形式的技法によるプログラム検証

JavaCard における適用例

永宮 悠大

群馬大学大学院工学研究科情報工学専攻 藤田研究室

Email : nagamiya@comp.cs.gunma-u.ac.jp

概要

- **形式的技法**
- Hoare **論理**
- Java **プログラムの検証**
- **まとめと今後の課題**

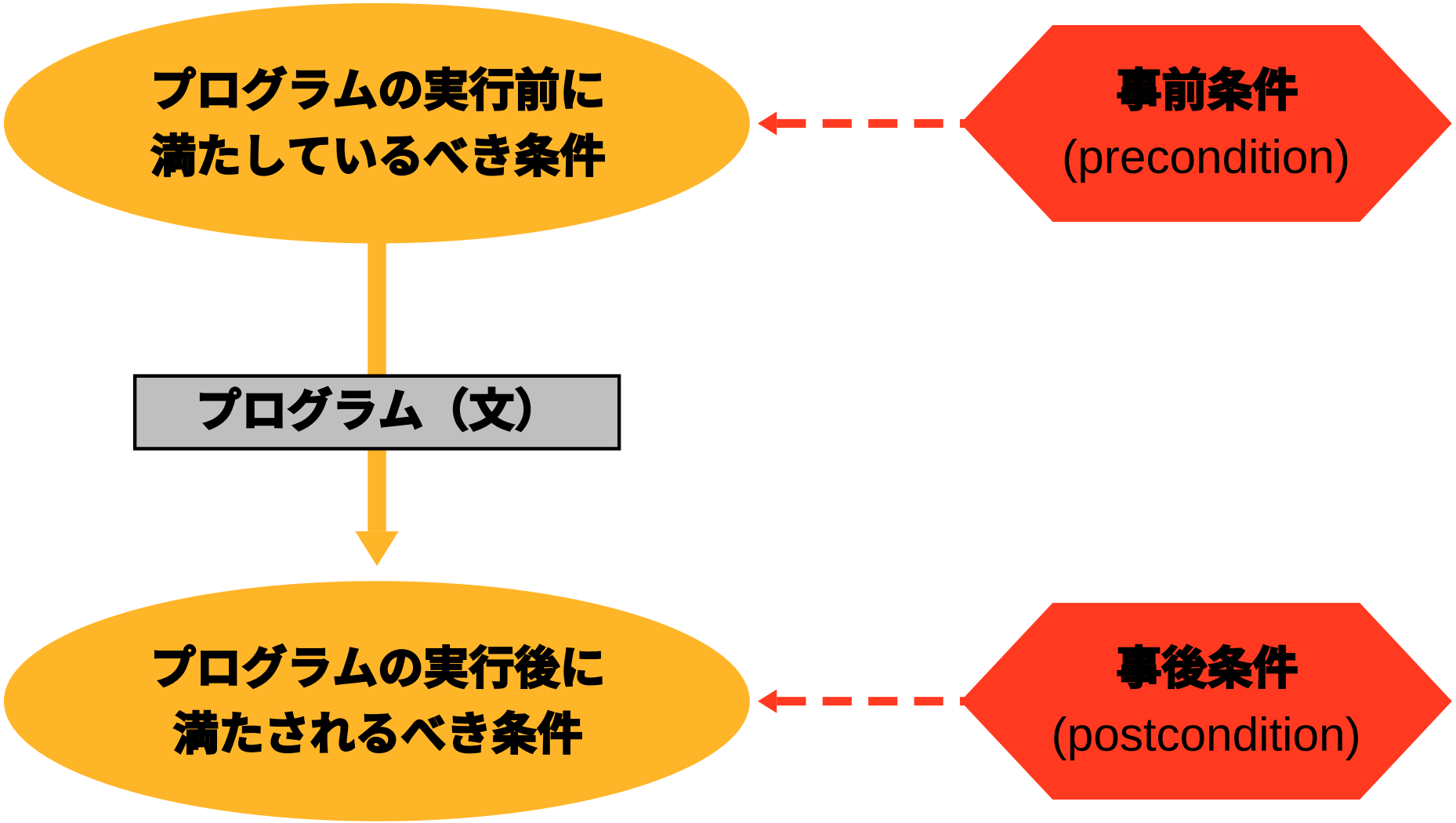
形式的技法(formal method)

- **形式的仕様記述** (formal specification)
  - **プログラムの仕様を形式的** (formal) **に記述すること**
    - **論理や代数を用いる**
    - JML, D, VDM, ・・・
- **形式的検証** (formal method)
  - **プログラムが仕様を満たすことを形式的に証明すること**
    - **論理的な推論（定理証明技法）**
    - **オートマトンによる全数検査（モデル検査技法）**

