数理的技法普及のために求められる技術と教育

産業技術総合研究所 (AIST)
システム検証研究センター (CVS)
木下　佳樹

概要

- 数理的技法の手順（シナリオ）
- 各手順で技術者にとって必要な技術と科学
- 研究所の立場から科学的技能と知識の教育への要望

お話したいこと

- 知識より技能を教えて（伝えて）ほしい。
  知識は独習しやすい。必要に応じて本を読めばよい。「本の読み方」が技能！
- 個別のシステムのコマンドの細かい知識よりも、その原理を理解することが大事
  技術者が使うシステムはどんどん変わっていく。
  変わらないのは原理
  動作原理を理解すれば、新しいシステムを使いこなすのも早いはず

情報処理システム開発の数理的技法

- 「システムが意図通りに動いている」ことを**数学の定理**として定式化し、証明する。

  ← システムの設計や、システムを使う意図を記述するための数学的枠組をつくる（＝プログラミング意味論）

- さらに、定式化や証明を計算機支援する。

  ← 定理やその証明を人工言語で記述する（＝形式化）

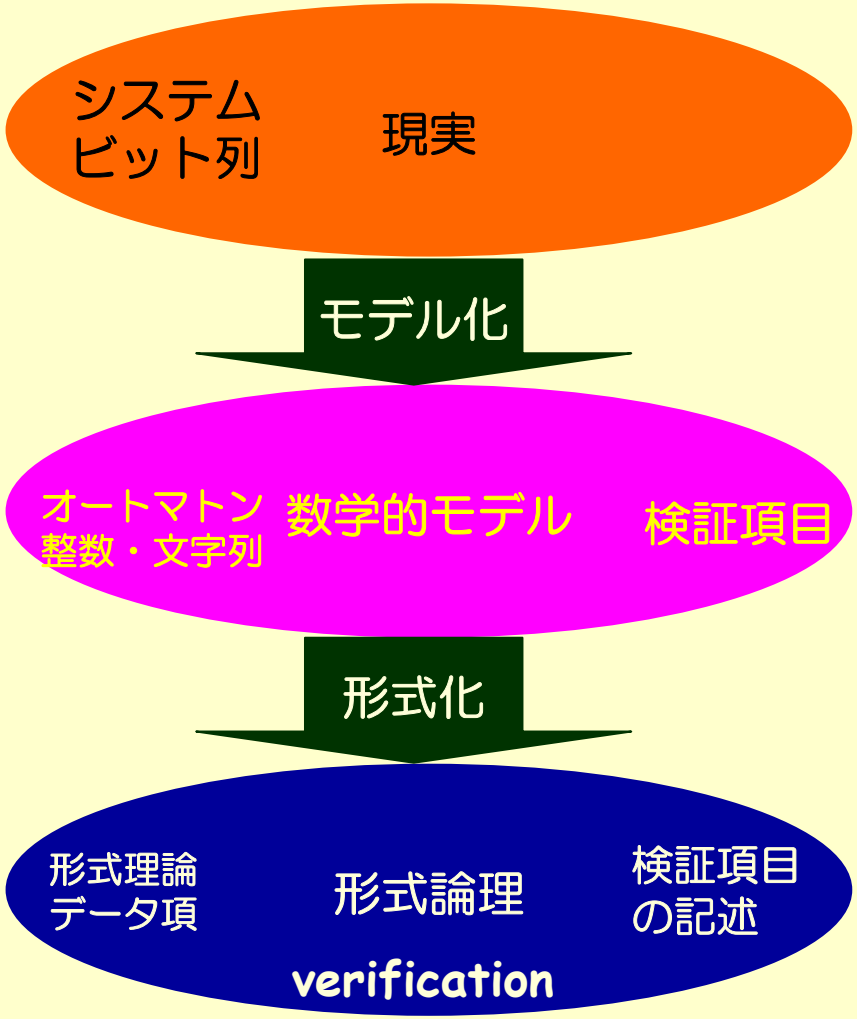
情報処理システム開発の数理的技法

- 数理的技法はコードレビューを計算機によって支援し、もっと厳密かつ大規模に行うことを可能にするもの
- 数理的技法は、システムの正しさを保証するのではなく、バグを検出する方法

