

(本当は)  
情報科学科の宣伝

# Hoare 論理による プログラム検証入門

蓮尾 一郎

東京大学理学部情報科学科 講師

<http://www-mmm.is.s.u-tokyo.ac.jp/~ichiro>



1

# プログラム検証： 最初の例

2

## While ループのある プログラム

```
n := N;  
k := 1;  
while (n > 0) {  
  k := k*n;  
  n := n-1;  
}
```

N の階乗 N! を  
計算するプログラム

ループ脱出!  
(n>0 でない)

k	n
1	N
N	N-1
$N*(N-1)$	N-2
$N*(N-1)*(N-2)$	N-3
...	...
$N*(N-1)*\dots*2$	1
$N*(N-1)*\dots*2*1$	0

3

## 本当に？

- \* 証明してみよう!
- \* 数学的に厳密に
- \* いろんなプログラムに使えるような、「スジのよい」方法で、できれば自動化
- \* **プログラム検証** program verification
- \* プログラムや計算機システムの正しさは重要、銀行、自動車、ロケット、セキュリティ、...

4

はてなブックマーク - タグ「システム障害」を含む注目エントリー

Show: 0 new items - all items | Mark all as read | Refresh | Feed settings...

お粗末！新幹線トラブル「今までなかったの」：社会：YOMIURI ONLINE (読売新聞) - お粗末！

IT業界の裏話：新幹線トラブルに見る、開発部門と運用部門の情報格差 - IT業界の裏話：新幹線トラブル

Ariane 5 打ち上げ失敗 (1996)

システム障害の仕様の超えるダイヤ変更が原因 - スラッシュドット・シ

せず能力超過 JR東 日本経済新聞 - 新幹線運行システム、1

17日に発生した新幹線輸送障害について (JR東日本 2011年1月18日)

因は処理限度値のオーバー - ITmedia News - JR東の新幹線システム

新幹線トラブル、運行担当者の誤解原因 JR東が謝罪 - 社会 - asahi.com (朝日新聞社) - 新幹線ト

因はシステムの処理容量オーバー - ニュース：ITpro - JR東日本の新幹線トラブル、原因はシステム

新幹線トラブル、大量のダイヤ変更が原因 JR東 日本経済新聞 - 新幹線トラブル、大量のダイヤ変更が原因 JR東

asahi.com (朝日新聞社)：人的ミスで運行管理ソフトに不具合 新幹線不通 - 社会 - asahi.com (朝日新聞社)：人的ミスで運行

新幹線のシステム障害は「自然復旧」原因不明、メーカーの技術員常駐で対応へ - ITmedia News - 新幹線のシステム障害は「自

東京新聞・新幹線ストップ 再発の懸念 障害原因不明のまま - 社会 (TOKYO Web) - 東京新聞・新幹線ストップ 再発の懸念 障害原

asahi.com (朝日新聞社)：新幹線ストップ、ソフトに不具合か 到着時刻表示されず - 社会 ■民営化後のゆうちょ銀行の主なシステム障害

時事ドットコム：「原因、復旧理由も分からず」=メーカー担当者常駐-JR東・新幹線 顧客情報管理システムの不具合で

【続報】三菱UFJ信託銀行のオンラインシステムが全面復旧も、原因は「調査中」 - ニュース 口座開設などの処理が一部遅延

でトラブル、ATMやインターネット取引が停止 - ニュース 機器の故障で企業顧客の送金処理

アートのバグが原因だった - Skypeの大規模ダウンは 約2500件、4億5000万円が遅延

新 | ブログ新聞批評 - 問題の本質伝えない岡崎図書館 4店舗でATMとゆうちょダイレ

きた - 都市修行僧 - 岡崎図書館事件のパネル討論会を クトに不具合。数十件規模の取引

記者の眼：ITpro - 「動かないコンピュータ」は怖く 21年8月24日 ATM約3000台でゆうちょ口座あ

転身で分かったこと - @IT - 大手ITからベンチャー 22年1月2日 ATM提携金融機関9社で最大約

電戦争！最前線!? - 気持ちよい生活を送ろう - 岡崎図 240件の取り扱いが不能に

男性、被書届取り下げ求める - マイタウン愛知 - asahi.com：図書館アクセスで起訴予告



5

# プログラム検証： 「意味論的」証明

6

## 正しさの証明

```

n := N;
k := 1;
while (n > 0) {
  k := k*n;
  n := n-1;
}

```

**N の階乗 N! を  
計算するプログラム**

**主張：**  
実行後の k の値は N!

**困難：**  
ループが何度回るかわからない

**アイデア：**  
ループの前後で保存される性質  
(ループ不変量) に注目

7

## 正しさの証明

```

n := N;
k := 1;
while (n > 0) {
  k := k*n;
  n := n-1;
}

```

**アイデア：**  
ループの前後で保存される性質  
(ループ不変量) に注目

k	n
1	N
N	N-1
N*(N-1)	N-2
N*(N-1)*(N-2)	N-3
...	...
N*(N-1)*...*2	1
N*(N-1)*...*2*1	0

**k \* (n!) = N!**

8

# 正しさの証明

```
n := N;
k := 1;
while (n > 0) {
  k := k*n;
  n := n-1;
}
```

ループの前 😊

ループの後 🍷

ループ不変量：

$k * (n!) = N!$

補題 1 :