概要

- 形式的技法
- Hoare 論理
- Java プログラムの検証
- まとめと今後の課題

事前条件と事後条件


プログラムの実行前に
満たしているべき条件
事前条件
(precondition)
プログラム（文）
プログラムの実行後に
満たされるべき条件
事後条件
(postcondition)

Hoare 論理

- $[\![\phi]\!]\ P\ [\![\psi]\!]$
  - ϕ, ψ は論理式で，P はプログラム（文）
  - 事前条件 ϕ を満たした状態でプログラム P を実行し，P の実行が停止するならば，実行後の状態は事後条件 ψ を満たす（部分正当性）
  - $[\![\phi]\!]\ P\ [\![\psi]\!]$ はプログラム P の部分正当性に関する検証文と呼ばれる

代入文の公理

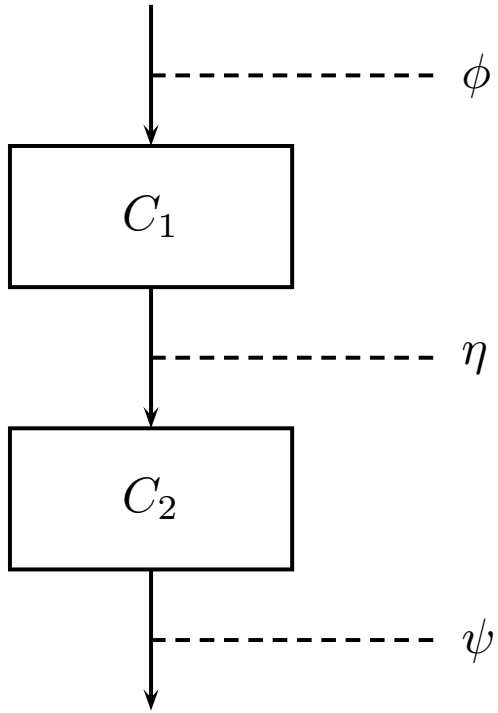
$$\overline{[\![\psi[E/x]]\!]\ x := E\ [\![\psi]\!]}$$

- $\psi[E/x]$ は ψ 中の変数 x を式 E で置き換えたもの
- 例：

  $[\![x + 1 > 1]\!]$ `x := x + 1` $[\![x > 1]\!]$

複合文の規則

$$\frac{[\![\phi]\!]\ C_1\ [\![\eta]\!] \qquad [\![\eta]\!]\ C_2\ [\![\psi]\!]}{[\![\phi]\!]\ C_1; C_2\ [\![\psi]\!]}$$



帰結の規則

$$\frac{\vdash_{AR} \phi' \rightarrow \phi \qquad [\![\phi]\!]\ C\ [\![\psi]\!] \qquad \vdash_{AR} \psi \rightarrow \psi'}{[\![\phi']\!]\ C\ [\![\psi']\!]}$$

証明の例

$$\dfrac{\dfrac{\vdash_{AR} \top \rightarrow 0 = 0 \qquad [\![0 = 0]\!]\ \mathtt{x\ =\ 0}\ [\![x = 0]\!]}{[\![\top]\!]\ \mathtt{x\ =\ 0}\ [\![x = 0]\!]} \qquad [\![x = 0]\!]\ \mathtt{y\ =\ x}\ [\![y = 0]\!]}{[\![\top]\!]\ \mathtt{x\ =\ 0;\ y\ =\ x}\ [\![y = 0]\!]}$$

証明の例

$$\dfrac{\dfrac{\vdash_{AR} \top \rightarrow 0 = 0 \qquad [\![0 = 0]\!]\ \mathtt{x\ =\ 0}\ [\![x = 0]\!]}{[\![\top]\!]\ \mathtt{x\ =\ 0}\ [\![x = 0]\!]} \qquad [\![x = 0]\!]\ \mathtt{y\ =\ x}\ [\![y = 0]\!]}{[\![\top]\!]\ \mathtt{x\ =\ 0;\ y\ =\ x}\ [\![y = 0]\!]}$$

