GCD:def 4
::Def04
  (((GBP:=0) ';' (SBP := 7) ';' saveIC(SBP,RetIC) ';'
  goto 2 ';' halt SCMPDS ) ';' (SBP,3)<=0_goto 9 ';'
  ((SBP,6):=(SBP,3)) ';' Divide(SBP,2,SBP,3) ';'
  ((SBP,7):=(SBP,3)) ';' ((SBP,4+RetSP):=(GBP,1))) ';'
  AddTo(GBP,1,4) ';' saveIC(SBP,RetIC) ';'
  (goto -7) ';' ((SBP,2):=(SBP,6)) ';' return SBP;
end;
```

# 計算量について例

◆だいたい乗算何回ぐらいというカウント

◆nビット整数のべき乗

- 単純にべき乗すると

$$O(2^n)$$

- バイナリ法

$$O(n)$$

# どんなことをやっていたか

◆共通鍵暗号に関するアーティクル
- DES暗号（DESCIP_1）、AES

◆数論及び代数系に関するアーティクル
- 素因数分解の一意性, 有限素体（INT_7）
- 有限素体上の楕円曲線(EC_PF_1,2)

◆アルゴリズムに関するアーティクル
- GCD,EXGCD,CRT(NTALGO_1)

◆確率に関するアーティクル
- 離散確率（DIST_1,2）
- 確率変数、確率測度（RANDOM_1,2）

◆計算量に関するアーティクル
- 多項式オーダー、無視可能関数(ASYMPT_2,3)

# アルゴリズムの形式化

- ◆流石にSCMでやるのはかなりつらそう
- ◆（将来的に）確率的アルゴリズムも扱わないといけない（暗号理論には必要）となるとうまくいくかわからない
- ◆とりあえずプログラミングっぽく書けないか試作してみた
  - GCD,EXGCD,CRT(NTALGO_1)
  - ユークリッド互除法、拡張ユークリッド互除法、中国人の剰余定理（の計算）
  - これはら先行して作成していたDESの応用

# ユークリッド互除法

## Pythonのソースコード

```
def gcd(a, b):
    a, b = abs(a), abs(b)
    while b:
        a, b = b, a % b
    return a
```

## これをMizarに移植？

```
definition
  let a,b be Element of INT;
  func ALGO_GCD(a,b) -> Element of NAT means
  :defALGOGCD:
```

**sequence**

```
ex A,B be sequence of NAT
st
A.0 = abs(a) & B.0 = abs(b) &
(for i be Element of NAT holds
A.(i+1) = B.i &
B.(i+1) = A.i mod B.i) &
```

**終了条件**

```
it =
  A. (min*{
  i where i is Element of NAT: B.i = 0} );
```

# ユークリッド互除法の形式化方針

- ◆各変数の遷移をステップごとに書く
- ◆変数の列としてあつかう
- ◆終了条件を記述
  - DESの場合は繰り返しが16段固定であったので簡単であったが、互除法は「終了条件」をうまく書けるかどうかわからなかったのでとりあえず試してみたがうまくいった。
- ◆ユークリッド互除法以外も書いてみた
  - 拡張ユークリッド互除法
  - 中国人の剰余定理のアルゴリズム

```
definition
  let a,b be Element of INT;
  func ALGO_GCD(a,b) -> Element of NAT means
  :defALGOGCD:

ex A,B be sequence of NAT
st
A.0 = abs(a) & B.0 = abs(b) &
(for i be Element of NAT holds
A.(i+1) = B.i &
B.(i+1) = A.i mod B.i) &

it =
  A. (min*{
  i where i is Element of NAT: B.i = 0} );
```

終了ステップ数
についても
高々5log_2(b)
であることを
Mizarで証明済
（ラメの定理）

# ユークリッド互除法

◆アルゴリズムの出力結果が最大公約数であることを証明

