VDM、SPINから始める形式手法入門
第1回：形式手法概論

国立情報学研究所　石川　冬樹
f-ishikawa@nii.ac.jp
2011年8月8日

自己紹介

- 石川 冬樹: 国立情報学研究所助教
  - http://research.nii.ac.jp/~f-ishikawa/
  - 形式手法の産業界向け教育・普及
    （主にVDMや形式仕様記述を担当）
    - トップエスイー（講義，3～6ヶ月の個別活動）
    - 日科技連SQiP（通年取り組みの演習・研究）
    - 「形式手法実践ポータル」Webサイト
      - http://formal.mri.co.jp/

おまけ: Webサービス・クラウドも（電子情報通信学会サービスコンピューティング研究会副委員長）

セミナー日程

8/8（月）	***形式手法概論***	そもそも「形式手法」とは何かを知り， 基本的な考え方を学ぶ
8/9（火）	***VDM++入門***	代表的な手法・ツールを実際に触ってみて，形式手法に共通する原則・基本的な考え方を学ぶ．また，記述や検証に関する異なるアプローチに応じた効果や，その有効範囲について考えてみる
8/10（水）	***SPIN入門***	
8/11（木）	***形式手法ツール概論***	学んだ手法・ツールとの比較を通して 他の代表的な手法・ツールにおける 異なるアプローチを紹介，俯瞰する
8/12（金）	***応用事例紹介***	近年盛んに発表されている，応用事例集や導入ガイダンスなどについて解説する

本日の内容

- 形式手法（フォーマルメソッド）の概要紹介
  - 簡単な例題を通し，何を目指してどういうアプローチをとるのか（何ではないのか）を理解する
    - 「問題解決のための共通する考え方」
    - 「状況に応じた選択肢・トレードオフ」
    - 「適用の難しさ」
  - 特定の手法・ツールには踏み込まない
    - 例題は概念的に紹介
    - 記述例・動作例は，イメージを伝える程度に紹介

目次

- 形式手法(フォーマルメソッド)とは
- 形式手法における様々なアプローチ
- 「数理論理学」?
- まとめ

形式手法(フォーマルメソッド)とは

数理論理学等に基づき品質の高い
ソフトウェアを効率よく開発するための
科学的・系統的アプローチ

システムの注目する側面を正確に，曖昧さのない言語で表現する

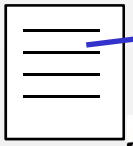
曖昧さや思い込みを開発プロセスの早期に排除し，手戻りによるコストを防ぐ

システムの満たす性質について科学的・系統的な分析・検証を行い，品質を高める

取り組む問題(１): 曖昧さ

記述においてその意味が一意に定まらない

- *記法(日本語，図等)自体で一意に決まっていない*
- *一意に定め用いる運用を徹底していない・できない*



「フラグ1がONになるまではフラグ2はOFFである」

フラグ1がONになった後に限り
必要に応じてフラグ2をONにして

フラグ1がONとなるとき
同時にフラグ2をONか！

「学生は各学期開始時に各講義の履修・非履修を選択する」
「各講義の開始時間と終了時間にはチャイムを鳴らす」
「各講義終了時に担当講師の口座への謝金振込処理を起動する」

えーと振り込みタイミングは・・・？

余談：日本語は簡単？

- 「特別な言語」を使うより日本語は「楽」？
  - もちろん皆がぱっと理解するために必須！
  - が，前例のような事態を一般に防ごうとすると，どれだけの注意事項・チェック事項が出てくる？
    - 「レビュー担当はそれらをすべて検証できる人？
    - 「開発のための正しい日本語」は，「社会人」「エンジニア」なら身についているから指針や訓練は不要 or 簡単？

取り組む問題(２): 誤り

「正しくない」振る舞いを定めてしまう

- *「何が正しくて何が正しくないか」という基準を明確に洗い出し，明記，共有していない*
- *その基準を満たしているかどうかを，(ある程度)信頼できる方法で分析，検証していない*

複数スレッドの切り替わりが特定のタイミング・順序で起きると，デッドロックしてしまう

様々な種類の予約操作において，
一部だけダブルブッキング防止のチェックが抜けている

形式手法の思想

厳密な文法・意味論，
理論的裏付けを持つ言語を用い

- 仕様や設計を「きちんと」書こうとする過程により
  - 曖昧さを解消することになる
  - 不正確さ，不整合も表面化する
  - システムに対する理解が深まる
- 仕様や設計を「きちんと」書いた結果により
  - 誤解なく情報を共有できる
  - 科学的・系統的な分析・検証ができる
  - 分析・検証をツールに支援させることができる

分析・検証内容は様々

（特に「作ってしまう」前に）
各成果物の質を保証し，引き継いでいく

形式手法のポイント(プログラムとの対比)

- プログラムを書くと
  - 対象が厳密，明確に，
    ある基準を満たすよう「一通り」書き出される
  - 型チェック，テスト等分析・検証を行える

厳密な文法・意味論や
記述制約等を持つ言語を用い

分析・検証内容は様々

➡ **上流の中間成果物(仕様・設計)でも！**

プログラムのように「全てを動かす」ことが目的ではないので，記述，分析・検証の目的に応じ

- システム内で扱う対象(範囲・観点)を絞る
- 用いる言語，手法・ツールを選ぶ

言葉:「モデル化」・「抽象化」・「捨象」

「モデル」: 計算や予測を助けるために用いられる，システムやプロセスの単純化された記述

（Oxford英英辞典の石川訳）

- 注目する側面に特化し，抽象化・単純化
  （形式手法の場合は厳密化も）

不要な実現詳細を捨象

ツールが効率的に，系統的に検査を行える

人間も重要な(難しい)本質に集中して
問題の明確化，分析を行える

言葉:「正当性」・「妥当性」

V&V
検証(Verification) & 妥当性確認(Validation)

- 検証:「成果物を正しく作っている?」
  その成果物が満たすべき性質(基準)を満たす?