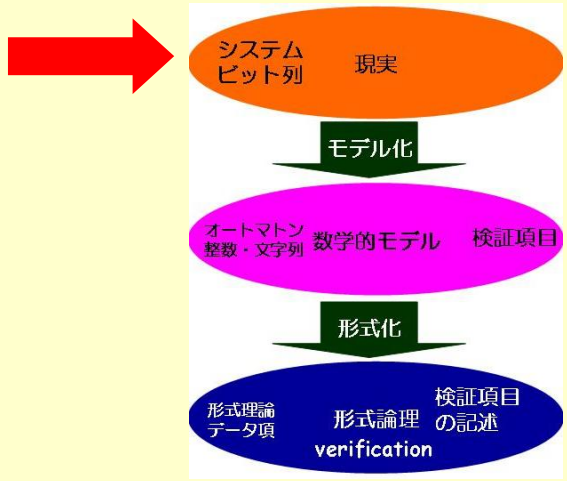
数理的技法のシナリオ

数理的技法のシナリオ

- モデル化（数学化）
- 形式化
- システムが検証項目を満たすかどうか調べる。
- 満たさなかった場合：
  反例を開発者が検討してバグかどうかを判断
- 満たした場合：
  バグは見つからなかった。



現実：プログラム



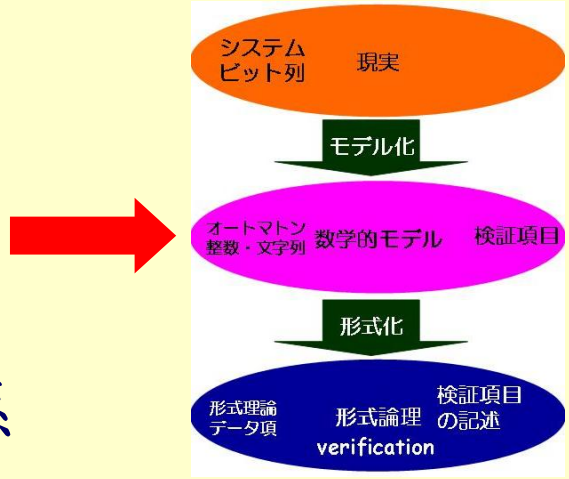

```
int st = 0;  int j = 0;
while (ch=getchar())
 switch (st) {
   0: st = ch==“0” ? 0 : 1
 | 1: st = ch==“0” ? 2 : 0
 | 2: st = ch==“1” ? 0 : 2
 };
if (st == 0)
  printf (“multiple of three!¥n”);
```