```
theorem ::Th1:
  for a,b be Element of INT
  holds
  ALGO_GCD(a,b) = a gcd b
```

アルゴリズムの計算結果

aとbの最大公約数

# ユークリッド互除法

◆最大公約数の定義(数学の意味のまま)

```
definition
  let a,b be Integer;
  func a gcd b -> Nat means
  :Def3:
  it divides a & it divides b &
  for m being Integer
  st m divides a & m divides b
  holds m divides it;
```

# 計算量評価のためのアルゴリズムの形式化

◆アルゴリズムの記述はできた

◆ステップ数の見積もりもできた

◆ただし出来ただけで課題は多数

- 単純なループだけ
  - ➢分岐
  - ➢サブルーチン呼び出し
- 計算量評価はステップ数だけではない
  - ➢ 各ステップの計算コストの評価が必要
  - ➢計算量の評価は入力パラメータのオーダーで評価

◆形式的にアルゴリズムとは何かを定義したい

# アルゴリズム記述の一般化方針を考える

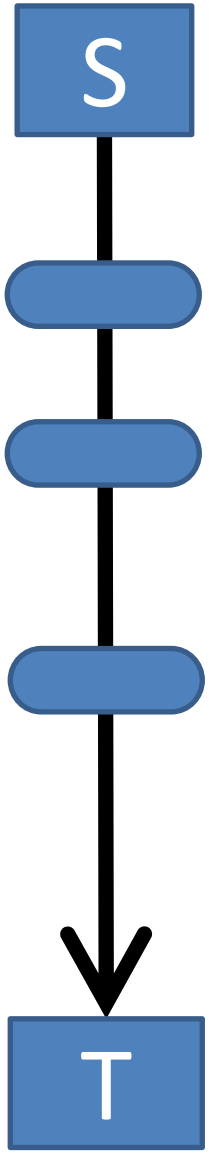