$[\![\top]\!]$

$[\![0 = 0]\!]$

`x = 0;`

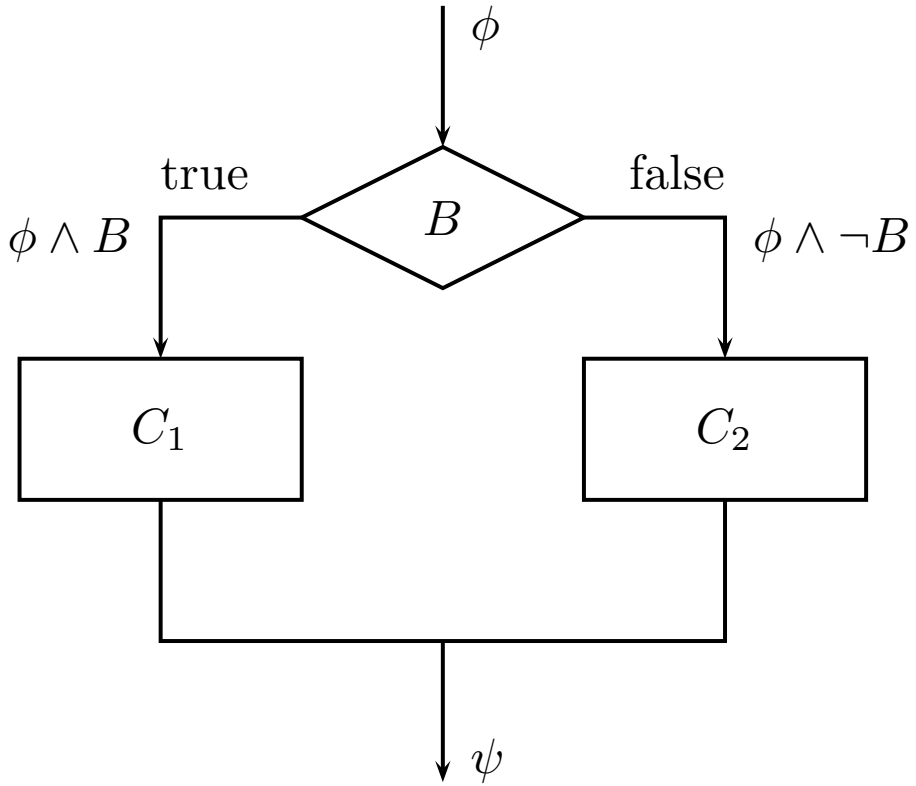
$[\![x = 0]\!]$

`y = x;`

$[\![y = 0]\!]$

if文の規則

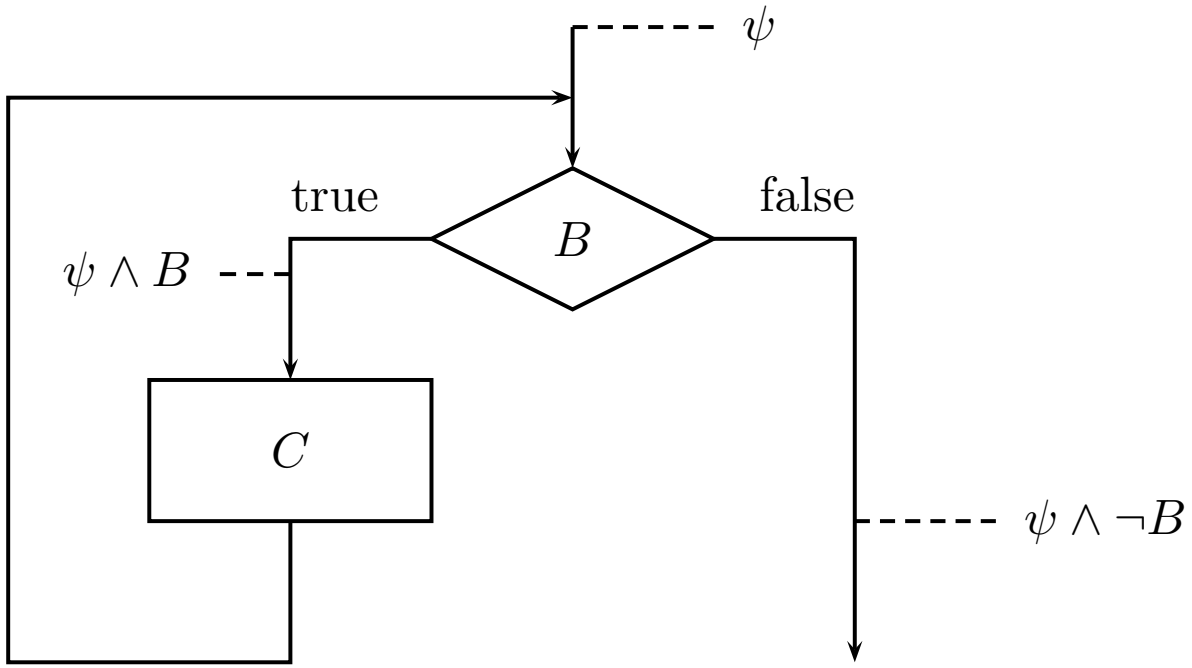
$$\frac{[\![\phi \wedge B]\!]\ C_1\ [\![\psi]\!] \qquad [\![\phi \wedge \neg B]\!]\ C_2\ [\![\psi]\!]}{[\![\phi]\!]\ \texttt{if}\ B\ \{C_1\}\ \texttt{else}\ \{C_2\}\ [\![\psi]\!]}$$



while文の規則

$$\frac{[\![\psi \wedge B]\!]\ C\ [\![\psi]\!]}{[\![\psi]\!]\ \texttt{while}\ B\ \{C\}\ [\![\psi \wedge \neg B]\!]}$$