$k * (n!) = N!$  は確かにループ不変量。  
すなわち, ☺ で成り立てば 🍷 でも成り立つ。

証明 :

ループの前と後の  $k, n$  の値をそれぞれ

$k_{old}, n_{old}$  と  $k_{new}, n_{new}$  と書くと,

(仮定 ☺)  $k_{old} * (n_{old}!) = N!$

(示したいこと)  $k_{new} * (n_{new}!) = N!$

いま,

$$\begin{aligned} k_{new} * (n_{new}!) &= (k_{old} * n_{old}) * ((n_{old} - 1)!) && [k_{new}, n_{new} \text{の定義より}] \\ &= k_{old} * (n_{old}!) && \\ &= N! && [\text{仮定 ☺ より}] \end{aligned}$$

よって成立. □

# 正しさの証明

```
n := N;
k := 1;
while (n > 0) {
  k := k*n;
  n := n-1;
}
```

ループの前 😊

ループの後 🍷

ループ不変量：

$k * (n!) = N!$

補題 1 :

$k * (n!) = N!$  は確かにループ不変量。  
すなわち, ☺ で成り立てば 🍷 でも成り立つ。

補題 2 :

$k * (n!) = N!$  はループに入る前 ☺ で成立。

証明 :

$n = N, k = 1$  より明らか. □

# 正しさの証明

```
n := N;
k := 1;
while (n > 0) {
  k := k*n;
  n := n-1;
}
```

ループの前 😊

ループの後 🍷

ループ不変量：

$k * (n!) = N!$

補題 1 :

$k * (n!) = N!$  は確かにループ不変量。  
すなわち, ☺ で成り立てば 🍷 でも成り立つ。

補題 2 :

$k * (n!) = N!$  はループに入る前 ☺ で成立。

プログラムの正しさの証明 :

補題 1, 2 より,  $k * (n!) = N!$  は  
ループの実行前, 実行中, 実行後  
全てで成立. ループの実行後は  $n=0$  なの  
で,  $k = N!$  でなければならない. □

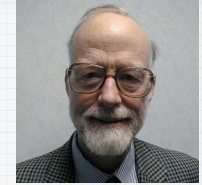
# 証明の本質をとらえて, 「機械化」

- \* 本質的な議論は単純?
- \* 代入による値の書き換え
- \* ループ不変量
- \* ... Hoare logic!

# Hoare 論理： プログラムの正しさを 証明する「機械」

13

## Hoare 論理



Sir Antony Hoare  
(1934.1.11-)  
Microsoft Research, Cambridge

- \* [Hoare, 1969]
- \* “Program logic” “Floyd-Hoare logic” と呼ばれる。
- \* 関連： Dynamic logic, Kleene algebra with tests
- \* Hoare triple を導いていく体系

$\{A\} P \{B\}$

例： $\{n=2\} n:=n+1 \{n=3\}$

実行前に成り立つ性質  
“precondition”

プログラム

実行後に成り立つ性質  
“postcondition”

14

## Hoare 論理の「材料」

- \* プログラム意味論
  - \* プログラム = メモリ状態の変換
- \* メモリ状態の性質を記述するための **assertion language**
- \* Hoare triple を導くための**導出規則** (ルール)
  - \* 機械的な, 文字列の書き換え

15

## プログラム意味論

16

# 「意味論」とは？

- \* プログラムの「意味」は何か
- \* 正確な答え：  
「実行した際の MacBook のメモリ状態の変化」
- \* → 細かすぎて「使えない」
- \* ここでは**メモリ状態**を用いる
- \* プログラミング言語による  
(宣言型言語だからメモリ状態を使う。たとえば関数型言語ならば関数として意味をつける)

```
n := N;
k := 1;
while (n > 0) {
  k := k*n;
  n := n-1;
}
```

# メモリ状態

- \* 変数と値の対応の表のこと。

$$\begin{bmatrix} x \mapsto 2 \\ y \mapsto 13 \\ \vdots \end{bmatrix}$$

- \* 数学的には：関数

$$\sigma : \text{Var} \longrightarrow \mathbb{Z}$$

# プログラムの意味

- \* メモリ状態の変換として
- \* つまり、関数  $\text{MSt} \longrightarrow \text{MSt} \cup \{\perp\}$

$\mapsto$  : 「変形する」「移す」

$$[x := a] : \sigma \mapsto \sigma[x \mapsto [a]\sigma]$$

プログラム  $x:=a$  の「意味」

メモリ状態、たとえば

$$\begin{bmatrix} x \mapsto 2 \\ y \mapsto 13 \\ \vdots \end{bmatrix}$$

アップデートされたメモリ状態。  
x の値を、a を  $\sigma$  のもとで計算した値 (たとえば  $[y + 1]\sigma = 14$  とか) にアップデート

x は変数、a は「数の表現」  
(たとえば y+1)

