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

（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

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;
}
```

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;
}
```

k	n

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;
}
```

k	n
1	N

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;
}
```

k	n
1	N
N	N-1

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;
}
```

k	n
1	N
N	N-1
N*(N-1)	N-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;
}
```

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

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;
}
```

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

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;
}
```

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

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;
}
```

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

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;
}
```

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)

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？

本当に？

* 証明してみよう！

本当に？

* 証明してみよう！
  * 数学的に厳密に

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

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

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

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，...

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「今までなかったので」：社会 : YOMIURI ONLINE（読売新聞） - お粗末！新幹線トラブル「今までなか
- 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: 新幹線トラブルに見る、開発部門と運用部
- JR東の新幹線運行トラブル、管理システムの仕様を超えるダイヤ変更が原因 - スラッシュドット・ジャパン - JR東の新幹線運行ト
- 新幹線運行システム、15年増強せず能力超過　ＪＲ東　：日本経済新聞 - 新幹線運行システム、15年増強せず能力超過　ＪＲ東
- 【PDF】2011年1月15日及び1月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日発表） - 【PDF】2011年1月
- JR東の新幹線システム障害、原因は処理限度値のオーバー - ITmedia News - JR東の新幹線システム障害、原因は処理限度値のオー
- asahi.com（朝日新聞社）：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
- JR東日本の新幹線トラブル、原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 - asahi.com（朝日新聞社）：新幹線
- 時事ドットコム：「原因、復旧理由も分からず」＝メーカー担当者当面常駐－ＪＲ東・新幹線トラブル - 時事ドットコム：「原因、
- ［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：ITpro - ［続報］三菱UFJ信託銀行の
- 三菱UFJ信託銀行のオンラインシステムでトラブル、ATMやインターネット取引が停止 - ニュース：ITpro - 三菱UFJ信託銀行のオン
- Skypeの大規模ダウンはWindowsクライアントのバグが原因だった - Skypeの大規模ダウンはWindowsクライアントのバグが原因だ
- 問題の本質伝えない岡崎図書館の被害届｜ブログ新聞批評 - 問題の本質伝えない岡崎図書館の被害届｜ブログ新聞批評ブログ新聞
- 岡崎図書館事件のパネル討論会をみてきた - 都市修行僧 - 岡崎図書館事件のパネル討論会をみてきた - 都市修行僧note よく晴れた
- 「動かないコンピュータ」は怖くない - 記者の眼：ITpro - 「動かないコンピュータ」は怖くない - 記者の眼：ITpro「動かないコン
- 大手ITからベンチャー「CROOZ」への転身で分かったこと － @IT - 大手ITからベンチャー「CROOZ」への転身で分かったこと －
- 岡崎図書館未来企画フォーラム「図書館戦争」最前線!? - 気持ちよい生活を送ろう - 岡崎図書館未来企画フォーラム「図書館戦争」
- asahi.com : 図書館アクセスで起訴猶予　男性、被害届取り下げ求める - マイタウン愛知 - asahi.com : 図書館アクセスで起訴猶予

Ariane 5 打ち上げ失敗 (1996)

Ariane 5 打ち上げ失敗 (1996)

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「今までなかったので」：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
- 管理システムの仕様を超えるダイヤ変更が原因 - スラッシュドット・ジ
- せず能力超過　ＪＲ東　：日本経済新聞 - 新幹線運行システム、15
- 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
- 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 - asahi.com（朝日新聞社）：新幹線
- 時事ドットコム：「原因、復旧理由も分からず」＝メーカー担当者当面常駐－ＪＲ東・新幹線トラブル - 時事ドットコム：「原因、
- ［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：ITpro - ［続報］三菱UFJ信託銀行の
- 三菱UFJ信託銀行のオンラインシステムでトラブル、ATMやインターネット取引が停止 - ニュース：ITpro - 三菱UFJ信託銀行のオン
- Skypeの大規模ダウンはWindowsクライアントのバグが原因だった - Skypeの大規模ダウンはWindowsクライアントのバグが原因だ
- 問題の本質伝えない岡崎図書館の被害届｜ブログ新聞批評 - 問題の本質伝えない岡崎図書館の被害届｜ブログ新聞批評ブログ新聞
- 岡崎図書館事件のパネル討論会をみてきた - 都市修行僧 - 岡崎図書館事件のパネル討論会をみてきた - 都市修行僧note よく晴れた
- 「動かないコンピュータ」は怖くない - 記者の眼：ITpro - 「動かないコンピュータ」は怖くない - 記者の眼：ITpro「動かないコン
- 大手ITからベンチャー「CROOZ」への転身で分かったこと – @IT - 大手ITからベンチャー「CROOZ」への転身で分かったこと –
- 岡崎図書館未来企画フォーラム「図書館戦争」最前線!? - 気持ちよい生活を送ろう - 岡崎図書館未来企画フォーラム「図書館戦争」
- asahi.com：図書館アクセスで起訴猶予　男性、被害届取り下げ求める - マイタウン愛知 - asahi.com：図書館アクセスで起訴猶予