  ユニットテスト・統合テスト
  形式手法における「証明」や「検査」

- 妥当性確認:「妥当な成果物を作っている?」
  自身や顧客の本来の要求に合致している?

  レビュー, 受け入れテスト, プロトタイピング
  (形式手法でも一部支援)

言葉:「数理論理学」

「数理論理学」: 論理的にものごとの性質を表現，分析するための数学的基盤

- 記述言語・ツールの提供側における拠り所
  （表現能力，効率よく正しさを保証する検証能力）

- 記述・検証を行う側にとっての基礎理解
  - 書き方が異なるが同じ意味を持つ記述の把握
  - ツール機能（オプション設定など）の理解

これらの言葉を併せると

- 「システムが何をするか」(What)を抽象的，しかし厳密なモデルとして記述
  - 「どのように」(How)の詳細(上流の過程で決めたくない・決められないこと)は捨象
- 「正しさ(正当性)」の基準を厳密に定義し，それに照らし合わせて検証を行う
- これら記述・検証において理論的裏付けがある

「何をどう書いて何をどう検証するか」は，
利用目的(採用する手法・ツール)により様々

形式手法への期待・その必要性の高まり

- ソフトウェアシステム開発への要求の高まり
  - 成果物の質・信頼性
  - 開発手法・プロセスの系統化・効率化
- 評価・認証
  - （例）ISO/IEC 15408（Common Criteria）：セキュリティの高レベルな保証の要件として，設計の形式検証が含まれている
- 多数の指針
  - セミナー，導入ガイダンス，事例集，FAQ，・・・

目次

- 形式手法(フォーマルメソッド)とは
- 形式手法における様々なアプローチ
  - 例: 形式仕様記述(VDM)
  - 例: モデル検査(SPIN)
  - 一般論
- 「数理論理学」?
- まとめ

VDM

- VDM（Vienna Development Method）
  - 1970年代にIBMウィーン研究所にて開発
  - 最近は国内でのツールサポート（CSK），適用事例（Felica）が有名
  - 「モデル規範型形式仕様記述」と分類される
    - 変数，操作に基づいたモデル化と分析・検証
  - 特徴：（今の）一般の開発者にとって身近・手軽
    - プログラミング言語に近い記法
    - インタプリタによる実行・テストを主とした検証

典型的な例題：設定

- イベント登録管理システム
  - ユーザ，部屋，イベントの情報を管理
    - 「部屋Aの定員は30名である」
    - 「イベントpは部屋Aで3/11の10～17時に行われる」
  - 予約やそのキャンセルの受付
    - 「ユーザ1が，イベントpに対して3名予約登録」
    - ➡「座席番号A-35～A-37，予約番号は20100035A」
    - 「20100035Aの予約をキャンセル」
    - 「イベントqに対し，ユーザ1～9999から5名を抽選で選び予約登録」

典型的な例題：「正しさ」の基準

- システムの状態に関する条件
  - *例：同じ時間に行われる複数イベントに対し，同じユーザが予約を行っていることはない*
- 操作に期待される効果を得るための前提条件
  - *例：キャンセル操作は，開催日が次の日以降のイベントに対してのみ実行可能である*
- 操作に期待される効果を定める条件
  - *例：抽選操作の実行結果においては，実行前に対象イベントに対し予約を行っていなかったユーザが選択されている*

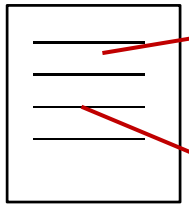
不変条件

関数・操作の事前条件，事後条件

解決しようとする問題

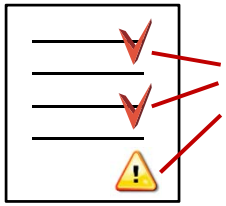
■ 定義の曖昧さ，不正確さ，不整合（誤解の原因）

例：「イベント」という言葉が，記述箇所により
異なる概念を指すのに使われてしまっている



- 「ユーザはイベントに参加登録する」
  （「VDMセミナー（8/8からの5日間）」といったひとくくり）
- 「各イベントには撮影者，補助者を割り当てる」
  （ 実は「VDMセミナー1日目1コマ目」といった各時間帯）

