(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

東京大学 THE UNIVERSITY OF TOKYO

プログラム検証：最初の例

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

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

ループ脱出！
(n>0 でない)

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，...

プログラム検証：「意味論的」証明

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

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!) = N!

	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

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,

$k_{new} * (n_{new}\,!)$

$= (k_{old} * n_{old}) * ((n_{old} - 1)!)$　[k_{new}, n_{new}の定義より]

$= k_{old} * (n_{old}\,!)$

$= N!$　[仮定☺より]

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. □

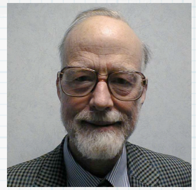
証明の本質をとらえて，「機械化」

- 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」

Hoare 論理



Sir Antony Hoare
(1934.1.11-)
Microsoft Research, Cambridge

* [Hoare, 1969]
* “Program logic” “Floyd-Hoare logic” とも呼ばれる.
* 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”

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

プログラム意味論

「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」

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

* 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）

メモリ状態

* 変数と値の対応の表のこと．

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

* 数学的には： 関数

$$\boldsymbol{\sigma} : \mathbf{Var} \longrightarrow \mathbb{Z}$$

プログラムの意味

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

$\mapsto$ ：「変形する」「移す」

$$[\![\mathrm{x} := a]\!] \;:\; \boldsymbol{\sigma} \longmapsto \boldsymbol{\sigma}\left[\mathrm{x} \mapsto [\![a]\!]\boldsymbol{\sigma}\right]$$

プログラム x:=a の「意味」

メモリ状態，
たとえば
$$\begin{bmatrix} \mathrm{x} & \mapsto & \mathbf{2} \\ \mathrm{y} & \mapsto & \mathbf{13} \\ & \vdots & \end{bmatrix}$$

アップデートされたメモリ状態．
x の値を，a を σ のもとで計算した値（たとえば
$$[\![\mathrm{y}+\mathbf{1}]\!]\boldsymbol{\sigma} = \mathbf{14}$$
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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\perp\}$

$$[\![P_1; P_2]\!] \;:\; \boldsymbol{\sigma} \longmapsto [\![P_2]\!]([\![P_1]\!]\boldsymbol{\sigma})$$

まず P_1 によって変換

次に P_2 によって変換

P_1，P_2 はプログラム

プログラムの意味

* メモリ状態の変換として
  * つまり，関数　$\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\bot\}$

$$[\![\texttt{if}\ b\ \texttt{then}\ P_1\ \texttt{else}\ P_2]\!]\ :\quad \sigma \longmapsto \begin{cases} [\![P_1]\!]\sigma & \text{if } [\![b]\!]\sigma \text{ is true} \\ [\![P_2]\!]\sigma & \text{if } [\![b]\!]\sigma \text{ is false} \end{cases}$$

P_1，P_2 はプログラム
b は「真偽表現」
（x > 0 とか）

プログラムの意味

* メモリ状態の変換として
  * つまり，関数　$\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\bot\}$

「停止しない」

$$[\![\texttt{while}\ b\ P]\!]\ :\quad \sigma \longmapsto ??$$

* 状態変換の繰り返し $[\![P]\!]^n\sigma$を考える：

$$\sigma,\ [\![P]\!]\sigma,\quad [\![P]\!]([\![P]\!]\sigma),\quad [\![P]\!]([\![P]\!]([\![P]\!]\sigma)),\quad \ldots$$

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

プログラムの意味論：まとめ

* メモリ状態の変換として
  * つまり，関数 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\bot\}$

「停止しない」

$$[\![\mathrm{x} := a]\!]\ :\quad \sigma \longmapsto \sigma[\,\mathrm{x} \mapsto [\![a]\!]\sigma\,]$$

$$[\![P_1 ; P_2]\!]\ :\quad \sigma \longmapsto [\![P_2]\!]([\![P_1]\!]\sigma)$$

$$[\![\texttt{if}\ b\ \texttt{then}\ P_1\ \texttt{else}\ P_2]\!]\ :\quad \sigma \longmapsto \begin{cases} [\![P_1]\!]\sigma & \text{if } [\![b]\!]\sigma \text{ is true} \\ [\![P_2]\!]\sigma & \text{if } [\![b]\!]\sigma \text{ is false} \end{cases}$$

$$[\![\texttt{while}\ b\ P]\!]\ :\quad \sigma \longmapsto \ldots$$

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）

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

Assertion Language

Assertion Language

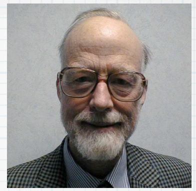
* Assertion: メモリ状態の性質を記述する「論理式」．例：
  * $x = 5 \land y \leq 3$
  * $\exists z.\ (x = 2 * z \land y = 3 * z)$
* 本当ならば，言語（文法）をはっきり定めることが望ましい．たとえば：

$$
\begin{array}{lll}
\mathbf{AExp} \ni & a & ::= \quad x \mid n \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \\
\mathbf{Fml} \ni & A & ::= \quad \mathtt{true} \mid \mathtt{false} \mid A_1 \land A_2 \mid \neg A \mid a_1 < a_2 \mid \\
& & \qquad \forall x \in \mathbb{N}.\ A \qquad \text{where } x \in \mathbf{Var}
\end{array}
$$

  * 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（時間がないから）

Hoare 論理の導出規則

Hoare 論理



Sir Antony Hoare
(1934.1.11-)
Microsoft Research, Cambridge

* [Hoare, 1969]
* “Program logic” “Floyd-Hoare logic” とも呼ばれる．
* 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”

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 σ について,
  * メモリ状態 ⟦P⟧σ が assertion B をみたすことをいう.