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「今までなかったので」：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
- 管理システムの仕様を超えるダイヤ変更が原因 - スラッシュドット・ジ
- せず能力超過　ＪＲ東　：日本経済新聞 - 新幹線運行システム、15
- 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
- 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 - asahi.com（朝日新聞社）：新幹線
- 時事ドットコム：「原因、復旧理由も分からず」＝メーカー担当者当面常駐－ＪＲ東・新幹線トラブル - 時事ドットコム：「原因、
- ［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：ITpro - ［続報］三菱UFJ信託銀行の
- でトラブル、ATMやインターネット取引が停止 - ニュース：ITpro - 三菱UFJ信託銀行のオン
- イアントのバグが原因だった - Skypeの大規模ダウンはWindowsクライアントのバグが原因
- 届｜ブログ新聞批評 - 問題の本質伝えない岡崎図書館の被害届｜ブログ新聞批評ブログ新聞
- きた - 都市修行僧 - 岡崎図書館事件のパネル討論会をみてきた - 都市修行僧note よく晴れた
- 記者の眼：ITpro - 「動かないコンピュータ」は怖くない - 記者の眼：ITpro「動かないコン
- 転身で分かったこと－@IT - 大手ITからベンチャー「CROOZ」への転身で分かったこと－
- 館戦争」最前線!? - 気持ちよい生活を送ろう - 岡崎図書館未来企画フォーラム「図書館戦争
- 男性、被害届取り下げ求める - マイタウン愛知 - asahi.com：図書館アクセスで起訴猶予

Ariane 5 打ち上げ失敗 (1996)

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「今までなかったので」：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
- 管理システムの仕様を超えるダイヤ変更が原因 - スラッシュドット・ジ
- せず能力超過　ＪＲ東　：日本経済新聞 - 新幹線運行システム、15
- 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
- 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
- 時事ドットコム：「原因、復旧理由も分からず」＝メーカー担当者当面常駐－ＪＲ東・新幹
- ［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
- でトラブル、ATMやインターネット取引が停止 - ニュ
- イアントのバグが原因だった - Skypeの大規模ダウンは
- 届｜ブログ新聞批評 - 問題の本質伝えない岡崎図書館
- きた - 都市修行僧 - 岡崎図書館事件のパネル討論会をみ
- 記者の眼：ITpro - 「動かないコンピュータ」は怖く
- 転身で分かったこと － @IT - 大手ITからベンチャー
- 館戦争」最前線!? - 気持ちよい生活を送ろう - 岡崎図書
- 男性、被害届取り下げ求める - マイタウン愛知 - asahi.com：図書館アクセスで起訴猶予

Ariane　5 打ち上げ失敗 (1996)

■民営化後のゆうちょ銀行の主なシステム障害

平成19年10月１日	顧客情報管理システムの不具合で口座開設などの処理が一部遅延
20年７月14日	機器の故障で企業顧客の送金処理約2500件、４億5000万円が遅延
21年５月30日	４店舗でＡＴＭとゆうちょダイレクトに不具合。数十件規模の取引不能に
21年８月24日	ＡＴＭ約3000台でゆうちょ口座あての送金と他行あて送金が不能に
22年１月２日	ＡＴＭ提携金融機関９社で最大約240件の取り扱いが不能に

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

正しさの証明

```
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

正しさの証明

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

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

N の階乗 N! を

計算するプログラム

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

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

$k * (n!) = N!$

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

$k * (n!) = N!$

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

$k * (n!) = N!$

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!

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!

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!

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

$k * (n!) = N!$

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;
}
```

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

$k * (n!) = N!$

正しさの証明

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

ループ不変量：

$k * (n!) = N!$

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 ☃

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.

ループ不変量：

k * (n!) = N!

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 ☃

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.

ループ不変量：

k * (n!) = N!

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.

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 ☃

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.

ループ不変量：

k * (n!) = N!

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」

証明の本質をとらえて，「機械化」

* 本質的な議論は単純？

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

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

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

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 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

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

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

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}

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

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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\perp\}$

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

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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\bot\}$

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

プログラム x:=a の「意味」

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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\bot\}$

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

プログラム x:=a の「意味」

メモリ状態，
たとえば

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

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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\bot\}$

$\longmapsto$ ：「変形する」「移す」

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

プログラム x:=a の「意味」

メモリ状態，
たとえば

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

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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\perp\}$

$\mapsto$ ：「変形する」「移す」

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

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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\perp\}$

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

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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\perp\}$

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

まず P_1 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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\perp\}$

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

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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\bot\}$

$$[\![\texttt{if}\ b\ \texttt{then}\ P_1\ \texttt{else}\ P_2]\!] \ :$$

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

P1，P2 はプログラム
b は「真偽表現」
（x > 0 とか）

プログラムの意味

$$\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\bot\}$$

$$[\![\texttt{while}\, \boldsymbol{b}\, \boldsymbol{P}]\!] \;:\; \boldsymbol{\sigma} \longmapsto ??$$