■ 誤り（「正しさ」の実現に関する抜け・漏れ）



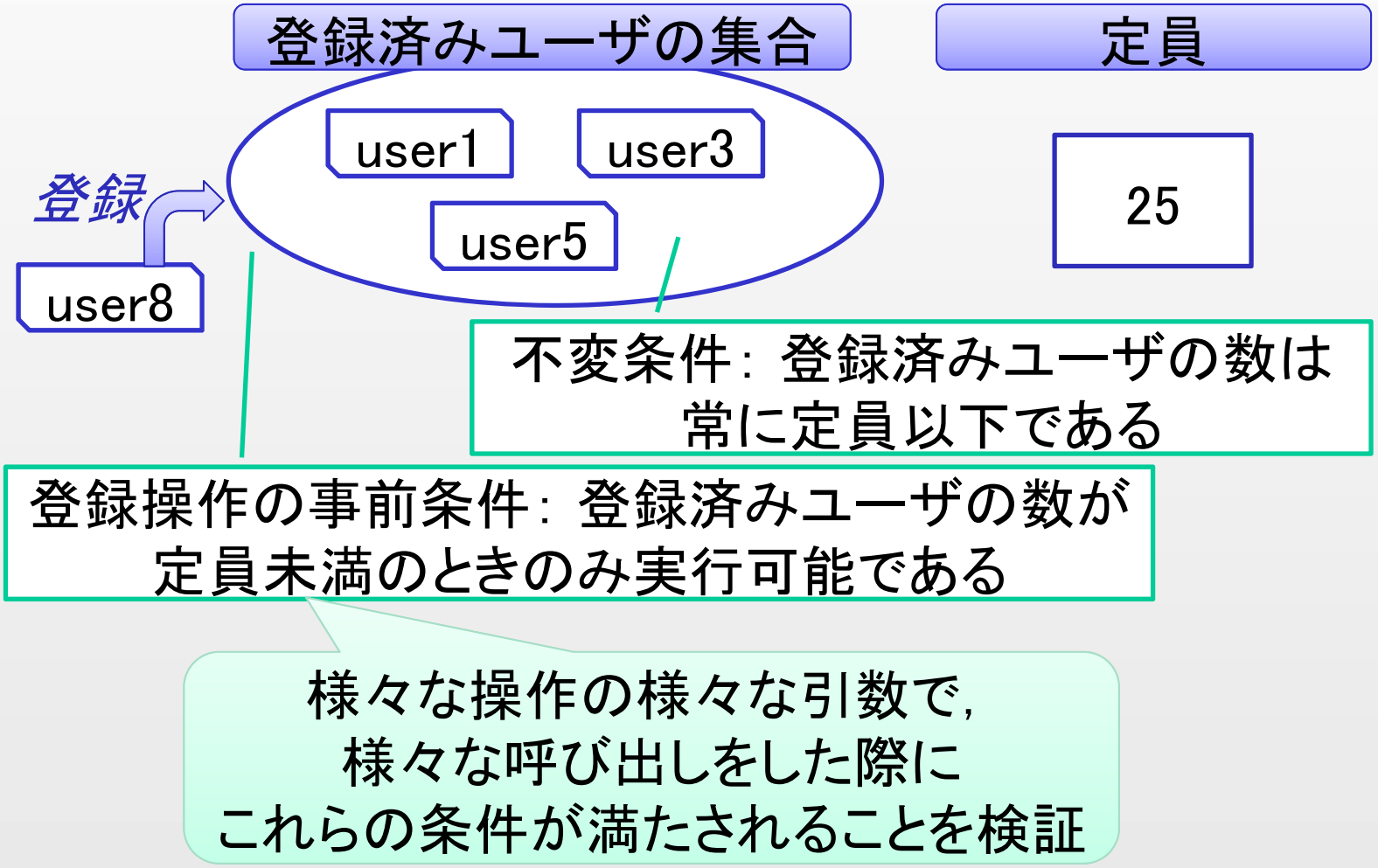
例：複数種類の予約操作のうち一部において，
ダブルブッキングチェックに関する記述が抜けている

抽象モデル化

- システムがどういう情報を保持し，個々の操作（関数・メソッド）において何をするかを記述
  - 保持される情報と，それらの間の関係（多重度や一意性など）を明記する
    - それらが計算機上でどう実現されるか（配列，ポインタ，ハッシュなど）は基本的に捨象する
  - 個々の操作が何をするか（実行結果として何が起きるか）を記述する
    - 詳細な手順・アルゴリズムは基本的に捨象する

➡ *プログラムにつながる情報を明確化，分析・検証*

抽象モデル化



VDM++言語による記述イメージ

```
class EventManager

types
  public UserID = token;

instance variables
  private registered : set of UserID;
  private capacity : nat;
  inv card registered <= capacity;

operations
  public register : UserID ==> ()
  register(user) ==
    registered := registered union {user}
  pre
    card registered < capacity
    and user not in set registered;
```