- ◆形式的にアルゴリズムとは何かを定義したい
- ◆互除法の形式化だと証明がやりやすかったので出来るだけ近い方針でやりたい
  - 何かのSequenceで表現する
- ◆ただし計算量評価が目的
  - 暗号理論の場合、性能評価ではないので計算量の最大値のオーダーが分かれば十分
- ◆アルゴリズムとは何か？から問い直す
  - 後で確率的アルゴリズムにも対応できる余地をのこしておきたい

# アルゴリズム記述の一般化方針を考える

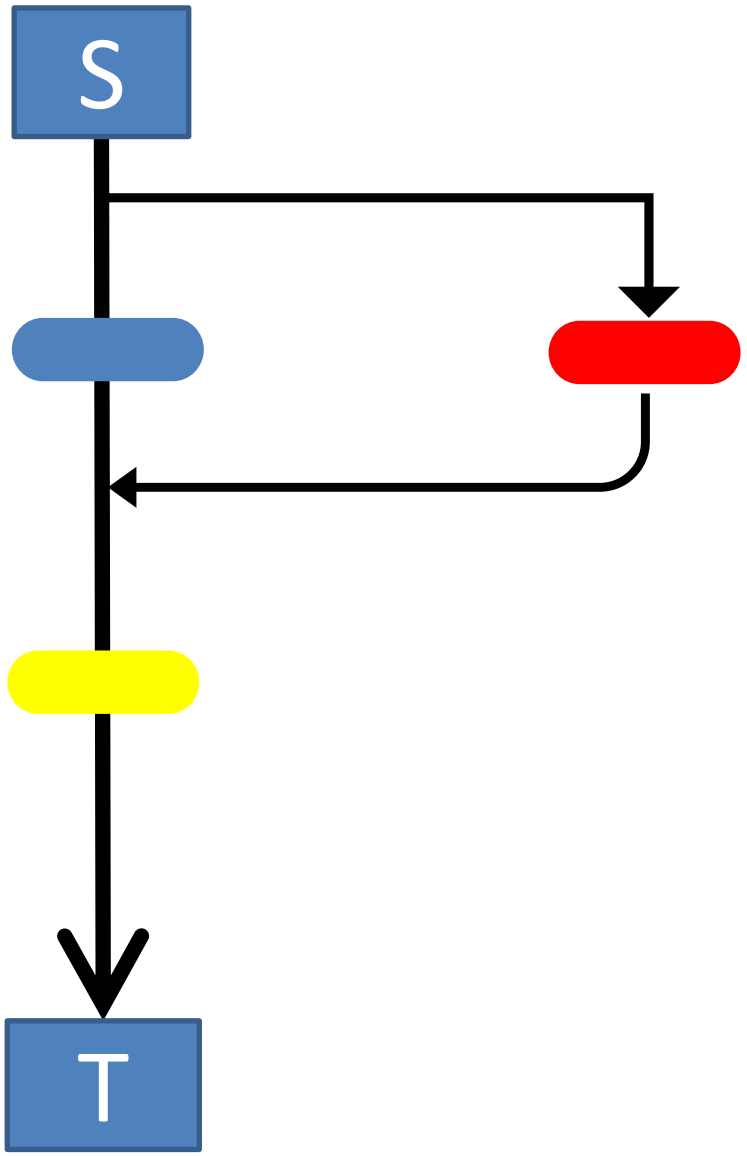
よくある状態遷移図みたいなものはどのような制約が課されているか（当たり前レベルで書いてみる）

- ◆初期状態S_0から終了状態S_Tへの遷移の列全体の集合
- ◆終了状態以外の状態は必ず次状態へ遷移する
- ◆いかなる遷移を経ようが必ず終了する

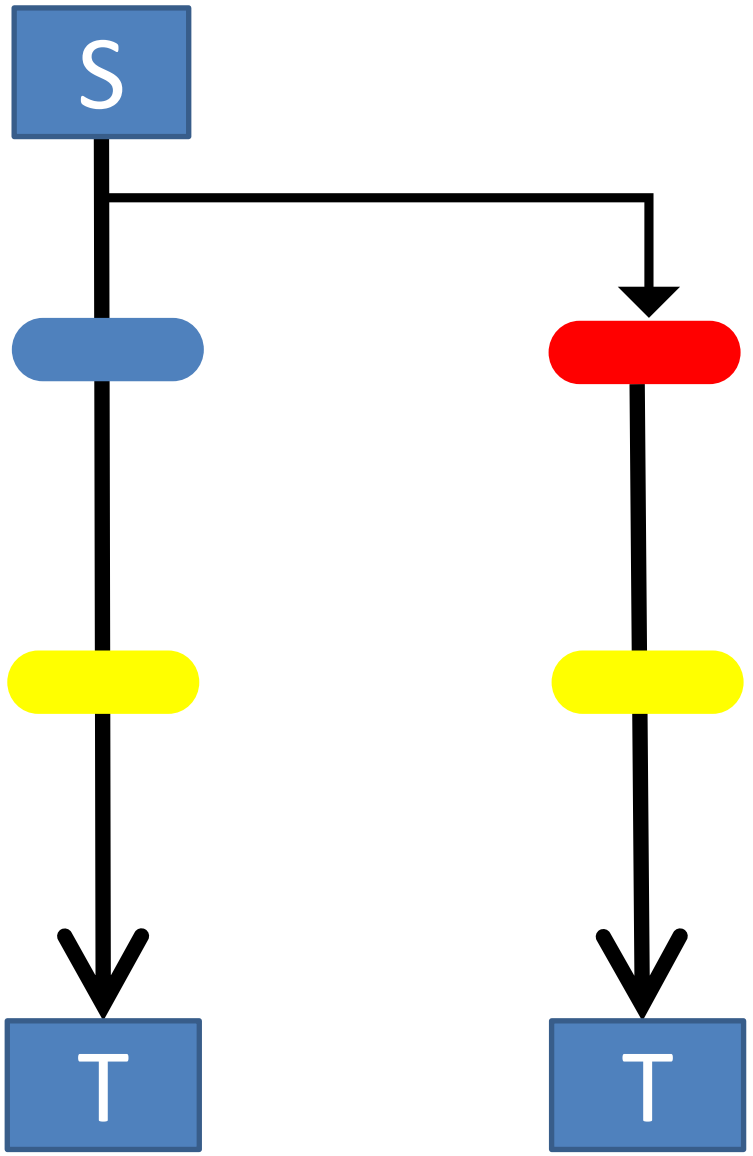
# アルゴリズム記述の一般化方針を考える



# アルゴリズム記述の一般化方針を考える



# アルゴリズム記述の一般化方針を考える



# アルゴリズム記述の一般化方針を考える

◆分岐等は木構造で表現

- 人間には面倒でも計算機にはその方が扱いやすい（はず）

◆計算量評価には最長のパス

- すなわち最も深い葉を含むパスを抽出して評価

（と思ったが問題がありそうなので今後の課題）