# プログラムの意味

- \* メモリ状態の変換として
- \* つまり、関数  $\text{MSt} \longrightarrow \text{MSt} \cup \{\perp\}$

$$[P_1; P_2] : \sigma \mapsto [P_2]([P_1]\sigma)$$

$P_1, P_2$  はプログラム

まず  $P_1$  によって変換

次に  $P_2$  によって変換

# プログラムの意味

- \* メモリ状態の変換として

- \* つまり、関数  $MSt \rightarrow MSt \cup \{\perp\}$

$\llbracket \text{if } b \text{ then } P_1 \text{ else } P_2 \rrbracket :$

$$\sigma \mapsto \begin{cases} \llbracket P_1 \rrbracket \sigma & \text{if } \llbracket b \rrbracket \sigma \text{ is true} \\ \llbracket P_2 \rrbracket \sigma & \text{if } \llbracket b \rrbracket \sigma \text{ is false} \end{cases}$$

$P_1, P_2$  はプログラム  
 $b$  は「真偽表現」  
 ( $x > 0$  とか)

21

# プログラムの意味

- \* メモリ状態の変換として

「停止しない」

- \* つまり、関数  $MSt \rightarrow MSt \cup \{\perp\}$

$\llbracket \text{while } b P \rrbracket : \sigma \mapsto ??$

- \* 状態変換の繰り返し  $\llbracket P \rrbracket^n \sigma$  を考える:

$\sigma, \llbracket P \rrbracket \sigma, \llbracket P \rrbracket (\llbracket P \rrbracket \sigma), \llbracket P \rrbracket (\llbracket P \rrbracket (\llbracket P \rrbracket \sigma)), \dots$

- \* ある時点で  $b$  が偽になれば、つまり  $\llbracket b \rrbracket (\llbracket P \rrbracket^n \sigma) = \text{false}$  なら、そうなるような最初の  $n$  に対する  $\llbracket P \rrbracket^n \sigma$  を返す
- \*  $b$  がずっと真であれば、 $\perp$  (未定義, 非停止)

22

# プログラムの意味論： まとめ

- \* メモリ状態の変換として

「停止しない」

- \* つまり、関数  $MSt \rightarrow MSt \cup \{\perp\}$

$\llbracket x := a \rrbracket : \sigma \mapsto \sigma[x \mapsto \llbracket a \rrbracket \sigma]$

$\llbracket P_1; P_2 \rrbracket : \sigma \mapsto \llbracket P_2 \rrbracket (\llbracket P_1 \rrbracket \sigma)$

$\llbracket \text{if } b \text{ then } P_1 \text{ else } P_2 \rrbracket :$

$$\sigma \mapsto \begin{cases} \llbracket P_1 \rrbracket \sigma & \text{if } \llbracket b \rrbracket \sigma \text{ is true} \\ \llbracket P_2 \rrbracket \sigma & \text{if } \llbracket b \rrbracket \sigma \text{ is false} \end{cases}$$

$\llbracket \text{while } b P \rrbracket : \sigma \mapsto \dots$

ポイント：

- \* 数学的に厳密な定義
- \* 要素還元的 (大きなプログラムの意味は、その部品の意味から決まる)

23

# Hoare 論理の「材料」 (思い出そう)

- \* プログラム意味論
- \* プログラム = メモリ状態の変換
- \* メモリ状態の性質を記述するための **assertion language**
- \* Hoare triple を導くための **導出規則**

24

# Assertion Language

25

# Assertion Language

\* **Assertion**: メモリ状態の性質を記述する「論理式」。例:

\*  $x = 5 \wedge y \leq 3$

\*  $\exists z. (x = 2 * z \wedge y = 3 * z)$

\* 本当ならば, 言語 (文法) をはっきり定めることが望ましい。たとえば:

$\text{AExp} \ni a ::= x \mid n \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2$   
 $\text{Fml} \ni A ::= \text{true} \mid \text{false} \mid A_1 \wedge A_2 \mid \neg A \mid a_1 < a_2 \mid \forall x \in \mathbb{N}. A \quad \text{where } x \in \text{Var}$

\* とくに, 証明の形式化・自動化のためには必須

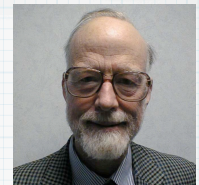
\* ここではやらない (時間が無いから)

26

# Hoare 論理の 導出規則

27

# Hoare 論理



Sir Antony Hoare  
(1934.1.11-)

Microsoft Research, Cambridge

\* [Hoare, 1969]

\* “Program logic” “Floyd-Hoare logic” と呼ばれる。

\* 関連: Dynamic logic, Kleene algebra with tests

\* **Hoare triple** を導いていく体系

$\{A\} P \{B\}$

例:  $\{n=2\} n:=n+1 \{n=3\}$

実行前に成り立つ性質  
“precondition”

プログラム

実行後に成り立つ性質  
“postcondition”

28

# Hoare Triple の意味論

実行前に成り立つ性質  
“precondition” を表す  
assertion

実行後に成り立つ性質  
“postcondition” を表す  
assertion

{A} P {B}

プログラム

- \* 「{A} P {B} が真」であるとは、
- \* assertion A をみたす任意のメモリ状態  $\sigma$  について、  
実行後のメモリ状態
- \* メモリ状態  $\llbracket P \rrbracket \sigma$  が assertion B をみたすことをいう。