型定義

変数定義

不変条件

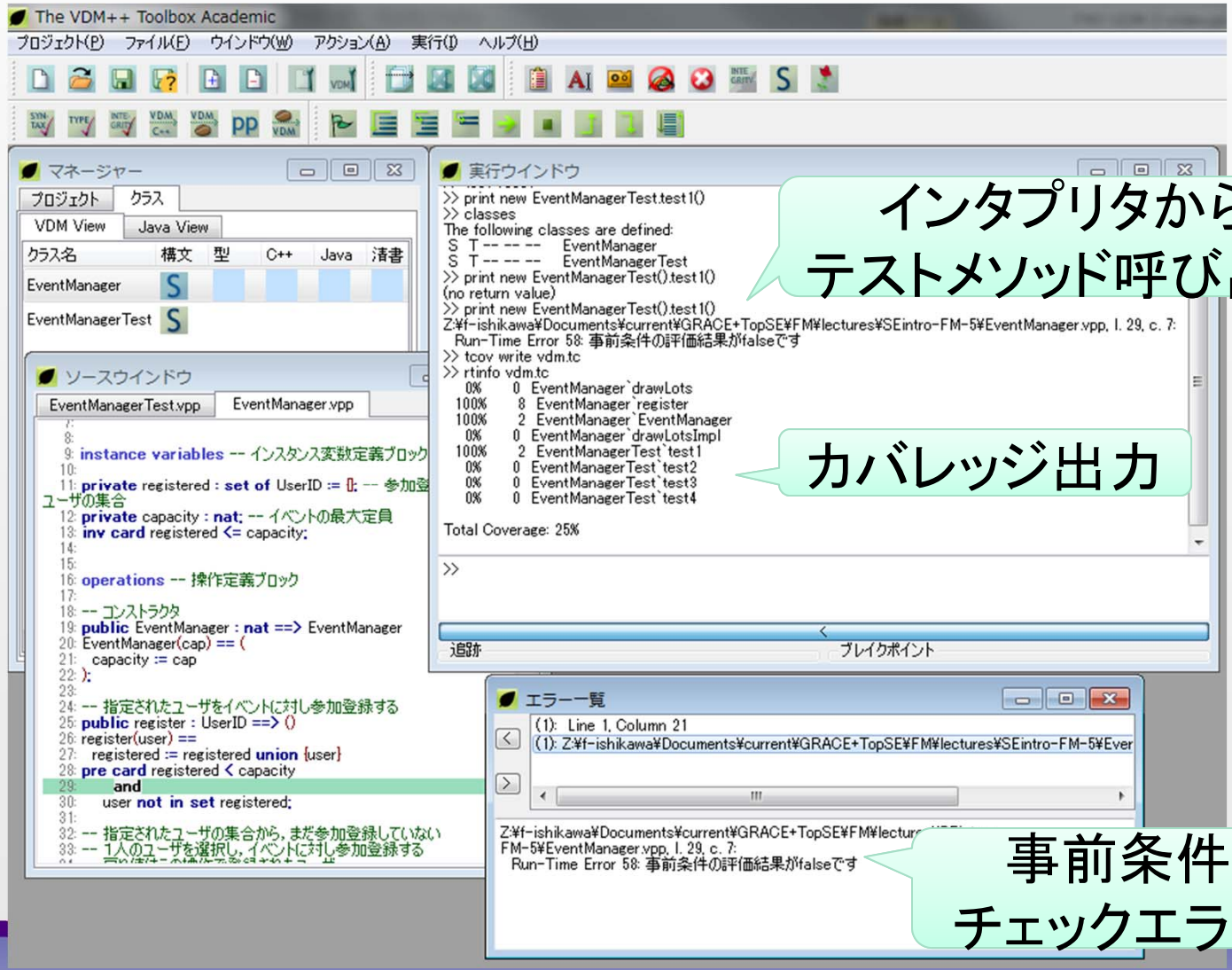
操作定義

本体＋事前条件

```
public choose :
        set of UserID ==> UserID
    choose(users) == is not yet specified
    pre
      card registered < capacity
       and
      exists user in set users
          & user not in set registered
    post
      registered =
          registered~ union {RESULT}
      and RESULT in set users
      and RESULT not in set registered~;
end EventManager
```

本体なし，事前＋事後条件

VDM++ Toolboxによる分析・検証イメージ



目次

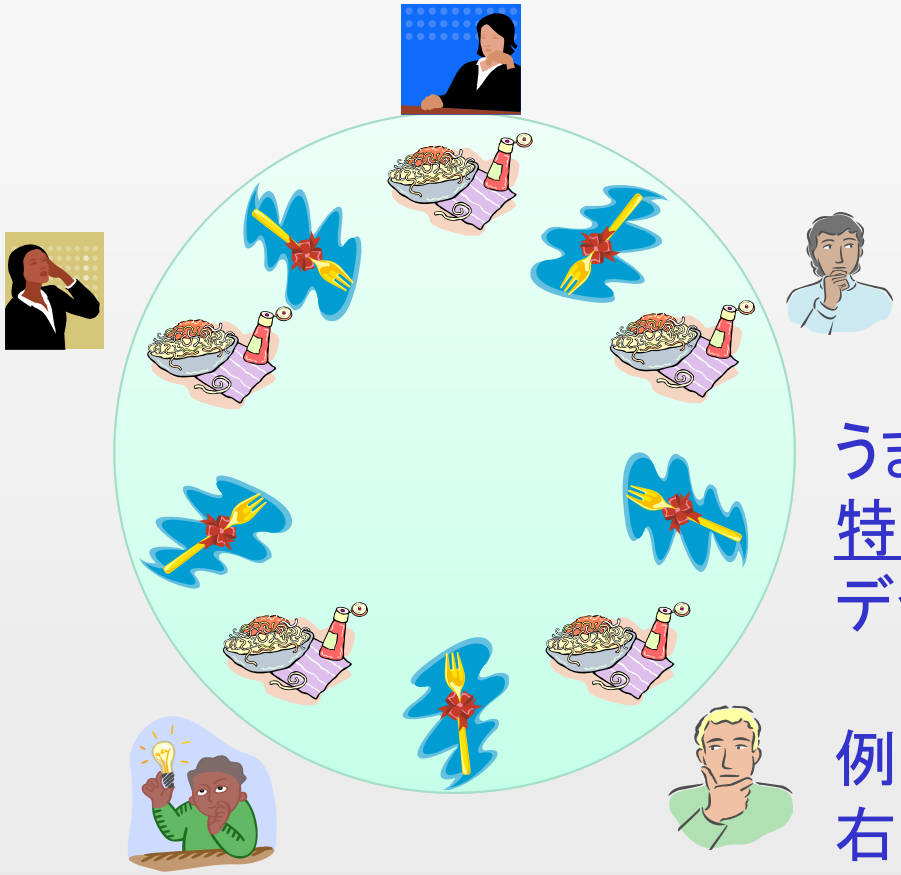
- 形式手法(フォーマルメソッド)とは
- 形式手法における様々なアプローチ
  - 例: 形式仕様記述(VDM)
  - 例: モデル検査(SPIN)
  - 一般論
- 「数理論理学」?
- まとめ