◆Mizarで木構造はFinSequenceの集合として定義されているので元々の互除法の形式化方法と親和性が良さそう

◆ということで木で表現してそこから一本FinSequenceを引っ張り出して計算量評価する

# アルゴリズム記述の一般化方針を考える

- ◆アルゴリズムでは分岐、サブルーチンなどいろいろ複雑になるが、最終的に評価する対象は初期状態から終了状態に至るある一本のパスなので、Mizarでは木構造（のうち葉からなる部分集合）から1つの要素を取り出してFinSequence（MizarではもともとFinSequence）として扱う
- ◆取り出したFinSequenceを互除法の形式化方法と同様にやってみる
- ◆FinSequenceであれば切ったり合成したりしやすい（Mizarの都合的に証明がやりやすい）

# アルゴリズムの形式化方針

◆分岐、等は木構造で書く

◆全体としては取り得る実行パス全体の集合

◆計算量評価のためにはある（一番コストの高い）実行パスを取り出して評価する。

◆最初に互除法でやったような形式化（手作り感満載の形式化）をきれいにしてみる

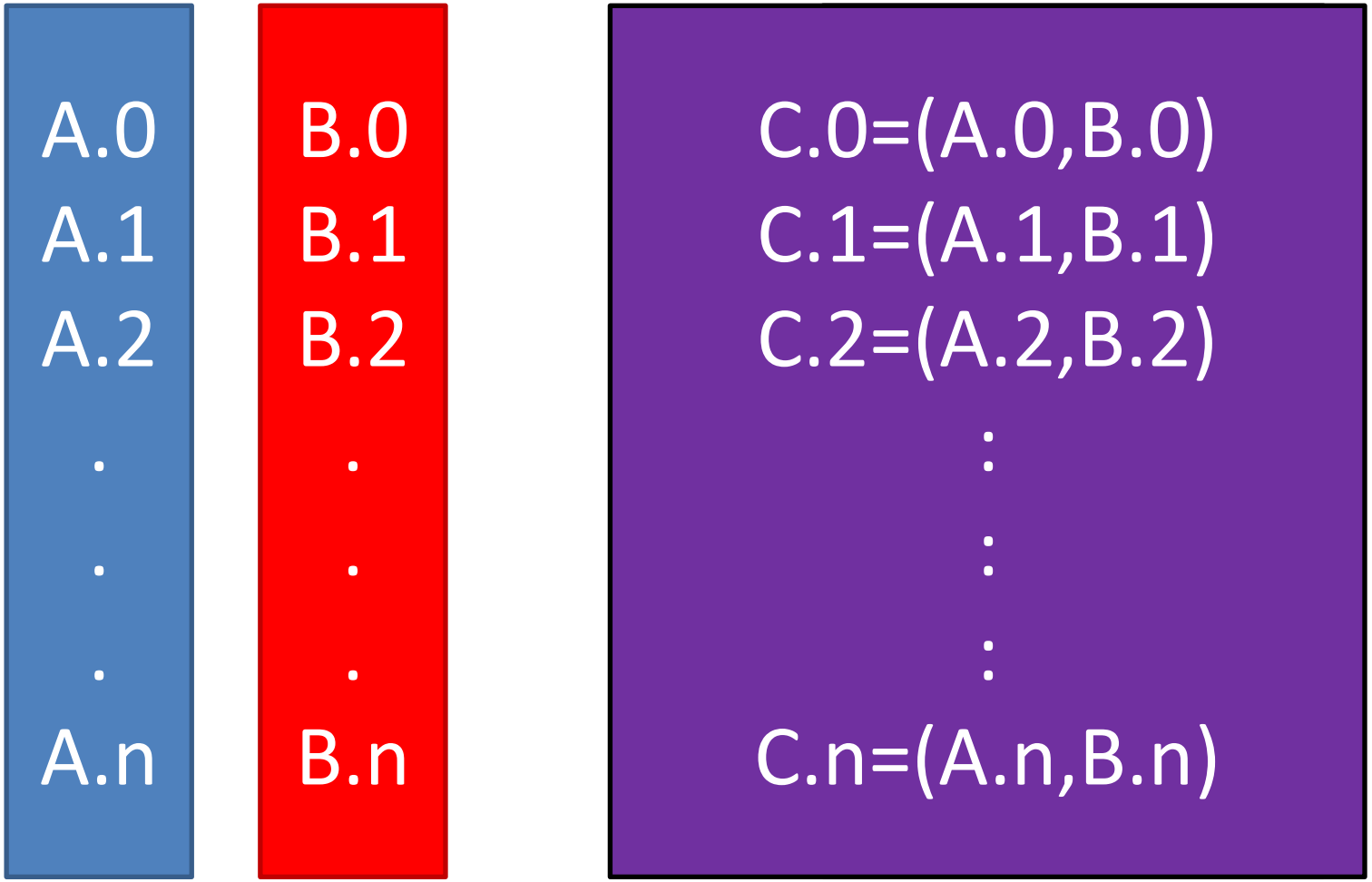
- 一般のアルゴリズムとは何かを定義する
- 計算量の評価、
  出力結果の正しさも一緒にかけるようにしたい。