# Partial Correctness

{A} P {B}

- \* 「{A} P {B} が真」であるとは、
- \* assertion A をみたす任意のメモリ状態  $\sigma$  について、
- \* メモリ状態  $\llbracket P \rrbracket \sigma$  が assertion B をみたすことをいう。  
↓  
だったらどうする？
- \* 「↓ はすべての assertion をみたす」と約束。
- \* → P が停止しないときに関しては、何も保証してくれない (“partial correctness”)
- \* そもそも「P が停止するか？」はとても難しい問題。  
決定不能！（判定してくれるプログラムは存在しない）

# 真な Hoare Triple の例

{ x=2 } x := x+1 { x=3 }  
 { x=2 } x := x+1; y := x { y=3 }  
 { } if x > 0 then x := x-1 else x := 0 { x ≥ 0 }  
 { x > 0 } while x > 0 { y := x } { x = x+1 }

停止しないので

postcondition が何であっても、この Hoare triple は真

# Hoare 論理の導出規則

規則の読み方：  
「上の仮定が成立していれば、下の結論も成立」  
(この規則では仮定はなし)

{ A[a/x] } x := a { A } (Assign)

規則の名前  
(assignment = 「値の割り当て」)

assertion A において、変数 x を a に書き換えたもの

- \* 結論 { A[a/x] } x := a { A } は、右から左に読むとよい。
- \* 「x := a の後に A が成り立つためには、実行前には A[a/x] が成り立たないといけない」



# Hoare 論理の導出規則

規則の読み方：  
「上の仮定が成立していれば、下の結論も成立」  
(この規則では仮定はなし)

$$\frac{\{A[a/x]\} x:=a \{A\}}{\text{(Assign)}}$$

assertion A において、変数 x を a に書き換えたもの

規則の名前  
(assignment = 「値の割り当て」)

\* 例：

$$\frac{\frac{\{y=2\} x:=y \{x=2\}}{\text{(Assign)}} \quad \frac{\{x-1=2\} x:=x-1 \{x=2\}}{\text{(Assign)}}}{\{k*((n-1)!)=N! \wedge n-1 \geq 0\} n:=n-1 \{k*(n!)=N! \wedge n \geq 0\}}{\text{(SeqComp)}}$$

# Hoare 論理の導出規則

規則の読み方：  
「上の仮定が成立していれば、下の結論も成立」

$$\frac{\{A\} P_1 \{C\} \quad \{C\} P_2 \{B\}}{\{A\} P_1; P_2 \{B\}} \text{(SeqComp)}$$

規則の名前  
(sequential composition = 「逐次合成」)

- \* (もう一回言っておく) 次のように読む。
- \* 規則は上から下に (上が成り立てば下が成り立つ)
- \* 証明を探すときは逆 (下が成り立つためには、どんな上が成り立てばいい?)
- \* Hoare triple は右から左に (postcondition が成り立つためには、. . .)

# Hoare 論理の導出規則

規則の読み方：  
「上の仮定が成立していれば、下の結論も成立」

$$\frac{\{A\} P_1 \{C\} \quad \{C\} P_2 \{B\}}{\{A\} P_1; P_2 \{B\}} \text{(SeqComp)}$$

規則の名前  
(sequential composition = 「逐次合成」)

\* 例：

$$\frac{\frac{\{x-1=2\} x:=y \{y-1=2\}}{\text{(Assign)}} \quad \frac{\{y-1=2\} y:=y-1 \{y=2\}}{\text{(Assign)}}}{\{x-1=2\} x:=y; y:=y-1 \{y=2\}}{\text{(SeqComp)}}$$

# さっきの例, くわしく!!!

$$\frac{\frac{\{x-1=2\} x:=y \{y-1=2\}}{\text{(Assign)}} \quad \frac{\{y-1=2\} y:=y-1 \{y=2\}}{\text{(Assign)}}}{\{x-1=2\} x:=y; y:=y-1 \{y=2\}}{\text{(SeqComp)}}$$

Step 1:  
「実行後に y=2 になってほしいけど、. . .」

$$\frac{\frac{\{x-1=2\} x:=y \{y-1=2\}}{\text{(Assign)}} \quad \frac{\{y-1=2\} y:=y-1 \{y=2\}}{\text{(SeqComp)}}}{\{x-1=2\} x:=y; y:=y-1 \{y=2\}}{\text{(SeqComp)}}$$

Step 2:  
「y:=y-1 の前には y-1=2 であればいいぞ」

$$\frac{\frac{\{x-1=2\} x:=y \{y-1=2\}}{\text{(Assign)}} \quad \frac{\{y-1=2\} y:=y-1 \{y=2\}}{\text{(Assign)}}}{\{x-1=2\} x:=y; y:=y-1 \{y=2\}}{\text{(SeqComp)}}$$