SPIN

- SPIN
  - 1980年前後に開発され，改良されてきたツール
  - IEEEの標準プロトコルにおけるバグ発見など多数の事例
  - 「モデル検査」における最も代表的なツール
    - （並行プロセスによる）複雑な状態遷移を網羅的に検証

典型的な例題：設定

有名な例：食事する哲学者



左右どちらかのフォークをとった後，もう片方をとり，
食事をしてフォークを戻す
（会話はしない）

うまく行動を定めないと，
特定の振る舞いの順序で
デッドロックやライブロック

例：全員が左のフォークを持ち
右が空くのを待ち続ける

典型的な例題：「正しさ」の基準

- デッドロックフリー
  - 誰も何もできなくなる状況に到達しない
- 進行性
  - 食べる以外の行動（様子見・断念してフォークを置く等）を全員が繰り返すだけの状況に陥らない
- 公平性
  - ある哲学者が（ある定義で）「ずっと」食べることができないような状況に陥らない

解決しようとする問題

- 特定の実行の分岐・順序で発生する誤りを把握
  - 並列・分散システムにおける相互作用
    - 複数スレッドが異なる資源のロックを確保して，互いに待ち合ってデッドロックする
    - 複数ノードがほぼ同時に互いに対しメッセージを送り，返事を待ち合ってデッドロックする

    （実装ではそもそも意図的に再現できない）
  - 複雑な条件分岐
    - 多数の行動・フラグの管理において，想定外の操作により予期しない状況に到達してしまう

抽象モデル化

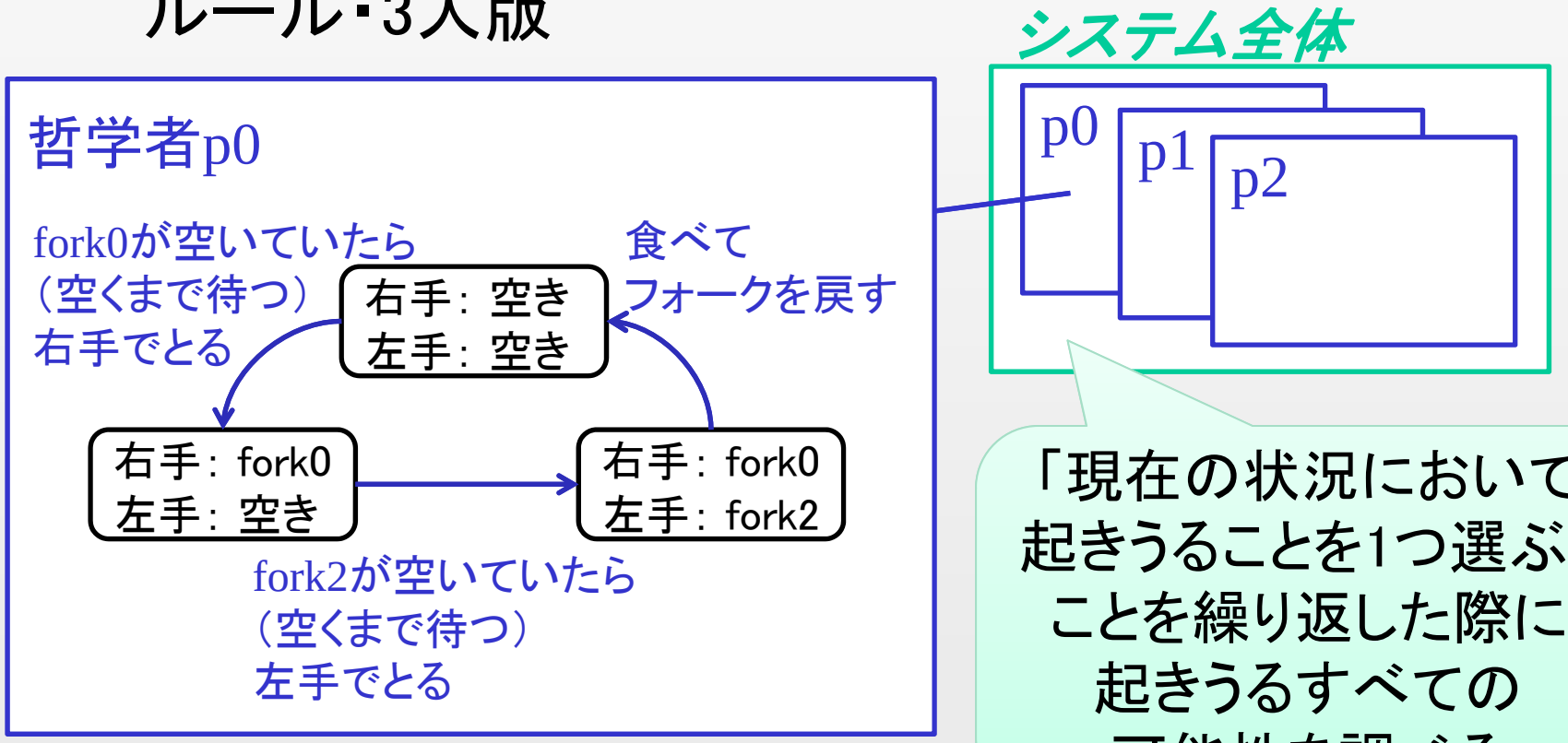
- 何が起きうるかを記述
  - 環境に対する仮定と振る舞い設計において，各状態で起きうること・起きえないことを区別する
  - 起きる順番・条件分岐だけ区別すればよい
    - 検証内容に応じ，区別する必要がないことも