```
definition
  let a,b be Element of INT;
  func ALGO_GCD(a,b) -> Element of NAT means
  :defALGOGCD:

  ex A,B be sequence of NAT
  st
  A.0 = abs(a) & B.0 = abs(b) &
  (for i be Element of NAT holds
  A.(i+1) = B.i &
  B.(i+1) = A.i mod B.i) &

  it =
    A. (min*{
    i where i is Element of NAT: B.i = 0} );
```

A,B2つのシーケンスを使っている

アルゴリズムによってパラメータ数は異なる

# パラメータを一般化




パラメータはnon empty setのSequenceでよい

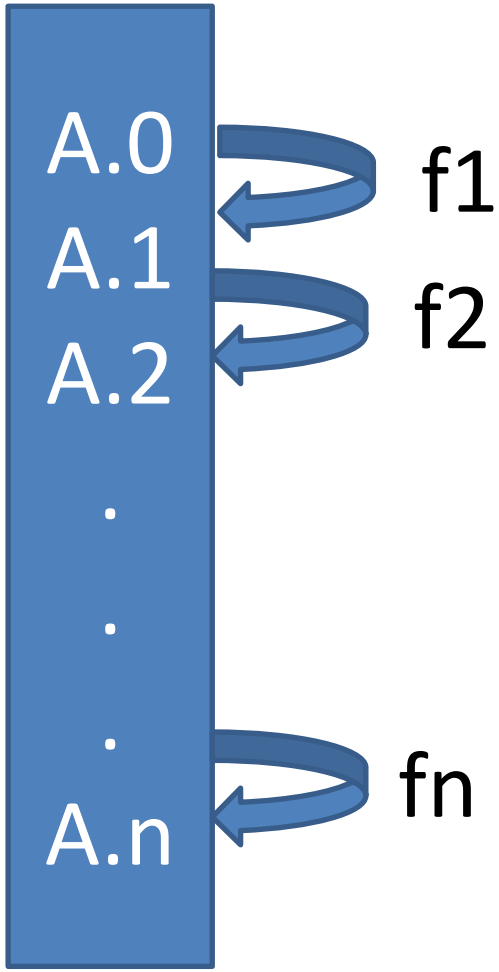
```
definition
  let a,b be Element of INT;
  func ALGO_GCD(a,b) -> Element of NAT means
  :defALGOGCD:
```