Step 3:  
「さらに x:=y の前には x-1=2 であればいいぞ」

$$\frac{\frac{\{x-1=2\} x:=y \{y-1=2\}}{\text{(Assign)}} \quad \frac{\{y-1=2\} y:=y-1 \{y=2\}}{\text{(SeqComp)}}}{\{x-1=2\} x:=y; y:=y-1 \{y=2\}}{\text{(SeqComp)}}$$

Step 4:  
「x-1=2 であるための precondition, わかった!」

← 考えの流れ

# Hoare 論理の導出規則

規則の読み方：  
「上の仮定が成立し  
ていれば、下の結論  
も成立」

$$\frac{\{A\} P_1 \{C\} \quad \{C\} P_2 \{B\}}{\{A\} P_1; P_2 \{B\}} \text{ (SeqComp)}$$

規則の名前  
(sequential composition =  
「逐次合成」)

\* 例：

$$\frac{\left\{ \begin{array}{l} k^* \cdot ((n-1)!) = N! \\ \wedge n-1 \geq 0 \end{array} \right\} k := k * n \quad \left\{ \begin{array}{l} k^* \cdot ((n-1)!) = N! \\ \wedge n-1 \geq 0 \end{array} \right\} \quad \left\{ \begin{array}{l} k^* \cdot ((n-1)!) = N! \\ \wedge n-1 \geq 0 \end{array} \right\} n := n-1 \quad \left\{ \begin{array}{l} k^* \cdot (n!) = N! \\ \wedge n \geq 0 \end{array} \right\}}{\left\{ \begin{array}{l} k^* \cdot ((n-1)!) = N! \\ \wedge n-1 \geq 0 \end{array} \right\} k := k * n; \quad \left\{ \begin{array}{l} k^* \cdot (n!) = N! \\ \wedge n \geq 0 \end{array} \right\}} \text{ (SeqComp)}$$

37

# Hoare 論理の導出規則

$$\frac{\{A \wedge b\} P_1 \{B\} \quad \{A \wedge \neg b\} P_2 \{B\}}{\{A\} \text{ if } b \text{ then } P_1 \text{ else } P_2 \{B\}} \text{ (If)}$$

\* 例：

$$\frac{\{x \geq 0 \wedge x > 0\} x := x-1 \{x \geq 0\} \quad \{x \geq 0 \wedge \neg(x > 0)\} x := x \{x \geq 0\}}{\{x \geq 0\} \text{ if } x > 0 \text{ then } x := x-1 \text{ else } x := x \{x \geq 0\}} \text{ (If)}$$

38

# Hoare 論理の導出規則

A はループ不変量！

$$\frac{\{A \wedge b\} P_1 \{A\}}{\{A\} \text{ while } b \text{ P}_1 \{A \wedge \neg b\}} \text{ (While)}$$

ループを脱出した →  
b は成立しないはず

\* 例：

$$\frac{\{x \geq 0 \wedge x > 0\} x := x-1 \{x \geq 0\}}{\{x \geq 0\} \text{ while } x > 0 ( x := x-1 ) \{x \geq 0 \wedge \neg(x > 0)\}} \text{ (While)}$$

39

# Hoare 論理の導出規則

A はループ不変量！

$$\frac{\{A \wedge b\} P_1 \{A\}}{\{A\} \text{ while } b \text{ P}_1 \{A \wedge \neg b\}} \text{ (While)}$$

ループを脱出した →  
b は成立しないはず

\* 例：

$$\frac{\left\{ \begin{array}{l} k^* \cdot (n!) = N! \\ \wedge n > 0 \end{array} \right\} k := k * n; \quad n := n-1 \quad \left\{ \begin{array}{l} k^* \cdot (n!) = N! \\ \wedge n > 0 \end{array} \right\}}{\left\{ \begin{array}{l} k^* \cdot (n!) = N! \\ \wedge n > 0 \end{array} \right\} \text{ while } (n > 0) \quad k := k * n; \quad n := n-1 \quad \left\{ \begin{array}{l} k^* \cdot (n!) = N! \\ \wedge n = 0 \end{array} \right\}} \text{ (While)}$$

40

# Hoare 論理の導出規則

$$\frac{A \Rightarrow A' \quad \{A'\} P \{B'\} \quad B' \Rightarrow B}{\{A\} P \{B\}} \text{(Conseq)}$$

仮定：「A が成り立てば、必ず A' も成り立つ」

規則の名前  
(consequence = 「論理的帰結」)

\* 例：

$$\frac{x > 0 \Rightarrow (x \geq 0 \wedge x > 0) \quad \{x \geq 0 \wedge x > 0\} x := x - 1 \{x \geq 0\}}{\{x > 0\} x := x - 1 \{x \geq 0\}} \text{(Conseq)}$$

# Hoare 論理の導出規則

$$\frac{A \Rightarrow A' \quad \{A'\} P \{B'\} \quad B' \Rightarrow B}{\{A\} P \{B\}} \text{(Conseq)}$$