      （例：パスタソースが途中で変わっても，
      デッドロックの有無には関係ない）
  - 対象とする問題上，ガード条件（成り立つまで待つ）やプロセス間通信を簡潔に記述できるとよい

➡ *網羅的に起きうる状態変化を探索・検査*

抽象モデル化

- 「右手側からフォークをとる（とったら戻さない）」ルール・3人版



SPIN(Promela言語)による記述イメージ

```
mtype = {p0, p1, p2, none};
mtype fork[3] = none;

active proctype P0(){
  do
    :: atomic{fork[0] == none -> fork[0] = p0};
       atomic{fork[2] == none -> fork[2] = p0};
       skip;
       fork[2] = none;
       fork[0] = none;
  od
}
...
```

列挙型

プロセスP0の定義

フォークが空いているならば確保（間で割り込まれず原子的に）

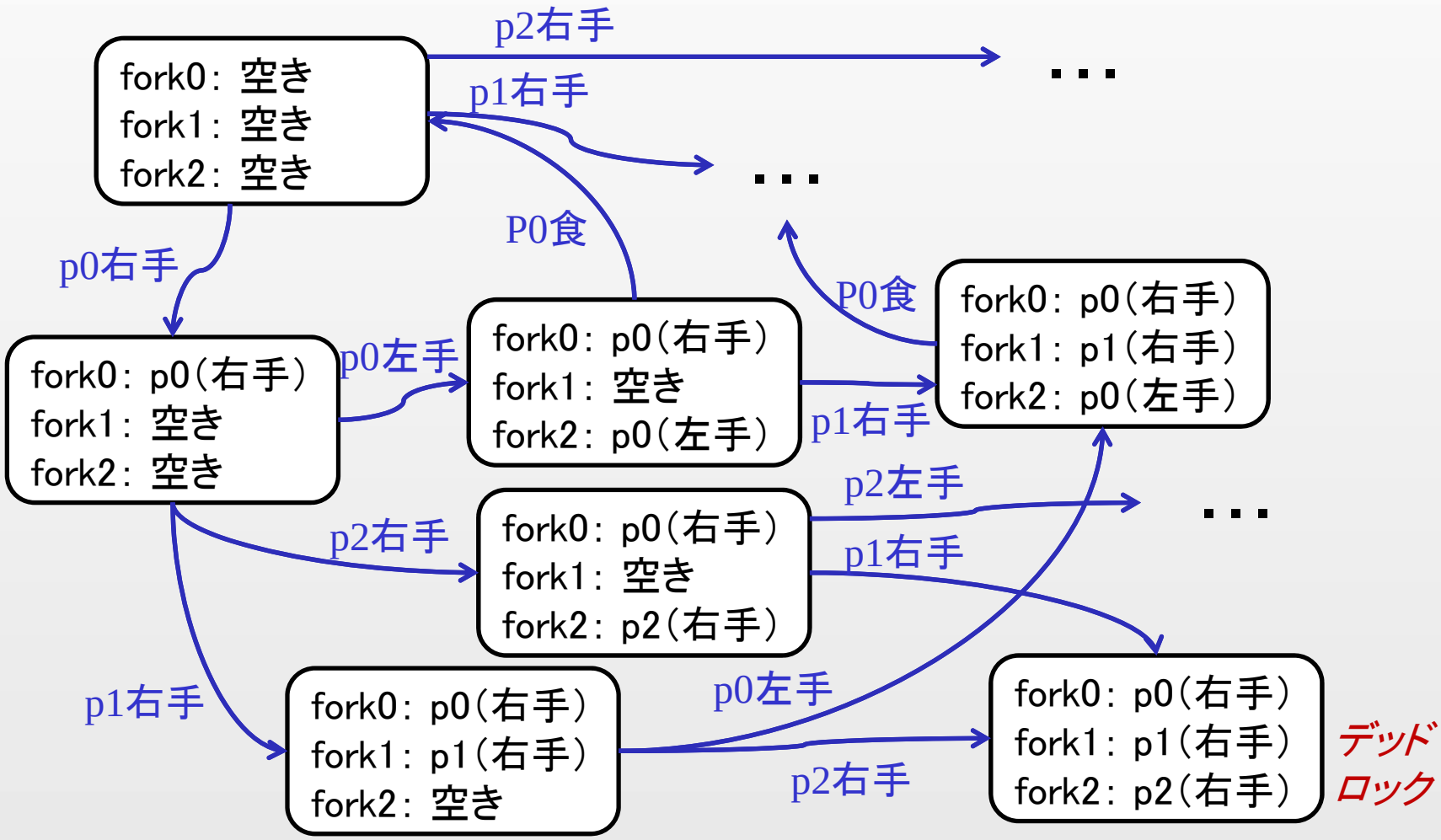
「食べる」ことは捨象

do-odには起きうることの選択肢を書く(今回は1つ)