- ψ **はループ不変条件** (loop invariant) **と呼ばれる**



例：階乗

```
      y = 1;
      z = 0;
l1:   while (z != x) {
          z = z + 1;
          y = y * z;
l2:   }
```

事前・事後条件の設定

$[\![\top]\!]$

```
      y = 1;
      z = 0;
l1:   while (z != x) {
          z = z + 1;
          y = y * z;
l2:   }
```

$[\![y = x!]\!]$

ループ不変条件の設定

- $x = 6$ の場合：

ループ回数	z at l1	y at l1	$z \neq x!$ at l1
0	0	1	true
1	1	1	true
2	2	2	true
3	3	6	true
4	4	24	true
5	5	120	true
6	6	720	false

ループ不変条件の設定

- $x = 6$ の場合：

ループ回数	z at l1	y at l1	$z \neq x!$ at l1
0	0	1	true
1	1	1	true
2	2	2	true
3	3	6	true
4	4	24	true
5	5	120	true
6	6	720	false

ループ不変条件は $y = z!$

while 文の規則の適用

$[\![\top]\!]$

```
    y = 1;
    z = 0;
```

$[\![y = z!]\!]$

```
l1: while (z != x) {
```

$[\![y = z! \wedge z \neq x]\!]$

```
        z = z + 1;
        y = y * z;
```

$[\![y = z!]\!]$

```
l2: }
```

$[\![y = z! \wedge z = x]\!]$

$[\![y = x!]\!]$

代入文の公理の適用

$[\![\top]\!]$

```
    y = 1;
    z = 0;
```

$[\![y = z!]\!]$

```
l1: while (z != x) {
```

$[\![y = z! \wedge z \neq x]\!]$

```
        z = z + 1;
```

$[\![y \cdot z = z!]\!]$

```
        y = y * z;
```

$[\![y = z!]\!]$

```
l2: }
```

$[\![y = z! \wedge z = x]\!]$

$[\![y = x!]\!]$

代入文の公理の適用

$[\![\top]\!]$

```
y = 1;
z = 0;
```

$[\![y = z!]\!]$

```
l1:  while (z != x) {
```

$[\![y = z! \wedge z \neq x]\!]$

$[\![y \cdot (z + 1) = (z + 1)!]\!]$

```
z = z + 1;
```

$[\![y \cdot z = z!]\!]$

```
y = y * z;
```

$[\![y = z!]\!]$

```
l2:  }
```

$[\![y = z! \wedge z = x]\!]$

$[\![y = x!]\!]$

代入文の公理の適用

$[\![\top]\!]$

```
    y = 1;
```

$[\![y = 0!]\!]$

```
    z = 0;
```

$[\![y = z!]\!]$

```
l1: while (z != x) {
```

$[\![y = z! \wedge z \neq x]\!]$

$[\![y \cdot (z + 1) = (z + 1)!]\!]$

```
        z = z + 1;
```

$[\![y \cdot z = z!]\!]$

```
        y = y * z;
```

$[\![y = z!]\!]$

```
l2: }
```

$[\![y = z! \wedge z = x]\!]$

$[\![y = x!]\!]$

代入文の公理の適用

$[\![\top]\!]$

$[\![1 = 0!]\!]$

```
    y = 1;
```

$[\![y = 0!]\!]$

```
    z = 0;
```

$[\![y = z!]\!]$

```
l1: while (z != x) {
```

$[\![y = z! \wedge z \neq x]\!]$

$[\![y \cdot (z + 1) = (z + 1)!]\!]$

```
        z = z + 1;
```

$[\![y \cdot z = z!]\!]$

```
        y = y * z;
```

$[\![y = z!]\!]$

```
l2: }
```

$[\![y = z! \wedge z = x]\!]$

$[\![y = x!]\!]$

帰結の規則の適用

$[\![\top]\!]$

$[\![1 = 0!]\!]$ $\quad \vdash_{AR} \top \rightarrow 1 = 0!$

```
    y = 1;
```

$[\![y = 0!]\!]$

```
    z = 0;
```

$[\![y = z!]\!]$

```
l1: while (z != x) {
```

$[\![y = z! \wedge z \neq x]\!]$ $\quad \vdash_{AR} (y = z! \wedge z \neq x) \rightarrow$

$[\![y \cdot (z + 1) = (z + 1)!]\!]$ $\quad (y \cdot (z + 1) = (z + 1)!)$

```
        z = z + 1;
```

$[\![y \cdot z = z!]\!]$

```
        y = y * z;
```

$[\![y = z!]\!]$

```
l2: }
```

$[\![y = z! \wedge z = x]\!]$

$[\![y = x!]\!]$ $\quad \vdash_{AR} (y = z! \wedge z = x) \rightarrow (y = x!)$

帰結の規則の適用

$[\![\top]\!]$

$[\![1 = 0!]\!]$ $\quad \vdash_{AR} \top \rightarrow 1 = 0!$

```
    y = 1;
```

$[\![y = 0!]\!]$

```
    z = 0;
```

$[\![y = z!]\!]$

```
l1: while (z != x) {
```

$[\![y = z! \wedge z \neq x]\!]$ $\quad \vdash_{AR} (y = z! \wedge z \neq x) \rightarrow$