実行後のメモリ状態

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 σ について,
  * メモリ状態 ⟦P⟧σ 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）

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

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」）

* 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] が成り立てばよい」

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：

$$\frac{}{\{y=2\}\ x:=y\ \{x=2\}} \qquad \frac{}{\{x-1=2\}\ x:=x-1\ \{x=2\}}$$

$$\frac{}{\{k*((n-1)!)=N! \land n-1\geq 0\}\ n:=n-1\ \{k*(n!)=N! \land n\geq 0\}}$$

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」

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

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」

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

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

* 例：

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

さっきの例，くわしく！！

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

Step 1:
「実行後に y=2 になってほしんだけど．．．」

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

Step 2:
「y:=y-1 の前には y-1=2 であればいいぞ」

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

Step 3:
「さらに x:=y の前には x-1=2 であればいいぞ」

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

考えの流れ

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

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」

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

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

* 例：

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

Hoare 論理の導出規則

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

* 例：

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

Hoare 論理の導出規則

A はループ不変量！

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

ループを脱出した ➜ b は成立しないはず

* 例：

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

Hoare 論理の導出規則

A はループ不変量！

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

ループを脱出した ➜ b は成立しないはず

* 例：

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

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」）

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

Hoare 論理の導出規則（まとめ）

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

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

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

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

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．

$$\{ k=1 \land n=N \} \quad \begin{array}{l}\text{while } (n>0)\\ \quad k:=k*n;\\ \quad n:=n-1\end{array} \quad \{ k = N! \}$$

* 「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 σ について,
  * メモリ状態 $[\![P]\!]\sigma$ 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

$$\dfrac{\dfrac{x\geq 0 \land x>0 \Rightarrow x-1\geq 0 \qquad \dfrac{}{\{x-1\geq 0\}\ x:=x-1\ \{x\geq 0\}}\,(\text{Assign})}{\{x\geq 0 \land x>0\}\ x:=x-1\ \{x\geq 0\}}\,(\text{Conseq}) \qquad \dfrac{x\geq 0 \land \neg(x>0) \Rightarrow x\geq 0 \qquad \dfrac{}{\{x\geq 0\}\ x:=x\ \{x\geq 0\}}\,(\text{Assign})}{\{x\geq 0 \land \neg(x>0)\}\ x:=x\ \{x\geq 0\}}\,(\text{Conseq})}{\{x\geq 0\}\ \text{if } x>0 \text{ then } x:=x-1 \text{ else } x:=x\ \{x\geq 0\}}\,(\text{If})$$

Hoare 論理の導出

木！！

（導出木，証明木）

$$\dfrac{\dfrac{x\geq 0 \land x>0 \Rightarrow x-1\geq 0 \qquad \dfrac{}{\{x-1\geq 0\}\ x:=x-1\ \{x\geq 0\}}\,(\text{Assign})}{\{x\geq 0 \land x>0\}\ x:=x-1\ \{x\geq 0\}}\,(\text{Conseq}) \qquad \dfrac{x\geq 0 \land \neg(x>0) \Rightarrow x\geq 0 \qquad \dfrac{}{\{x\geq 0\}\ x:=x\ \{x\geq 0\}}\,(\text{Assign})}{\{x\geq 0 \land \neg(x>0)\}\ x:=x\ \{x\geq 0\}}\,(\text{Conseq})}{\{x\geq 0\}\ \text{if } x>0 \text{ then } x:=x-1 \text{ else } x:=x\ \{x\geq 0\}}\,(\text{If})$$

Hoare 論理の導出

* たとえば，次の Hoare triple を証明したい．

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

* 「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 σ について,
  * メモリ状態 ⟦ P ⟧σ 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！

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）

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．

例： 階乗の計算

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!