オートマトン

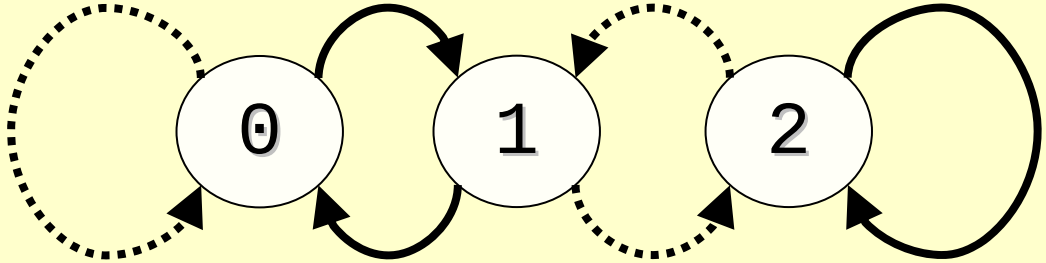
状態集合: { 0, 1, 2 }

入力記号: { 0, 1 }

受理状態: { 0 }



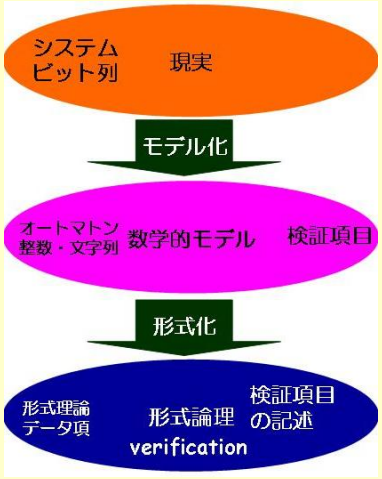
遷移関係



入力が0の時の遷移：

入力が1の時の遷移：

（オートマトンの）性質



受理言語は、
二進数としてみたときに、
3の倍数になっているものの
全体である。

“0” が零、“0”以外の文字が一だとして二進数としてみる。

最も左の文字（はじめに入力された文字）が、最も高位の数字だと見なす。

（Agdaによる）形式化


```
module AnAutomaton
   where
data Bool : Set where
  false : Bool;
   true : Bool
if_then_else_fi :
   {A : Set} -> Bool
   -> A -> A -> A
if true then x else _
   fi = x
if false then _ else
   y fi = y
data Nat : Set where
zero : Nat;  succ :
   Nat -> Nat
one = succ zero
two = succ one
three = succ two
```

```
data Vec (A : Set) :
 Nat -> Set where
  []   : Vec A zero
  _::_ :
    {n : Nat} -> A ->
   Vec A n
          -> Vec A
   (succ n)
State = Nat
s0 = zero;  s1 = one
s2 = two
```

```
OneStep :
 State -> Bool ->
   State
OneStep s0 sym =
 if sym then s1  else
   s0 fi
OneStep s1  sym =
 if sym then s0 else
   s2  fi
OneStep s2  sym =
 if sym then s2  else
   s1  fi
```

（Agdaによる）形式化


```
ManySteps : (l : Nat)
 ->(Vec Bool l)
 -> State -> State
ManySteps zero [] st
   = st
ManySteps (succ n)
   (sym :: syms) st =
   ManySteps n syms
   (OneStep st sym)
OurAutomaton :
   (inputLength :
   Nat) -> (Vec Bool
   inputLength) ->
   State
OurAutomaton n syms =
   ManySteps n syms
   s0
```

```
data _==_ :
 Nat -> Nat -> Set
 where
 zeq : zero == zero
 seq : (x y : Nat)
  -> x == y
  -> succ x == succ y
 refl : (x : Nat)
  -> x == x
 symm : (x y : Nat)
  -> x == y -> y == x
 tran : (x y z : Nat)
  ->  x == y
  -> y == z -> x == z
```

```
_<_ : Nat -> Nat
 -> Bool
zero < zero = false
zero < _ = true
succ _ < zero = false
succ m < succ n
 = m < n
_+_ : Nat -> Nat ->
   Nat
zero + n = n
(succ m) + n = succ
   (m + n)
```

（Agdaによる）形式化


```
_*_ : Nat -> Nat
      -> Nat
zero * _ = zero
(succ m) * n =
 (m * n) + n
_-_ : Nat -> Nat ->
   Nat
zero - _ = zero
succ m – zero
 = succ m
succ m - succ n
 = m – n
_mod_ : Nat -> Nat ->
   Nat
m mod n = if (m < n)
   then m
   else (m - n) mod n
   fi
```

```
NumberForInput : (n :
   Nat) -> Vec Bool n
   -> Nat
NumberForInput zero
   [] = zero
NumberForInput
 (succ m)
 (sym :: syms) =
 if sym then
  succ(two *
   NumberForInput m
   syms)
 else two *
   NumberForInput m
   syms
 fi
```

```
Proposition1 :
 (n : Nat) ->
 (syms : Vec Bool n)
   ->
 ((NumberForInput n
   syms) mod three)
   == OurAutomaton n
   syms

Proposition1
 zero [] _ = ?
```