プログラムの意味

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

$$[\![\mathtt{while}\, b\, P]\!] \;:\; \sigma \longmapsto ??$$

プログラムの意味

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

$$[\![\texttt{while}\, b\, P]\!] \;:\; \boldsymbol{\sigma} \longmapsto ??$$

* 状態変換の繰り返し $[\![P]\!]^n\boldsymbol{\sigma}$を考える：

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

  * ある時点で b が偽になれば，つまり $[\![b]\!]([\![P]\!]^n\boldsymbol{\sigma}) = \mathrm{false}$ なら，そうなるような最初の n に対する $[\![P]\!]^n\boldsymbol{\sigma}$ 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

「停止しない」

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

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

* 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 $[\![b]\!]([\![P]\!]^n\sigma) = \mathrm{false}$ 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

「停止しない」

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

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

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

$$[\![\texttt{if}\; b \;\texttt{then}\; P_1 \;\texttt{else}\; P_2]\!] \;:$$

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

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

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 $\mathbf{MSt} \longrightarrow \mathbf{MSt} \cup \{\perp\}$

「停止しない」

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

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

$$[\![\texttt{if}\ b\ \texttt{then}\ P_1\ \texttt{else}\ P_2]\!] :$$

$$\boldsymbol{\sigma} \longmapsto \begin{cases} [\![P_1]\!]\boldsymbol{\sigma} & \text{if } [\![b]\!]\boldsymbol{\sigma} \text{ is true} \\ [\![P_2]\!]\boldsymbol{\sigma} & \text{if } [\![b]\!]\boldsymbol{\sigma} \text{ is false} \end{cases}$$

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

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：
  * $x = 5 \land y \leq 3$
  * $\exists z.\ (x = 2 * z \land y = 3 * z)$

Assertion Language

* Assertion: メモリ状態の性質を記述する「論理式」．例：
  * $x = 5 \wedge y \leq 3$
  * $\exists z. (x = 2 * z \wedge y = 3 * z)$

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：

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

  * 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 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

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 }

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}

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

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

Hoare Triple の意味論

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

Hoare Triple の意味論

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

{ A } P { B }

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

プログラム

Hoare Triple の意味論

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

{ A } P { B }

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

プログラム

* 「{A} P {B} が真」であるとは,

Hoare Triple の意味論

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

{ A } P { B }

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

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,

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 $[\![P]\!]\sigma$ が assertion B をみたすことをいう.

Hoare Triple の意味論

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

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

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

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 $[\![P]\!]\sigma$ 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.
* 「⊥ 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）

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 $[\![P]\!]\sigma$ 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）

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 $[\![P]\!]\sigma$ 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 }

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

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

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

* 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

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

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）

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

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」）

* 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：

$$\frac{}{\{y=2\}\ x:=y\ \{x=2\}}$$

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

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

Hoare 論理の導出規則

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

* （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」

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

* （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」

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

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：

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

さっきの例，くわしく！！

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

Step 1:
「実行後に y=2 になってほしんだけど．．．」

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

Step 2:
「y:=y-1 の前には y-1=2 であればいいぞ」

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

Step 3:
「さらに x:=y の前には x-1=2 であればいいぞ」

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

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

さっきの例，くわしく！！

(Assign) (Assign)

{x-1=2} x:=y {y-1=2} {y-1=2} y:=y-1 {y=2}

(SeqComp)

{x-1=2} x:=y; y:=y-1 {y=2}

Step 1:
「実行後に y=2 になってほしんだけど．．．」

(Assign) (Assign)

{x-1=2} x:=y {y-1=2} {y-1=2} y:=y-1 {y=2}

(SeqComp)

{x-1=2} x:=y; y:=y-1 {y=2}

Step 2:
「y:=y-1 の前には y-1=2 であればいいぞ」

(Assign) (Assign)

{x-1=2} x:=y {y-1=2} {y-1=2} y:=y-1 {y=2}

(SeqComp)

{x-1=2} x:=y; y:=y-1 {y=2}

Step 3:
「さらに x:=y の前には x-1=2 であればいいぞ」

(Assign) (Assign)

{x-1=2} x:=y {y-1=2} {y-1=2} y:=y-1 {y=2}

(SeqComp)

{x-1=2} x:=y; y:=y-1 {y=2}

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

さっきの例，くわしく！！

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

Step 1:
「実行後に y=2 になってほしんだけど．．．」

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

Step 2:
「y:=y-1 の前には y-1=2 であればいいぞ」

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

Step 3:
「さらに x:=y の前には x-1=2 であればいいぞ」

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

考えの流れ

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

さっきの例，くわしく！！

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

Step 1:
「実行後に y=2 になってほしんだけど．．．」

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

Step 2:
「y:=y-1 の前には y-1=2 であればいいぞ」

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

Step 3:
「さらに x:=y の前には x-1=2 であればいいぞ」

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

考えの流れ

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

さっきの例，くわしく！！

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

Step 1:
「実行後に y=2 になってほしんだけど．．．」

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

Step 2:
「y:=y-1 の前には y-1=2 であればいいぞ」

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

Step 3:
「さらに x:=y の前には x-1=2 であればいいぞ」

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

考えの流れ

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

さっきの例，くわしく！！

{x-1=2} x:=y {y-1=2} (Assign)

{y-1=2} y:=y-1 {y=2} (Assign)