仮定：「A が成り立てば、必ず A' も成り立つ」

規則の名前  
(consequence = 「論理的帰結」)

$$\frac{\begin{array}{l} k^*(n!) = N! \\ \wedge n > 0 \\ \Rightarrow \\ k^*n^*((n-1)!) = N! \\ \wedge n-1 \geq 0 \end{array} \quad \left\{ \begin{array}{l} k^*n^*((n-1)!) = N! \\ \wedge n-1 \geq 0 \end{array} \right\} \quad \begin{array}{l} k := k*n; \\ n := n-1 \end{array} \quad \left\{ \begin{array}{l} k^*(n!) = N! \\ \wedge n \geq 0 \end{array} \right\}}{\left\{ \begin{array}{l} k^*(n!) = N! \\ \wedge n > 0 \end{array} \right\} \quad \begin{array}{l} k := k*n; \\ n := n-1 \end{array} \quad \left\{ \begin{array}{l} k^*(n!) = N! \end{array} \right\}} \text{(Conseq)}$$

## Hoare 論理の導出規則 (まとめ)

$$\frac{}{\{A[a/x]\} x := a \{A\}} \text{(Assign)}$$

$$\frac{\{A\} P_1 \{C\} \quad \{C\} P_2 \{B\}}{\{A\} P_1; P_2 \{B\}} \text{(SeqComp)}$$

$$\frac{\{A \wedge b\} P_1 \{B\} \quad \{A \wedge \neg b\} P_2 \{B\}}{\{A\} \text{if } b \text{ then } P_1 \text{ else } P_2 \{B\}} \text{(If)}$$

$$\frac{\{A \wedge b\} P_1 \{A\}}{\{A\} \text{while } b \text{ } P_1 \{A \wedge \neg b\}} \text{(While)}$$

## Hoare 論理の導出規則 (まとめ, つづき)

$$\frac{A \Rightarrow A' \quad \{A'\} P \{B'\} \quad B' \Rightarrow B}{\{A\} P \{B\}} \text{(Conseq)}$$

# Hoare 論理における導出

# Hoare 論理の導出

\* たとえば、次の Hoare triple を証明したい。

$$\{ k=1 \wedge n=N \} \text{ while } (n>0) \{ k=N! \}$$

$k:=k*n;$   
 $n:=n-1$

\* 「意味論的立場」

- \* 「 $\{A\} P \{B\}$  が真」であるとは、
- \* assertion A をみたす任意のメモリ状態  $\sigma$  について、
- \* メモリ状態  $\llbracket P \rrbracket \sigma$  が assertion B をみたすことをいう。
- \* 直接の証明
- \* 「数学の証明」, 頭を使う。  
→ 自動化できない!

\* 「構文論的立場」

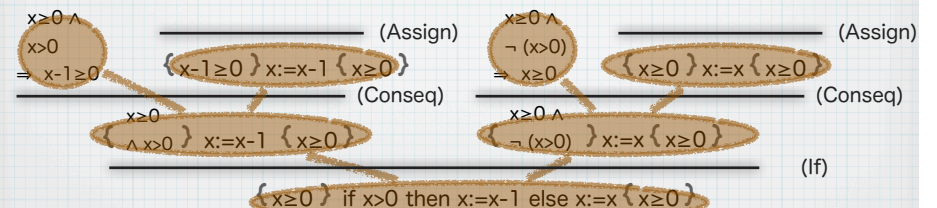
- \* Hoare 論理の導出規則を繰り返し使って、仮定なしの「導出木」(「証明木」)が作れればOK!
- \* 記号操作, 証明「検索」  
→ 自動化!

# Hoare 論理の導出

$$\frac{\frac{x \geq 0 \wedge x > 0}{\Rightarrow x-1 \geq 0} \quad \frac{x \geq 0 \wedge \neg(x > 0)}{\Rightarrow x \geq 0}}{\frac{\{x-1 \geq 0\} x:=x-1 \{x \geq 0\} \quad \{x \geq 0\} x:=x \{x \geq 0\}}{\{x \geq 0\} x:=x-1 \{x \geq 0\}} \text{ (Conseq)} \quad \frac{\{x \geq 0\} x:=x \{x \geq 0\}}{\{x \geq 0\} x:=x \{x \geq 0\}} \text{ (Assign)}}{\frac{\{x \geq 0 \wedge x > 0\} x:=x-1 \{x \geq 0\} \quad \{x \geq 0 \wedge \neg(x > 0)\} x:=x \{x \geq 0\}}{\{x \geq 0\} \text{ if } x > 0 \text{ then } x:=x-1 \text{ else } x:=x \{x \geq 0\}} \text{ (If)}$$

# Hoare 論理の導出

木!!  
(導出木, 証明木)



# Hoare 論理の導出

- \* たとえば、次の Hoare triple を証明したい。

$$\{ k=1 \wedge n=N \} \text{ while } (n>0) \{ k=N! \}$$

$k:=k*n;$   
 $n:=n-1$

## \* 「意味論的立場」

- \* 「 $\{A\} P \{B\}$  が真」であるとは、
- \* assertion A をみたす任意のメモリ状態  $\sigma$  について、
- \* メモリ状態  $\llbracket P \rrbracket \sigma$  が assertion B をみたすことをいう。
- \* 直接の証明
- \* 「数学の証明」、  
頭を使う。  
→ 自動化できない！