```
ex A,B be sequence of NAT
st
A.0 = abs(a) & B.0 = abs(b) &
(for i be Element of NAT holds
A.(i+1) = B.i &
B.(i+1) = A.i mod B.i) &
```

次ステップへの計算直接書いているけど、Functionがあるといってしまえば良くない？

```
it =
  A. (min*{
   i where i is Element of NAT: B.i = 0} );
```

# 次ステップへの関数が存在するとすればよい

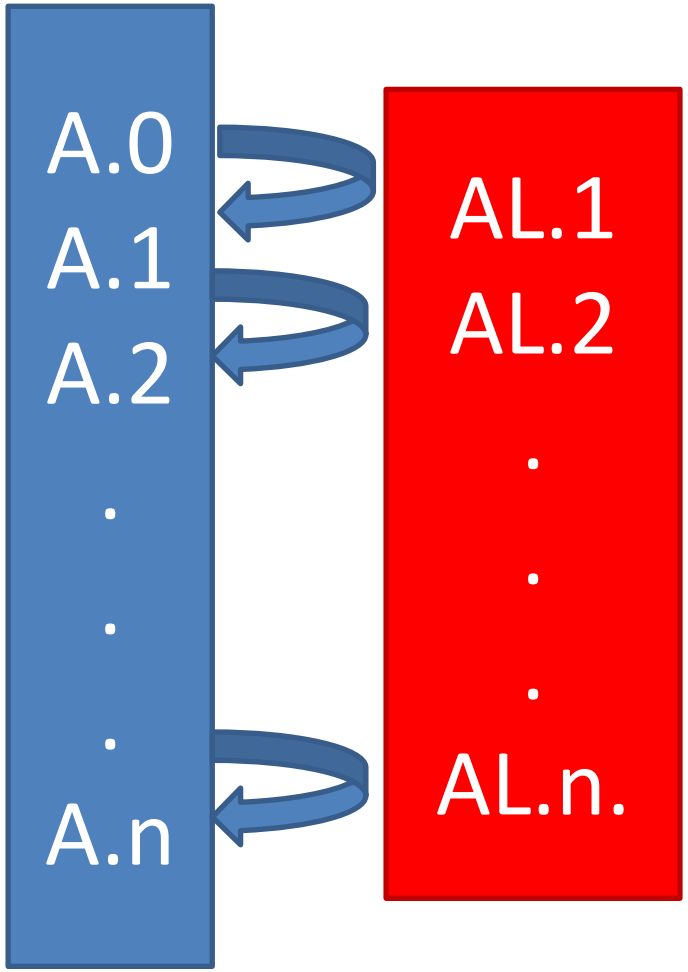



```
definition
let D be non empty set;
let A  be sequence of D;
   attr A is Program-like means
ex N be Nat st 0 < N &
   for n be Nat st n < N holds
    ex transF be Function st
[A.n,A.(n+1)] in transF;
end;
```

# Atttribute Program-like(仮)の定義から