$[\![y \cdot (z+1) = (z+1)!]\!]$ $\quad (y \cdot (z+1) = (z+1)!)$

```
      z = z + 1;
```

$[\![y \cdot z = z!]\!]$

```
      y = y * z;
```

$[\![y = z!]\!]$

```
l2: }
```

$[\![y = z! \wedge z = x]\!]$

$[\![y = x!]\!]$ $\quad \vdash_{AR} (y = z! \wedge z = x) \rightarrow (y = x!)$

Q.E.D

概要

- **形式的技法**
- Hoare **論理**
- Java **プログラムの検証**
- **まとめと今後の課題**

契約による設計(Design by contract)

- $[\![\phi]\!]\ P\ [\![\psi]\!]$ をメソッド呼出し側と実装側の間の契約 (contract) と考える
  - 呼出し側が ϕ を満たした状態でメソッド P を呼出すことを約束するならば，実装側は P を実行した結果が ψ を満たすことを保証する
  - ϕ が満たされなかったときは，メソッド呼出し側にバグがある
  - ψ が満たされなかったときは，メソッド内部にバグがある

契約

- **事前条件** (precondition)
  - **メソッドの入力引数の検査**
- **事後条件** (postcondition)
  - **メソッドの返り値，副作用の検査**
- **クラス不変条件** (invariant)
  - **常に真でなければならないクラスの性質を記述**

Java Modeling Language

- Java **プログラムの形式的な仕様を記述するための言語**
- Design by contract **の考えに基づいている**

JML による仕様の記述

- JML **仕様はソースコードの中に@マーク付きのコメントとして記述**

```
//@ ...

/*@ ...
  *@ ...
  *@/
```

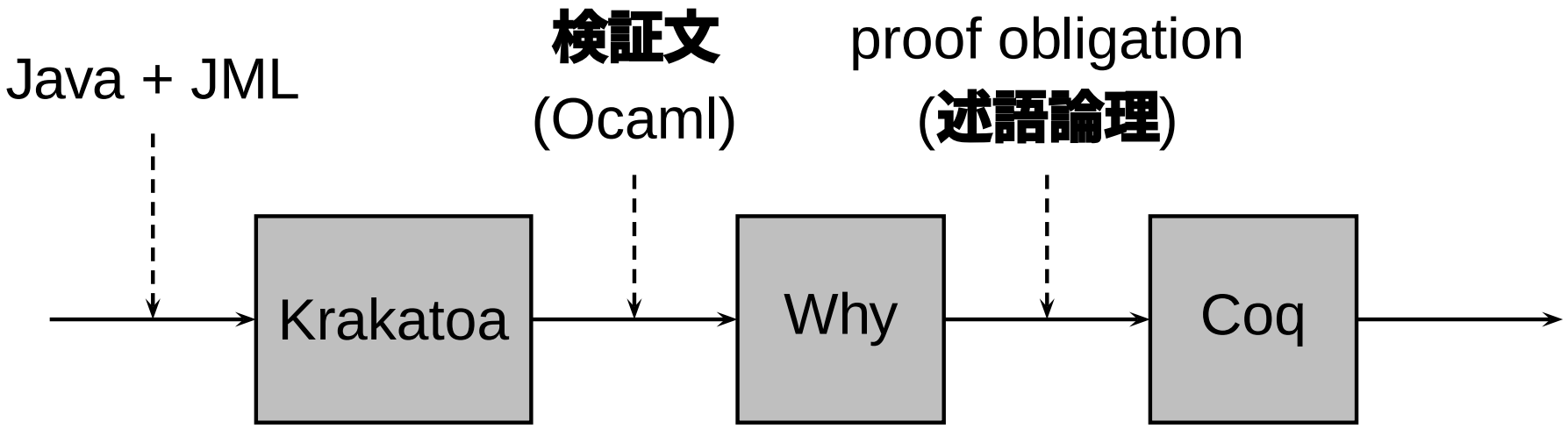
JMLの構文

- **事前・事後条件：**requires **と** ensures
  - `//@ requires array.length >=1;`
  - `//@ ensures \result >= x;`
- **クラス不変条件：**invariant
  - `//@ public invariant balance >= 0;`
- **フレーム条件：**modefiable
  - **メソッドによって変更される変数**
  - `//@ modifiable array[x];`