{x-1=2} x:=y; y:=y-1 {y=2} (SeqComp)

Step 1:
「実行後に y=2 になってほしんだけど．．．」

{x-1=2} x:=y {y-1=2} (Assign)

{y-1=2} y:=y-1 {y=2} (Assign)

{x-1=2} x:=y; y:=y-1 {y=2} (SeqComp)

Step 2:
「y:=y-1 の前には y-1=2 であればいいぞ」

{x-1=2} x:=y {y-1=2} (Assign)

{y-1=2} y:=y-1 {y=2} (Assign)

{x-1=2} x:=y; y:=y-1 {y=2} (SeqComp)

Step 3:
「さらに x:=y の前には x-1=2 であればいいぞ」

{x-1=2} x:=y {y-1=2} (Assign)

{y-1=2} y:=y-1 {y=2} (Assign)

{x-1=2} x:=y; y:=y-1 {y=2} (SeqComp)

考えの流れ

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

さっきの例，くわしく！！

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

Step 1:
「実行後に y=2 になってほしんだけど．．．」

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

Step 2:
「y:=y-1 の前には y-1=2 であればいいぞ」

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

Step 3:
「さらに x:=y の前には x-1=2 であればいいぞ」

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

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：

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

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

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

- 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！

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

* 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：

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

Hoare 論理の導出規則

A はループ不変量！

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

ループを脱出した ➜
b は成立しないはず

* 例：

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

Hoare 論理の導出規則

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

* 例：

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

Hoare 論理の導出規則

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

規則の名前
（consequence =
「（論理的）帰結」）

* 例：

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

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」）

* 例：

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

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」）

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

Hoare 論理の導出規則 (まとめ)

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

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

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

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

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．

$\{ k=1 \wedge n=N \}$ 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！

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．

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

* 「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⟧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