N の階乗 N! を
計算するプログラム

この証明を，Hoare 論理を使ってシステマティックに！
（構文論的・機械的な，記号書き換えによる証明）

Hoare 論理による証明

* 目標： 次の Hoare triple の導出

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

Hoare 論理による証明

{ k*n*((n-1)!)=N! ∧ n-1≥0 } k:=k*n { k*((n-1)!)=N! ∧ n-1≥0 } (Assign)

{ k*((n-1)!)=N! ∧ n-1≥0 } n:=n-1 { k*(n!)=N! ∧ n≥0 } (Assign)

(SeqComp)

k*(n!)=N! ∧ n≥0 ∧ n>0 ⇒ k*n*((n-1)!)=N! ∧ n-1≥0

{ k*n*((n-1)!)=N! ∧ n-1≥0 } k:=k*n; n:=n-1 { k*(n!)=N! ∧ n≥0 }

(Conseq)

{ k*(n!)=N! ∧ n≥0 ∧ n>0 } k:=k*n; n:=n-1 { k*(n!)=N! ∧ n≥0 }

(While)

k=1 ∧ n=N ⇒ k*(n!) = N! ∧ n≥0

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

k*(n!)=N! ∧ n≥0 ∧ ¬(n>0) ⇒ k=N!

(Conseq)

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

Hoare 論理による証明

（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）

{ k*n*((n-1)!)=N! ∧ n-1≥0 } k:=k*n { k*((n-1)!)=N! ∧ n-1≥0 } (Assign)

{ k*((n-1)!)=N! ∧ n-1≥0 } n:=n-1 { k*(n!)=N! ∧ n≥0 } (Assign)

(SeqComp)

k*(n!)=N! ∧ n≥0 ∧ n>0 ⇒ k*n*((n-1)!)=N! ∧ n-1≥0

{ k*n*((n-1)!)=N! ∧ n-1≥0 } k:=k*n; n:=n-1 { k*(n!)=N! ∧ n≥0 }

(Conseq)

{ k*(n!)=N! ∧ n≥0 ∧ n>0 } k:=k*n; n:=n-1 { k*(n!)=N! ∧ n≥0 }

ループ不変量！

(While)

k=1 ∧ n=N ⇒ k*(n!) = N! ∧ n≥0

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

k*(n!)=N! ∧ n≥0 ∧ ¬(n>0) ⇒ k=N!

(Conseq)

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

大問題
(気づいた？)

* ループ不変量の見つけ方 (invariant discovery)
  * Hoare 論理のルールを見ても，書いてない！
* 理論的には
  * 述語論理で書き下せる.（While ループの意味論をなぞる．cf. 相対完全性）
  * が，とても複雑な式 ➜ その後の検証がうまく回らない．（A'⇒A の真偽判定などで検証器落ちる）
* プログラム検証の研究の大部分が，invariant discovery の研究
  * さまざまな heuristics: (non-)linear arithmetic, abstract interpretation, CEGAR, ...

参考文献

* G. Winskel, The Formal Semantics of Programming Languages: An Introduction. The MIT Press, ISBN 0-262-23169-7
* K. Suenaga and I. Hasuo. Programming with Infinitesimals: A While-Language for Hybrid System Modeling. Proc. ICALP 2011, Track B. LNCS 6756, p. 392-403. Springer-Verlag
* I. Hasuo and K. Suenaga. Exercises in Nonstandard Static Analysis of Hybrid Systems. To appear in Proc. CAV 2012.

プログラム自動検証器のデモ

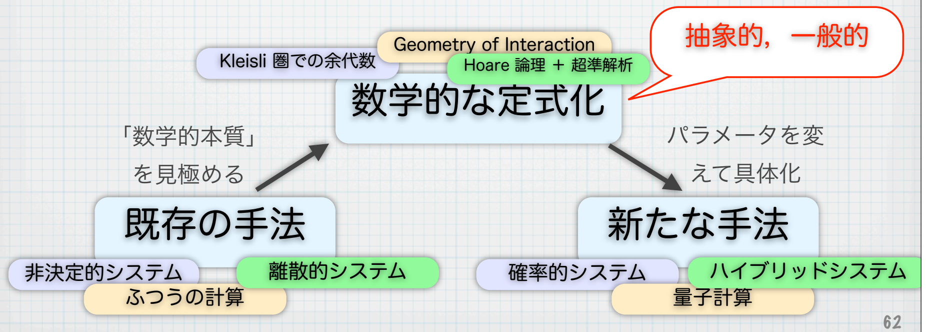
* Hoare 論理 ＋ 超準解析
  * 離散的なプログラムだけでなく，
  * 連続的な物理量も含むハイブリッドシステムの検証
  * 応用： 物理情報システム（車，飛行機，家電，...）
* 末永幸平先生（京大）との共同研究

自己紹介

* 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

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



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

お待ちかね：レポート課題

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` $\{x=3\}$

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

$\{X\geq 0\}$

```
y:=0; x:=X;
while (x>100) {
  y:=y+1;
  x:=x-100
}
```

$\{0\leq x<100 \wedge 100*y + x = X\}$

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

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．

$\{x\geq 0\}$ if x>0 then x:=x-1 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．

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

「n は負でない偶数」

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，

$$GCD(m, n) = GCD(m - n, n)$$

であることを示せ．ただし，GCD(m, n) は m と n の最大公約数を表す．

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，
  $x>0 \wedge y>0 \wedge GCD(x,y)=GCD(X,Y)$

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 $\{x=y=GCD(X,Y)\}$

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