```
[] (fork[0]==p0 -> <> fork[2]==p0)
```

検証対象の性質例：p0が右手側のフォークをとったときには「いつも」，その後「いつか」p0が左手側のフォークをとる

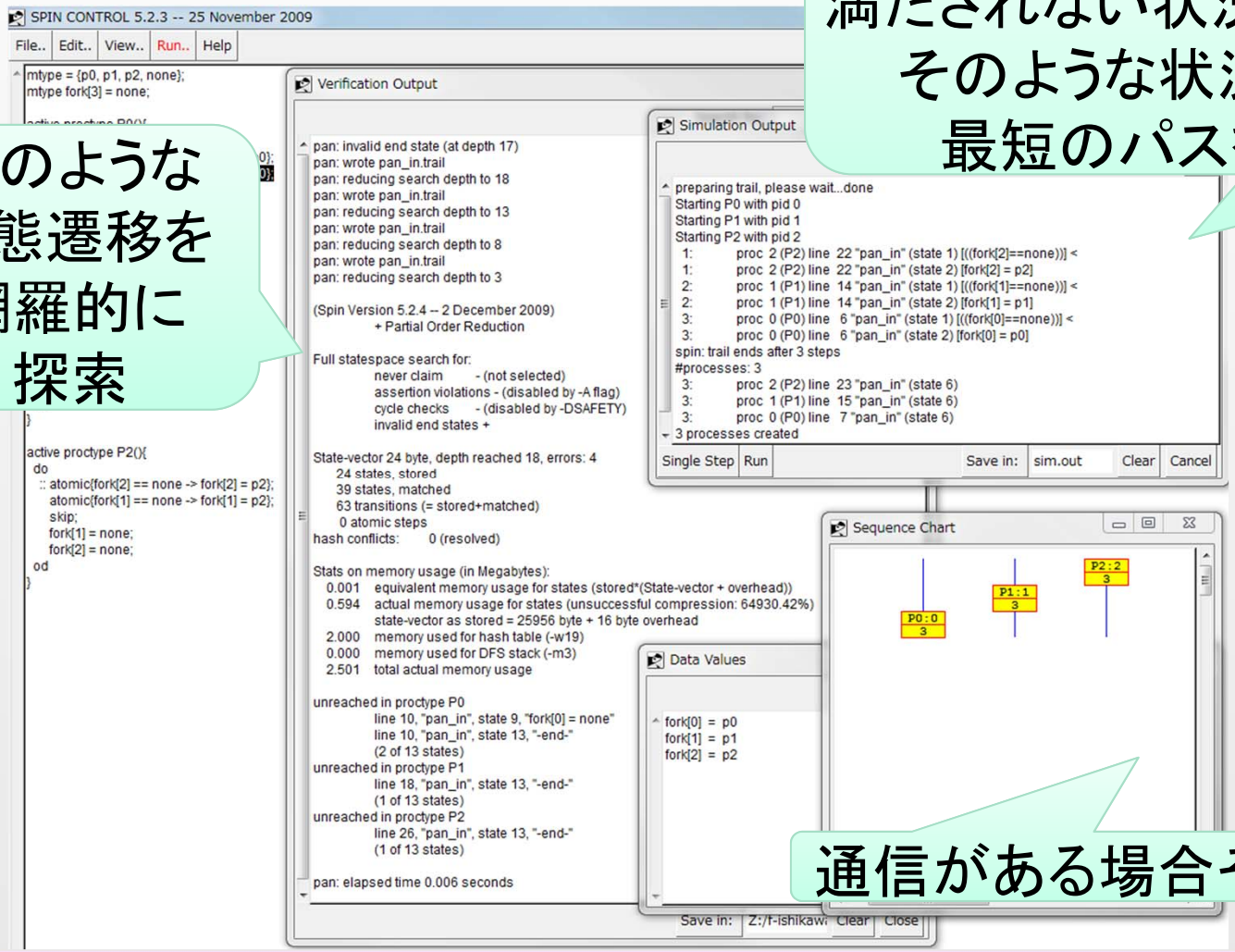
実際の状態遷移(分析・検証対象)



SPINによる検証イメージ

デッドロックや性質が
満たされない状況があれば,
そのような状況に至る
最短のパスを報告

先のような
状態遷移を
網羅的に
探索



通信がある場合それも図示

目次

- 形式手法(フォーマルメソッド)とは
- 形式手法における様々なアプローチ
  - 例: 形式仕様記述(VDM)
  - 例: モデル検査(SPIN)
  - 一般論
- 「数理論理学」?
- まとめ

モデル化・記述言語

- 大まかなグループ分け
  - プログラム同様の構造（変数・操作）
    - モデル規範型形式仕様記述手法における言語
      （VDM，B，Z，Alloyなど）
  - （特に並行プロセスの）状態遷移（や通信）
    - モデル検査ツールにおける言語
      （SPIN，LTSA，SMV，UPPAALなど）
    - プロセス代数
      （CSPなど）