はじめの例

```
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!

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

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

Hoare 論理による証明

(Assign)

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

(Assign)

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

(SeqComp)

$k*(n!)=N! \land n \geq 0 \land n>0 \Rightarrow k*n*((n-1)!)=N! \land n-1 \geq 0$

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

(Conseq)

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

(While)

$k=1 \land n=N \Rightarrow k*(n!) = N! \land n \geq 0$

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

$k*(n!)=N! \land n \geq 0 \land \neg(n>0) \Rightarrow k=N!$

(Conseq)

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

Hoare 論理による証明

(Assign)

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

(Assign)

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

(SeqComp)

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

$k*(n!)=N! \land n\geq 0 \land n>0 \Rightarrow k*n*((n-1)!)=N! \land n-1\geq 0$

(Conseq)

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

(While)

$k=1 \land n=N \Rightarrow k*(n!) = N! \land n\geq 0$

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

$k*(n!)=N! \land n\geq 0 \land \neg(n>0) \Rightarrow k=N!$

(Conseq)

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

Hoare 論理(

最初にやった数学的証明の
「形式化」
（エッセンスは同じ）

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

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

$$\dfrac{\cdots}{\{k*n*((n-1)!)=N! \land n-1\geq 0\}\ k:=k*n;\ n:=n-1\ \{k*(n!)=N! \land n\geq 0\}}\ \text{(SeqComp)}$$

$k*(n!)=N!$
$\land\ n\geq 0 \land n>0$
$\Rightarrow k*n*((n-1)!)=N!$
$\quad \land\ n-1\geq 0$

$$\dfrac{\cdots}{\{k*(n!)=N! \land n\geq 0 \land n>0\}\ k:=k*n;\ n:=n-1\ \{k*(n!)=N! \land n\geq 0\}}\ \text{(Conseq)}$$

$$\dfrac{\cdots}{\{k*(n!) = N! \land n\geq 0\}\ \text{while } (n>0)\ k:=k*n;\ n:=n-1\ \{k*(n!)=N! \land n\geq 0 \land \neg(n>0)\}}\ \text{(While)}$$

$k=1 \land n=N$
$\Rightarrow$
$k*(n!) = N!$
$\land\ n\geq 0$

$k*(n!)=N!$
$\land\ n\geq 0 \land \neg(n>0)$
$\Rightarrow k=N!$

$$\dfrac{\cdots}{\{k=1 \land n=N\}\ \text{while } (n>0)\ k:=k*n;\ n:=n-1\ \{k = N!\}}\ \text{(Conseq)}$$

Hoare 論理(

最初にやった数学的証明の
「形式化」
（エッセンスは同じ）

(Assign)

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

(Assign)

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

(SeqComp)

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

$k*(n!)=N! \wedge n \geq 0 \wedge n>0 \Rightarrow k*n*((n-1)!)=N! \wedge n-1 \geq 0$

(Conseq)

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

ループ不変量！

(While)

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

$k=1 \wedge n=N \Rightarrow k*(n!) = N! \wedge n \geq 0$

$k*(n!)=N! \wedge n \geq 0 \wedge \neg(n>0) \Rightarrow k=N!$

(Conseq)

$\{ k=1 \wedge n=N \}$ 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）