## \* 「構文論的立場」

- \* Hoare 論理の導出規則を繰り返し使って、仮定なしの「導出木」（「証明木」）が作れればOK！
- \* 記号操作、証明「検索」  
→ 自動化！

49

# 健全性, 完全性

- \* 意味論と構文論のせめぎあい！！  
(導出規則=構文論=「機械」)
- \* **健全性 soundness:** 導出規則で導かれる Hoare triple は、すべて真
  - \* 「ウソは言わない」
  - \* 「証明能力が高すぎない」
  - \* この性質は必須。(ウソをつかれると困る)
- \* **完全性 completeness:** 真である Hoare triple は、すべて導出規則で導ける
  - \* 「真なことはすべて言ってくれる」
  - \* 「証明能力が十分高い」
  - \* これは成り立たない場合も多い(仕方ない、不完全性定理)

50

# 健全性, 完全性

- \* Hoare 論理は
  - \* 健全性を満たす。
  - \* 相対完全性 relative completeness を満たす。
    - \* 相対完全性 = 「条件付き」完全性
    - \* 条件付きでない、フルの完全性は、Gödel の不完全性定理により排除される。

51

# 例：階乗の計算

52

# はじめの例

```

n := N;
k := 1;
while (n > 0) {
  k := k*n;
  n := n-1;
}
    
```

主張：  
実行後の k の値は N!

この証明を、Hoare 論理を使ってシステムティックに！

(構文論的・機械的な、記号書き換えによる証明)

N の階乗 N! を  
計算するプログラム

# Hoare 論理による証明

\* 目標： 次の Hoare triple の導出

$$\{ k=1 \wedge n=N \} \text{ while } (n>0) \{ k=N! \}$$

$k:=k*n;$   
 $n:=n-1$

# Hoare 論理による証明

$$\frac{\frac{\frac{\{ k^*n*((n-1)!) = N! \wedge n-1 \geq 0 \}}{k:=k*n} \frac{\{ k^*((n-1)!) = N! \wedge n-1 \geq 0 \}}{n:=n-1} \frac{\{ k^*(n!) = N! \wedge n \geq 0 \}}{k^*(n!) = N! \wedge n \geq 0}}{\text{(SeqComp)}}}{\frac{\{ k^*n*((n-1)!) = N! \wedge n-1 \geq 0 \}}{k:=k*n; n:=n-1} \frac{\{ k^*(n!) = N! \wedge n \geq 0 \}}{k^*(n!) = N! \wedge n \geq 0}}{\text{(Conseq)}}}$$

$$\frac{\frac{\{ k^*(n!) = N! \wedge n \geq 0 \}}{k:=k*n; n:=n-1} \frac{\{ k^*(n!) = N! \wedge n \geq 0 \}}{k^*(n!) = N! \wedge n \geq 0}}{\text{(While)}}}{\frac{\{ k=1 \wedge n=N \}}{\Rightarrow \{ k^*(n!) = N! \wedge n \geq 0 \}} \frac{\text{while } (n>0) \{ k:=k*n; n:=n-1 \} \{ k^*(n!) = N! \wedge n \geq 0 \}}{\Rightarrow k=N!}}{\text{(Conseq)}}}$$

$$\frac{\{ k=1 \wedge n=N \} \text{ while } (n>0) \{ k:=k*n; n:=n-1 \} \{ k=N! \}}$$

(頭を使わずに、規則を見てパターンマッチングでできるぞ...)

最初にやった数学的証明の「形式化」(エッセンスは同じ)

$$\frac{\frac{\frac{\{ k^*n*((n-1)!) = N! \wedge n-1 \geq 0 \}}{k:=k*n} \frac{\{ k^*((n-1)!) = N! \wedge n-1 \geq 0 \}}{n:=n-1} \frac{\{ k^*(n!) = N! \wedge n \geq 0 \}}{k^*(n!) = N! \wedge n \geq 0}}{\text{(SeqComp)}}}{\frac{\{ k^*n*((n-1)!) = N! \wedge n-1 \geq 0 \}}{k:=k*n; n:=n-1} \frac{\{ k^*(n!) = N! \wedge n \geq 0 \}}{k^*(n!) = N! \wedge n \geq 0}}{\text{(Conseq)}}}$$

$k^*(n!) = N!$   
 $\wedge n \geq 0$

$k:=k*n;$   
 $n:=n-1$

$k^*(n!) = N!$   
 $\wedge n \geq 0$

**ループ不変量!**

$$\frac{\frac{\{ k^*(n!) = N! \wedge n \geq 0 \}}{k:=k*n; n:=n-1} \frac{\{ k^*(n!) = N! \wedge n \geq 0 \}}{k^*(n!) = N! \wedge n \geq 0}}{\text{(While)}}}{\frac{\{ k=1 \wedge n=N \}}{\Rightarrow \{ k^*(n!) = N! \wedge n \geq 0 \}} \frac{\text{while } (n>0) \{ k:=k*n; n:=n-1 \} \{ k^*(n!) = N! \wedge n \geq 0 \}}{\Rightarrow k=N!}}{\text{(Conseq)}}}$$

$$\frac{\{ k=1 \wedge n=N \} \text{ while } (n>0) \{ k:=k*n; n:=n-1 \} \{ k=N! \}}$$