JML のキーワード

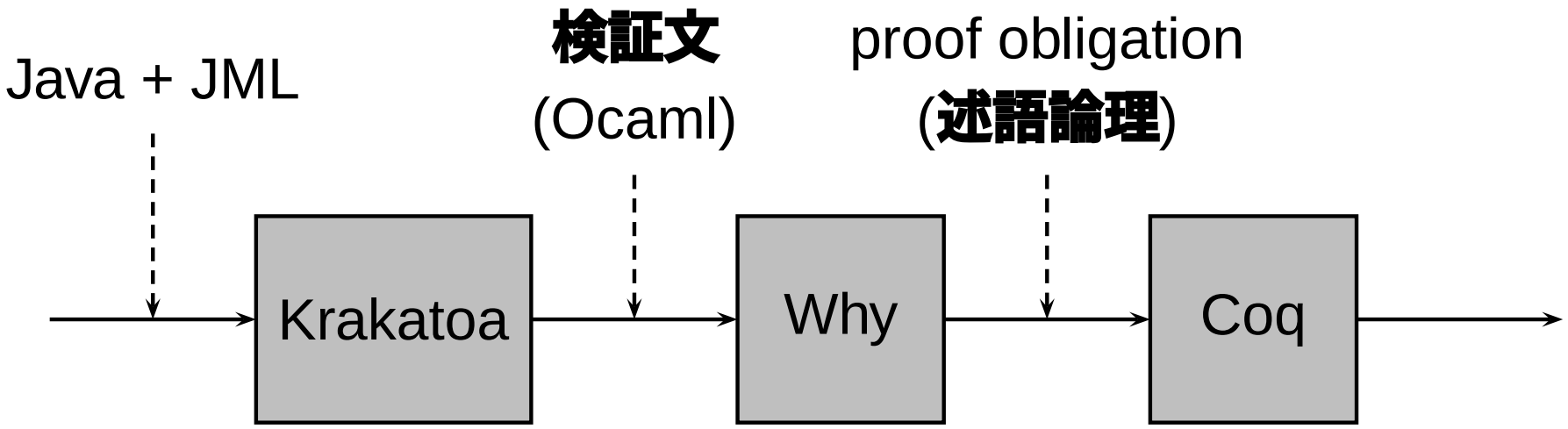
- **メソッドの返り値：** `\result`
- **メソッド実行前の式 E の値：** `\old(E)`
- **量化記号：** `\forall`, `\exists`

Krakatoa+Coq



- Krakatoa: Java/JML **ソース** $\mapsto$ **検証文**
- Why: **検証文** $\mapsto_{\text{Hoare}}$ proof obligation
- Coq: proof obligation **を対話的に証明**

Krakatoa+Coq



- Krakatoa: Java/JML **ソース** $\mapsto$ **検証文**
- Why: **検証文** $\mapsto_{\text{Hoare}}$ proof obligation
- Coq: proof obligation **を対話的に証明**

proof obligation **が証明できる** $\Rightarrow$ **プログラムが仕様を満たす**

Gemplus の電子財布

- JavaCard の検証のための現実的な例として開発されたアプリケーション
- 入金，出金，外貨両替
- アプレット間通信
- 本人認証 (ID とパスワード)
- ロイヤリティ・ポイント

電子財布のクラス図

Utils**パッケージ**

- Jour **日**
- Mois **月**
- Annee **年**
- Decimal **残高に関する演算**

Pcpcapinterfaces**パッケージ**

- PurseLoyaltyInterface Loyalty **アプレットとの通信**
- TransactionInterface Shareable **インタフェース**