(Assign) (Assign)

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

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

(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！

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）

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）

* 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-）

* 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：

* 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

* 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

* 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

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

既存の手法

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

数学的な定式化

「数学的本質」
を見極める

既存の手法

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

抽象的，一般的

数学的な定式化

「数学的本質」

を見極める

既存の手法

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

抽象的，一般的

数学的な定式化

「数学的本質」
を見極める

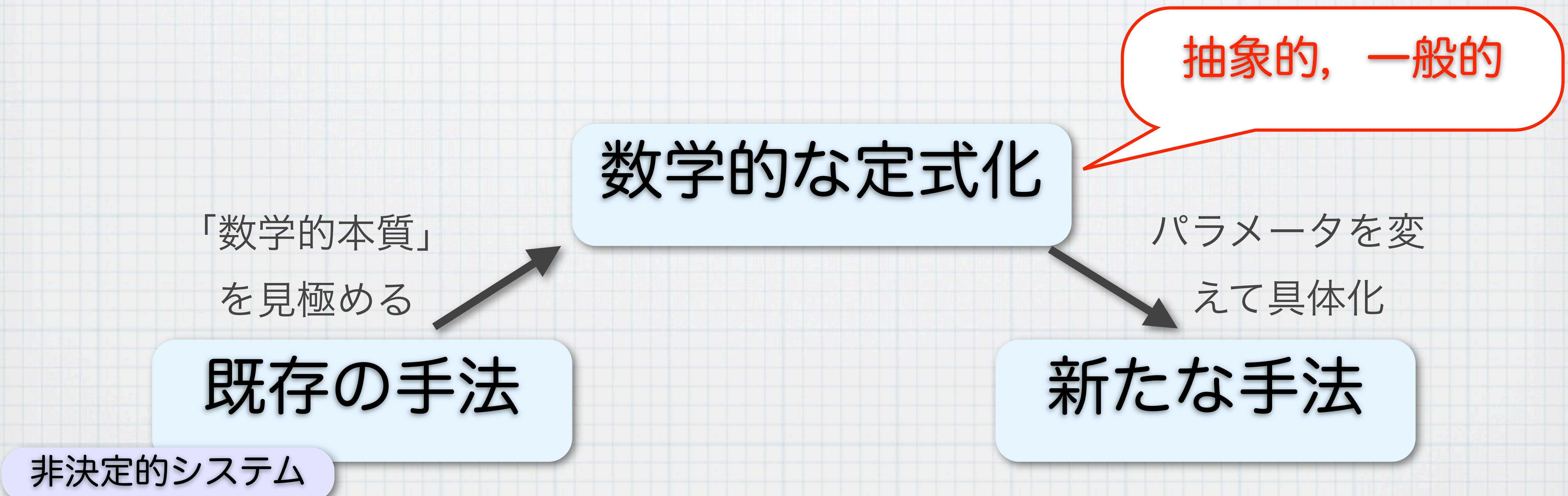
パラメータを変
えて具体化

既存の手法

新たな手法

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



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

抽象的，一般的

Kleisli 圏での余代数

数学的な定式化

「数学的本質」を見極める

パラメータを変えて具体化

既存の手法

非決定的システム

新たな手法

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

抽象的，一般的

Kleisli 圏での余代数

数学的な定式化

「数学的本質」を見極める

パラメータを変えて具体化

既存の手法

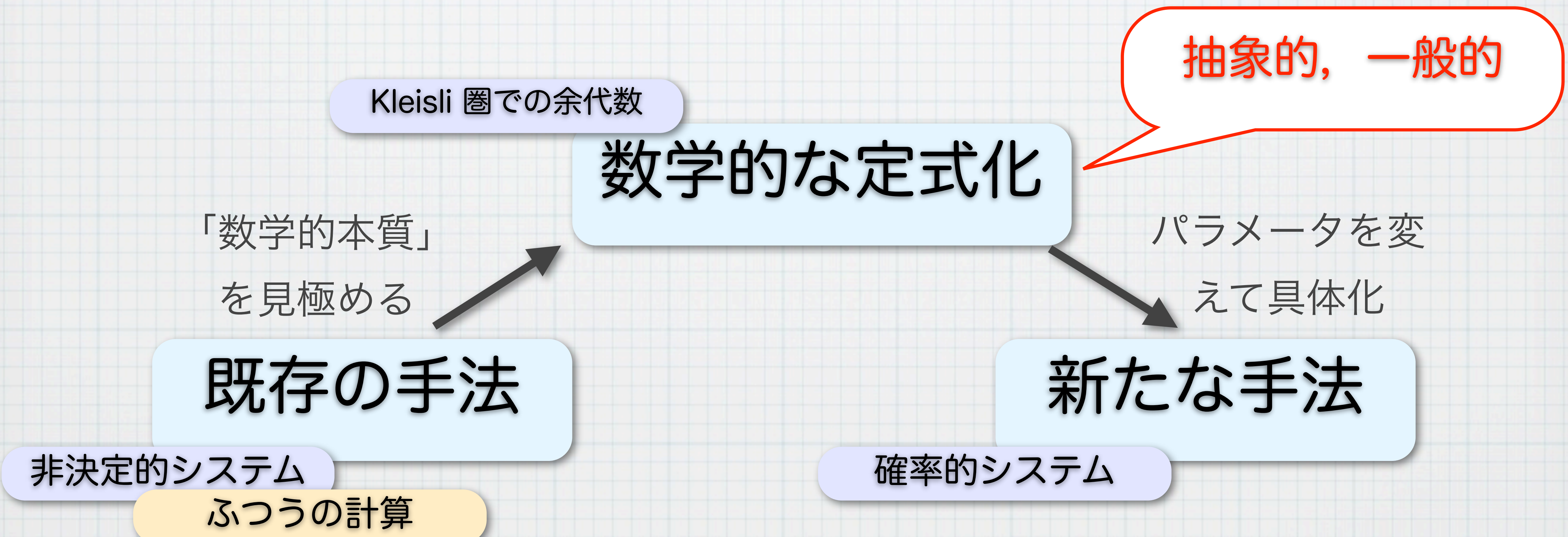
非決定的システム

新たな手法

確率的システム

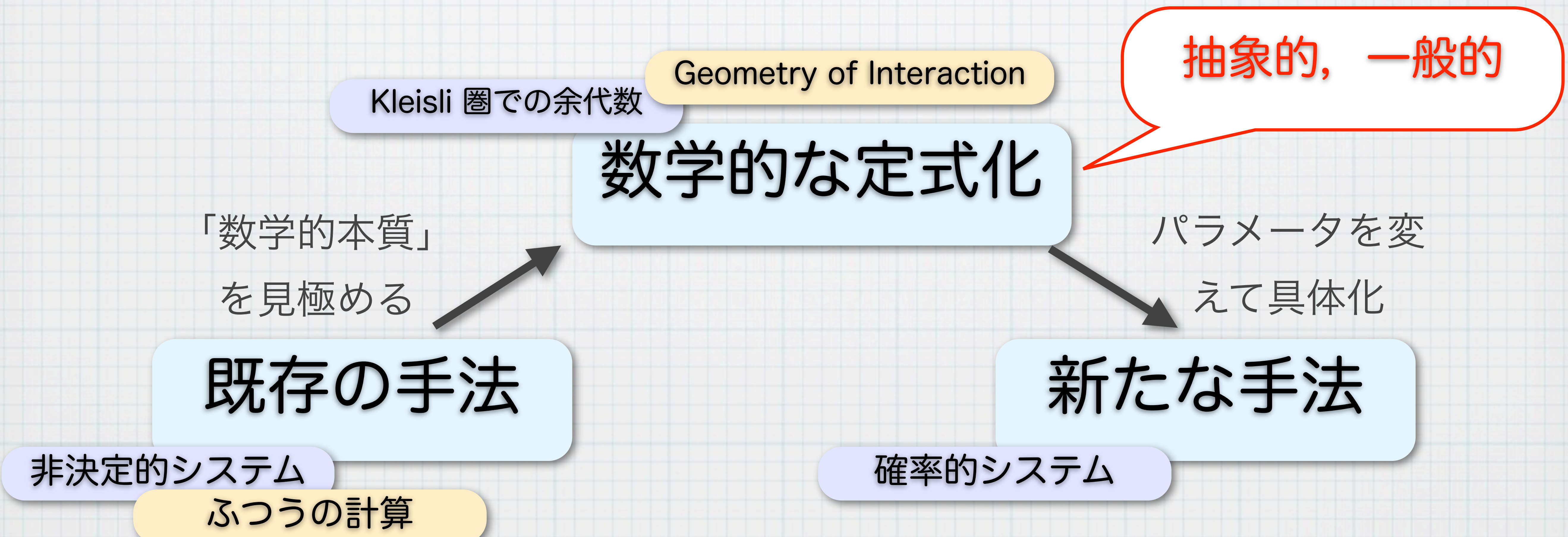
研究テーマ