# 参考文献

- \* G. Winskel, The Formal Semantics of Programming Languages: An Introduction. The MIT Press, ISBN 0-262-23169-7

57

# 自己紹介

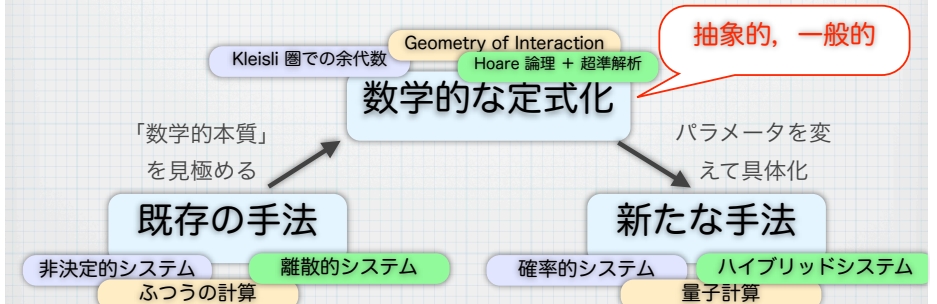
58

- \* 蓮尾 一郎 (はすお・いちろう)
- \* BSc (東大数学, 2002)  
MSc (東工大情報, 2004)  
PhD (U. Nijmegen, 2008)
- \* 京都大学数理解析研究所 助教 (2007-2011)  
東京大学理学部情報科学科 講師 (2011-)
- \* 専門分野: 数学と計算機科学を行ったり来たり
- \* 理学部情報科学科での立ち位置:
  - \* 応用 vs. 理論
  - \* 速度 vs. 正しさ
- \* <http://www-mmm.is.s.u-tokyo.ac.jp/~ichiro>

59

# 研究テーマ

- \* 抽象数学をつかって, 情報科学の新パラダイムに斬り込む
- \* 特に: 圏論, 数理論理学, 代数学, 幾何学



60

# メッセージ

- \* 情報科学と数学のインタラクション, おもしろいよ!
- \* そもそもコンピュータは数学(数学基礎論)から生まれた

61

# お待ちかね： レポート課題

62

# レポート課題

1. 次の4つの Hoare triple について, 真であるかどうか判定せよ. (証明・反例があればなおよい)

$\{ x=3 \} x:=y \{ y=3 \}$

$\{ y=2 \} x:=y+1; y:=x \{ x=3 \}$

$\{ \} \text{if } x \geq 0 \text{ then } x:=x-1 \text{ else } x:=0 \{ x \geq 0 \}$

$\{ X \geq 0 \}$  `y:=0; x:=X;  
while (x>100) {  
  y:=y+1;  
  x:=x-100  
}`  $\left\{ \begin{array}{l} 0 \leq x < 100 \wedge \\ 100 * y + x = X \end{array} \right\}$

気持ち: X セントを「y ドル x セント」に変換するプログラム 63

# レポート課題

2. 次の2つの Hoare triple に対して, Hoare 論理の導出木を与えよ. ただし x, n, m は整数(int)型の変数とする.

$\{ x \geq 0 \} \text{if } x > 0 \text{ then } x:=x-1 \text{ else } x:=x \{ x \geq 0 \}$

ヒント: 38 ページのルールの適用 (まだ仮定が残ってる!) を完成させる.

$\{ \exists m (m \geq 0 \wedge n = 2 * m) \} \text{while } (n > 0) \{ n:=n-2 \} \{ n=0 \}$

「n は負でない偶数」

64



# レポート課題

3. [Hoare 論理とは関係ない算数の問題]  
2つの正の整数  $m, n$  (ただし  $m > n$ ) について,

$$\text{GCD}(m, n) = \text{GCD}(m - n, n)$$

であることを示せ。ただし,  $\text{GCD}(m, n)$  は  $m$  と  $n$  の最大公約数を表す。

65

# レポート課題

4. 正の整数  $X, Y$  の最大公約数を計算するための, 次のプログラム  $P$  を考える (ユークリッドの互除法)。

```
x:=X; y:=Y;
while (x≠y) {
  if x≥y
    then x:=x-y
  else y:=y-x
}
```

ヒント:

- ・ 55ページを参考に。
- ・ レポート課題の問題3を使う。
- ・ ループ不変量は,  
 $x > 0 \wedge y > 0 \wedge \text{GCD}(x, y) = \text{GCD}(X, Y)$

次の Hoare triple の, Hoare 論理における導出木を与えよ。

$$\{X > 0 \wedge Y > 0\} P \{x = y = \text{GCD}(X, Y)\}$$

66

# 念のため

- \* 全問解かなくても大丈夫だが, たくさん解いたほうがよい (当たり前)
- \* コピーはダメ!
  - \* 東北大学 小林直樹先生のページを参照  
<http://www.kb.ecei.tohoku.ac.jp/~koba/report.html>
  - \* 盗作と判断された場合には, 深刻な結果に至る場合があります。
    - \* たとえば, 法学部では一発退学

67