電子財布のクラス図

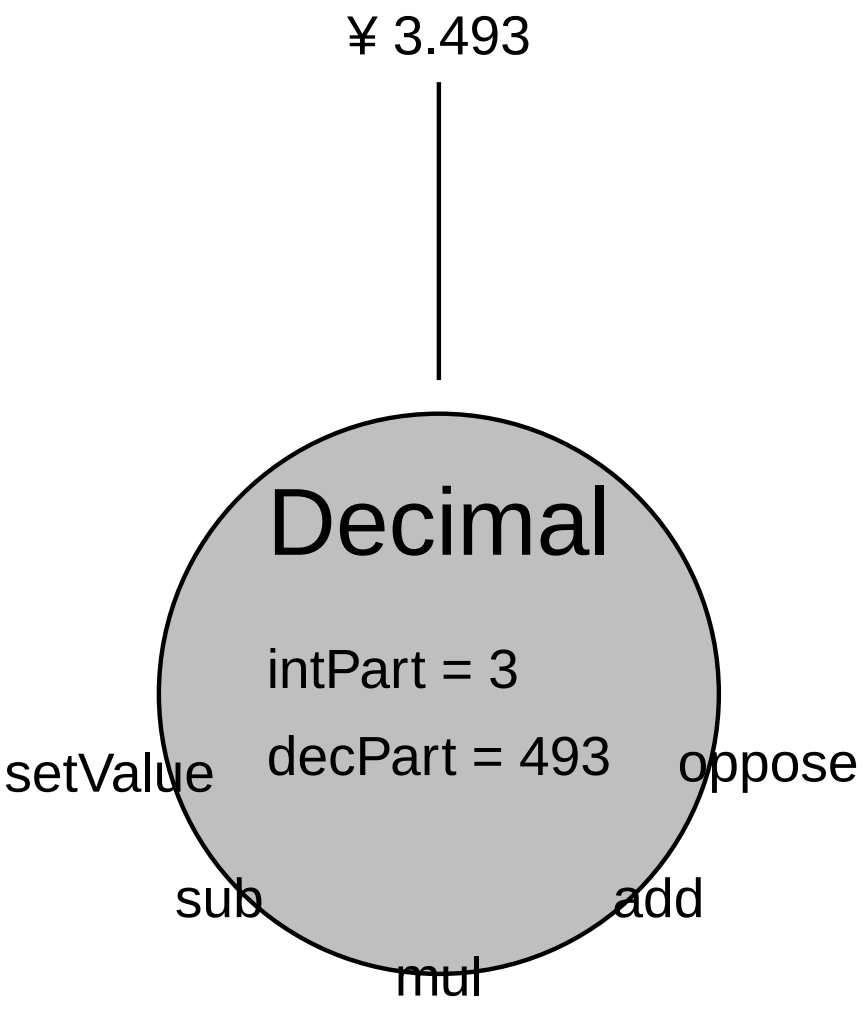
Utils**パッケージ**

- Jour **日**
- Mois **月**
- Annee **年**
- Decimal **残高に関する演算**

Pcpcapinterfaces**パッケージ**

- PurseLoyaltyInterface Loyalty **アプレットとの通信**
- TransactionInterface Shareable **インタフェース**

Decimalクラスによる残高の表現



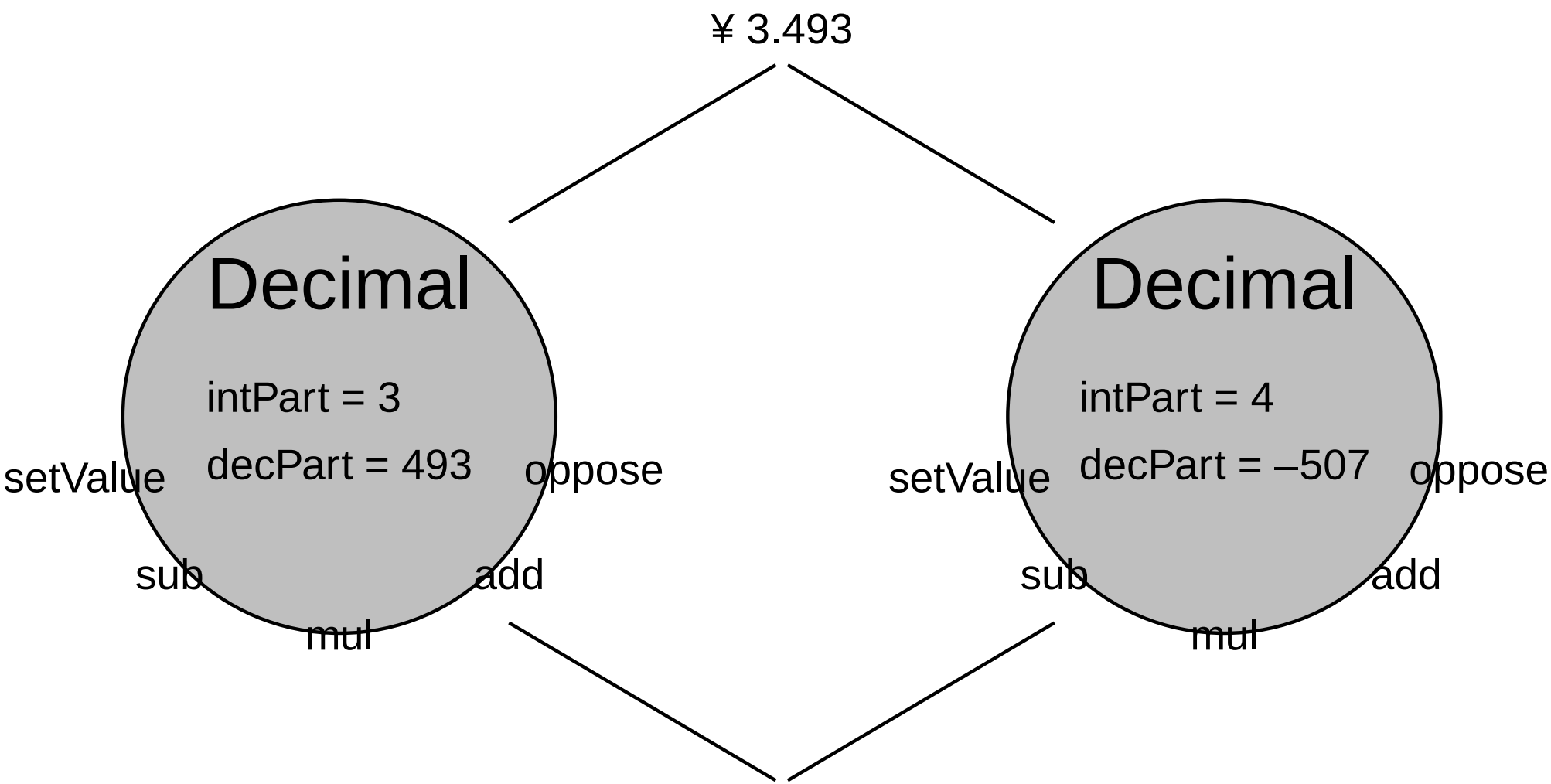
- JavaCard **には** int **型，浮動小数点型がない**
- `intPart`, `decPart` **は** short **型の変数**
- `decPart` **の有効桁数は** 3 **桁**
  - `0 <= decPart && decPart < PRECISION`
  - `PRECISION` **は** short **型の定数** 1000