モデル化とは？

情報処理の現実に数学的対象を
　あてはめて、数理的な処理を可能にする

技能：正しく「あてはめる」リテラシー
　現代数学の正しい理解

知識：あてはめることができる数学的対象
　の定石を沢山知っておく

数学的リテラシー

- 日常の議論を数学的対象によって表現する
- 公理化による議論

  「公理さえ満たしておれば、点がジョッキで線が椅子、面が机でもいい」Hilbert

  モノの名前から連想される性質を仮定しない。

  Cf. Lisp での「シンボル」と「数値」
- 集合と論理
  - 素朴集合論 (P. Halmos)

    {|}、直積、直和、写像と関係、集合の族、冪

    自然数と無限、帰納法
  - 数学の論理 ()

    ∀x∃y P(x,y)と∃y∀x P(x,y)の区別！

モデル化の定石

代数

普遍代数と等式論理

余代数

遷移系、オートマトン

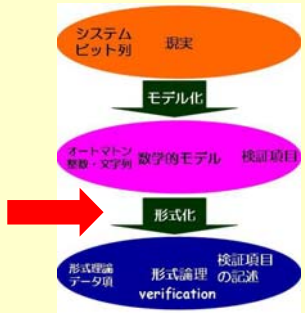
位相

順序集合

タルスキの不動点定理

束論

形式化とは？

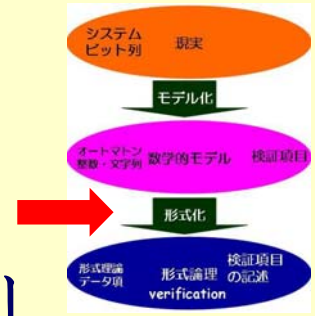


数学的対象を形式的体系にあてはめて、
　機械による処理を可能にする

技能：形式的体系の周辺

知識：あてはめることができる形式的対象
　のパタンを沢山知っておく

形式化の技能



- 構文(syntax)と意味(semantics)の区別

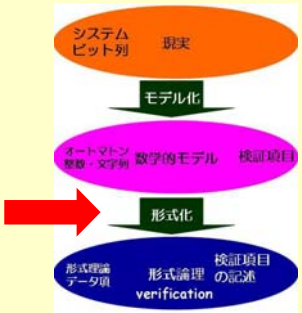
  while命令の構文と意味

- 帰納的定義、構成に関する帰納法

  S式の帰納的定義を与える

  S式の簡単な性質を、構成に関する帰納法によって示す。

whileコマンドの構文



原子コマンドの集まりAtomおよび
条件判定文の集まりCondの上の
whileコマンドCommの構文を以下のように定める。

Comm ::= Atom
| Comm ; Comm
| **if** Cond **then** Comm **else** Comm **fi**
| **while** Cond **do** Comm **od**

whileコマンドの意味

関係代数 (B,1,~,・,;,id,○,*) における
Atom, Condの上のwhileコマンドの解釈とは、写像 I :
Atom + Cond → B で、
b∈Condであれば I(b)≦idであるようなもの。

解釈Iの下で、各コマンドcの意味【c】が次のように定められる。

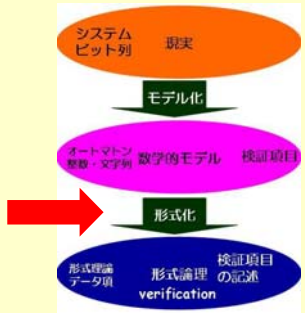
a∈Atomならば【a】= I(a)

c,c'∈Commならば【c;c'】=【c】;【c'】

b∈Cond, c,c'∈Commならば【**if** b **then** c **else** c' **fi**】=
【b】;【c】+ ~【b】;【c'】

b∈Cond, c∈Commならば【**while** b **do** c **od**】= (I(b);
【c】)*; ~I(b)