* 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



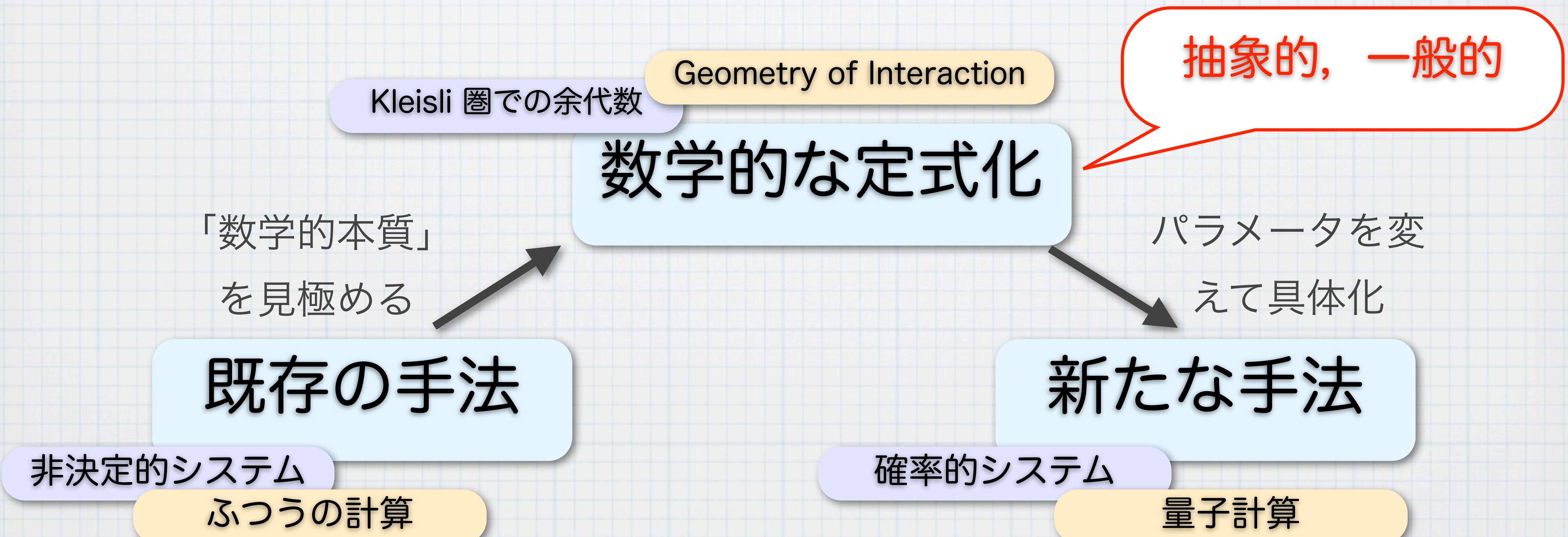
研究テーマ

* 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



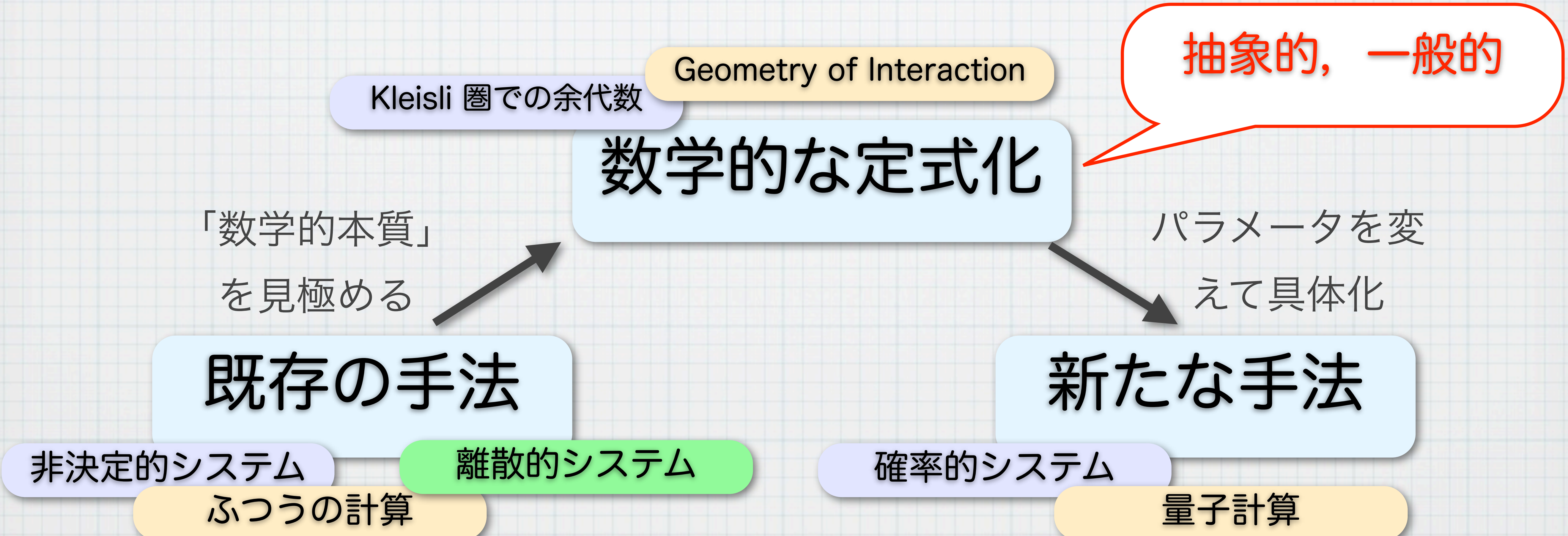
研究テーマ

* 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



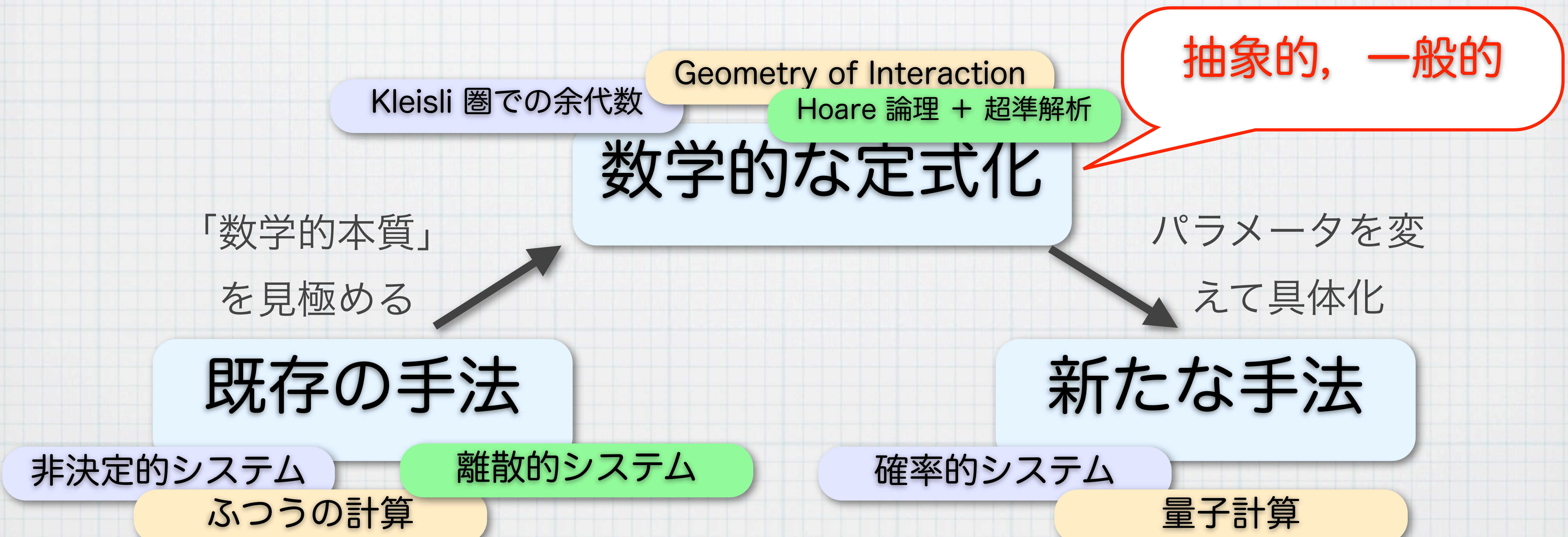
研究テーマ

* 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



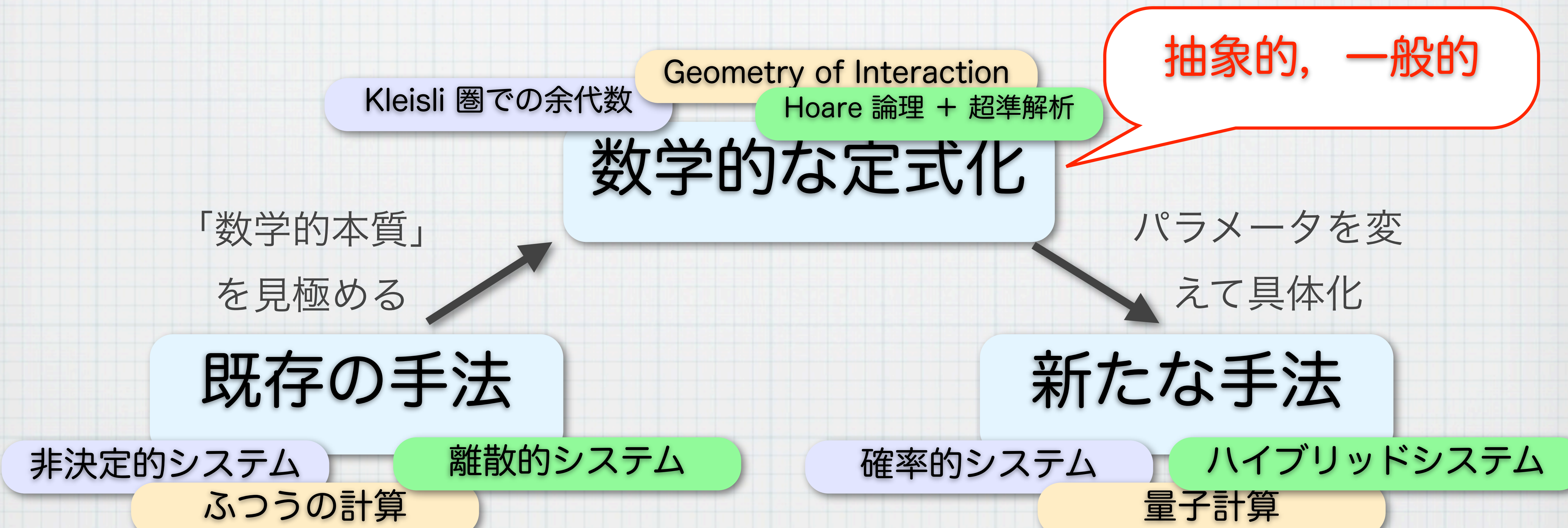
研究テーマ

* 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



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！

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）

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

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

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

$\{\, X\geq 0 \,\}$

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

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

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≥0 }

{ X≥0 }

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

{ 0≤x<100 ∧ 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 $\{x \geq 0\}$

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

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} $\{ n = 0 \}$

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
}
```

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

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

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