addメソッド

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

addメソッドの仕様を考える



addメソッドの仕様

```
/*@ private normal_behavior
  @   requires -PRECISION < f && f < PRECISION &&
  @            -MAX_DECIMAL_NUMBER <= e && e <= MAX_DECIMAL_NUMBER &&
  @            -MAX_DECIMAL_NUMBER <= e + intPart - 1 &&
  @            e + intPart + 1 <= MAX_DECIMAL_NUMBER;
  @   modifiable intPart, decPart;
  @   ensures intPart * PRECISION + decPart ==
  @            (\old(intPart) + e) * PRECISION + \old(decPart) + f;
  @*/
private void all (short e, short f) { ... }
```

addメソッドの検証

論理式

コード

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

addメソッドの検証

論理式

$$intPart \cdot PRECISION + decPart$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

コード

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

addメソッドの検証

論理式

$$intPart \cdot PRECISION + decPart$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

$$intPart \cdot PRECISION + decPart$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

コード

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

addメソッドの検証

論理式

$$intPart \cdot PRECISION + decPart$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

$$(intPart + 1) \cdot PRECISION + decPart$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

コード

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

addメソッドの検証

論理式

$$intPart \cdot PRECISION + decPart$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

$$(intPart + 1) \cdot PRECISION$$
$$+(decPart - PRECISION)$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

コード

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

addメソッドの検証

論理式

$$(intPart - 1) \cdot PRECISION + decPart$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

$$(intPart + 1) \cdot PRECISION$$
$$+(decPart - PRECISION)$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

コード

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

addメソッドの検証

論理式

$$(intPart - 1) \cdot PRECISION + (decPart + PRECISION) = (intPart_0 + e) \cdot PRECISION + decPart_0 + f$$

$$(intPart + 1) \cdot PRECISION + (decPart - PRECISION) = (intPart_0 + e) \cdot PRECISION + decPart_0 + f$$

コード

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

addメソッドの検証

論理式

$$(decPart \leq -PRECISION) \rightarrow$$
$$(intPart - 1) \cdot PRECISION$$
$$+(decPart + PRECISION)$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

$$\wedge$$

$$(decPart > -PRECISION) \wedge (decPart \geq PRECISION) \rightarrow$$
$$(intPart + 1) \cdot PRECISION$$
$$+(decPart - PRECISION)$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

コード

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

addメソッドの検証

論理式

$$(decPart \leq -PRECISION) \rightarrow$$
$$(intPart - 1) \cdot PRECISION$$
$$+(decPart + PRECISION)$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

$$\wedge$$

$$(decPart > -PRECISION) \wedge (decPart \geq PRECISION) \rightarrow$$
$$(intPart + 1) \cdot PRECISION$$
$$+(decPart - PRECISION)$$
$$= (intPart_0 + e) \cdot PRECISION$$
$$+ decPart_0 + f$$

Q.E.D

コード

```
private void add(short e, short f) {
    intPart += e;
    decPart += f;

    if (decPart <= -PRECISION) {
        decPart += PRECISION;
        intPart--;
    }
    else if (decPart >= PRECISION) {
        decPart -= PRECISION;
        intPart++;
    }
}
```

概要

- **形式的技法**
- Hoare **論理**
- Java **プログラムの検証**
- **まとめと今後の課題**

まとめと今後の課題

まとめ:

- Krakatoa **を用いて add メソッドを検証した**
- proof obligation = **証明タブローで最後に残る証明（帰結の規則）**

今後の課題:

- **より複雑な例の検証**
  - `mul` **メソッド**
  - **インサーション・ソート，セレクション・ソート**
- **証明タブローを構成するアルゴリズムの実装**

Thank You!