S式の帰納的定義



S式を次のように定義する。[佐藤雅彦]

1. 記号＊はS式である。
2. s、tがS式であれば、(s . t)はS式である。
3. 以上の1. 2. の規則を有限回適用してS式であると確かめられるものだけがS式である。

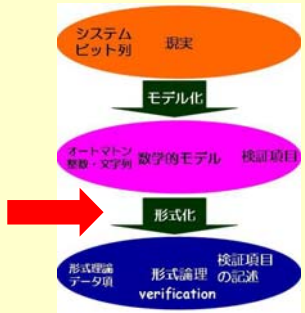
S式の例：

(＊.((＊.＊).＊))はS式である。

(＊.((△.＊).□))はS式ではない。

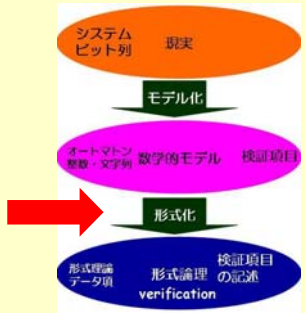
(＊.(＊.(＊.(＊. ....)))) はS式ではない。

S式の構成に関して
帰納的な定義



- S式sに対して、自然数nleaf(s)を、sの構成に関して帰納的に定める。
  1. nleaf(＊) = 1
  2. nleaf((s . t)) = nleaf(s) + nleaf(t)
- S式sに対して、自然数height(s)を、 sの構成に関して帰納的に次のように定める。
  1. height(＊) = 0
  2. height((s . t)) = max (height(s), height(t))

S式の構成に関する帰納法による証明



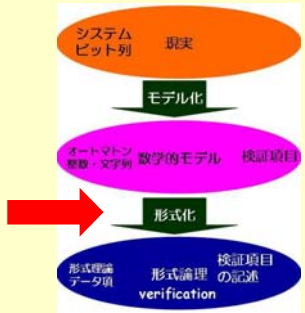
- height(s)≦nleaf(s) をsの構成に関して帰納法で証明する。この命題をP(s)と記す。
  1. s = ＊のとき height(＊) = 0 ≦ 1 = nleaf(＊) なので P(＊)が成り立つ。
  2. P(s) および P(t) が成り立つと仮定（帰納法の仮定）して、P((s . t)) が成り立つことを示す。

     height((s . t)) = max(height(s), height(t))

     ≦ height(s) + height(t) ≦ nleaf(s) + nleaf(t)

     = nleaf((s . t)).

形式化の定石



- 等式論理（Cf. データ型）
- 様相論理
- 一階述語論理
  - 完全性定理

    論理式の構文とそのモデルの区別

    帰納的定義と構造に関する帰納法による証明法

    代入と束縛

数理的技法には必要がうすい知識

- 算法全般、e.g., 自動証明の算法
  - CTLやLTLのモデル検査、FOLの定理証明などの算法
  - 自動検査器の性能をぎりぎりまで使う高度な利用をするのでなければ不要

研究所の立場からの要望

- 以上は**技術者**に求められる知識。
  - **研究者**にはこれらを教えることができる程度の理解がほしい。
- 研究者には同じことを、より基礎から理解することを求める
  - 構文と意味の**Lawvere theory**による理解
  - 帰納的定義や不動点の**普遍性(universality)**による理解
  - 遷移系の**余代数**による理解

お話したいこと（再掲）

- 知識より技能を教えて（伝えて）ほしい。
  特定の知識を教えることを通して技能を伝える。
  本の読み方を伝えるには本を読まなければならない。
- 個別のシステムのコマンドの細かい知識よりも、その原理を理解することが大事
  動作原理の理解のために個別のシステムを使う。
  複数のシステムを同時に学ぶのが、底に潜む動作原理を理解するために有効？

おわり