```
definition
let D be non empty set;
let A  be sequence of D;
   attr A is Program-like means
ex N be Nat st 0 < N & for n be Nat st n < N holds
    ex transF be Function st  [A.n,A.(n+1)] in transF;
end;

theorem
for D be non empty set,
 A  be sequence of D st A is Program-like holds
ex N be Nat,  AL be Function st  0< N & [A.0,A.N] in AL;
```

# 関数の有限列として与えればよい？




$$F=(AL.n)*\cdots*(AL.2)*(AL.1)$$

# ユークリッド互除法

◆アルゴリズムの出力結果が最大公約数であることを証明

```
theorem ::Th1:
  for a,b be Element of INT
  holds
  ALGO_GCD(a,b) = a gcd b
```

アルゴリズムの計算結果

aとbの最大公約数

# ユークリッド互除法

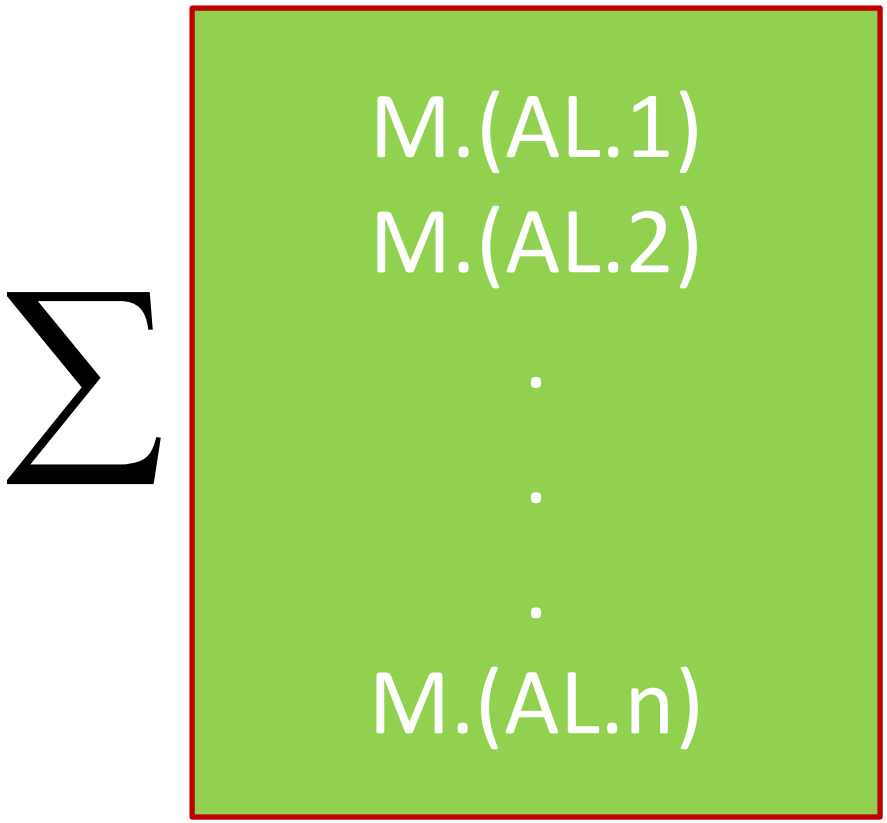
◆アルゴリズムの出力結果が最大公約数であることを証明

```
theorem ::Th1:
  for a,b be Element of INT
  holds
  ALGO_GCD(a,b) = a gcd b
```

(AL.n)*・・*(AL.2)*(AL.1)

F

# 関数の有限列とすれば計算量がうまく書けそう




関数から実数値への関数M

# まとめ

- ◆分岐等は木構造で表現
- ◆アルゴリズムを関数の有限列としてあつかうとうまくいきそう
- ◆計算量もうまく書けそう

- ◆まずは分岐のあるアルゴリズムの簡単な例を書いてみたい
- ◆計算量評価もやってみる