注：名称は手法名，言語名，ツール名ごちゃ混ぜ

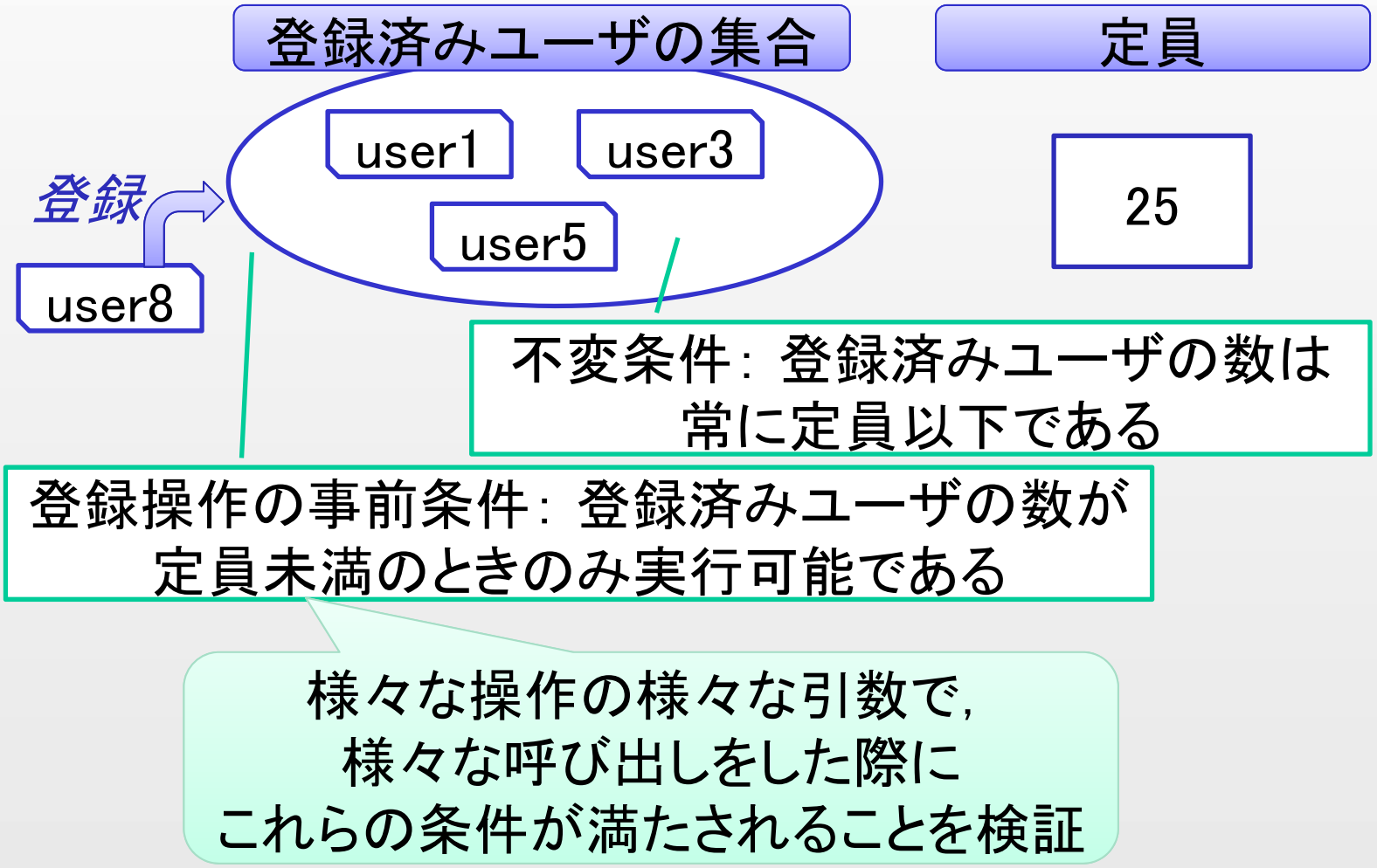
検証

- 客観的に性質が成り立つことを示すには・・・
  - 「観測」によって示す（帰納）

    様々な実行列における結果を確認する
    - インタプリタを用いたテスト，シミュレーション
    - 限られた範囲での網羅的実行（有界モデル検査）
    - 網羅的実行（モデル検査）
  - 「法則」によって示す（演繹）

    何が起きえて何が保証できるかを，一般論として理屈で推論，説明する
    - 定理証明

VDMでの例（再掲）



例:「観測」による検証

- 不変条件の観点からの操作の正しさ
  - 「正しい」状態で「正しく」予約操作を呼び出したときに,「正しくない」状態になることは決してない

状態: 登録済みユーザ集合, 定員
不変条件:
登録済みのユーザ数 ≦ 定員

予約操作(引数: ユーザ)
事前条件:
なし
定員制約付け忘れ
事後条件(or 操作本体定義):
引数ユーザが登録済みになる

「ユーザ10人が計20回操作実行」の全パターン検査

or

具体的なテスト設定

定員を越える予約操作が発生するケース

解釈実行中に不変条件違反検出

例:「規則」による検証

- 不変条件の観点からの操作の正しさ
  - 「正しい」状態で「正しく」予約操作を呼び出したときに,「正しくない」状態になることは決してない

仮定(実行前)
登録済みのユーザ数(実行前) ≦ 定員　・・・不変条件
登録済みのユーザ数(実行前) ＜ 定員　・・・事前条件
仮定(実行の意味)
登録済みのユーザ数(実行前)＋1
＝登録済みのユーザ数(実行後)
証明したい結論(実行後)
登録済みのユーザ数(実行後)≦ 定員　・・・不変条件

補足：中学数学風に書き直し

- 不変条件の観点からの操作の正しさ
  - 「正しい」状態で「正しく」予約操作を呼び出したときに，「正しくない」状態になることはない

自然数x，cについて下記が成り立つとき，

$x \leqq c$ ・・・①

$x < c$ ・・・②

$x + 1 = x'$ ・・・③

$x' \leqq c$ であることを証明せよ

x，cが自然数であるから② より $x + 1 \leqq c$

よって③ より $x' \leqq c$

本セミナーでの分類(ある程度一般的？)

- 形式仕様記述のための手法・ツール
  (VDM, B, Z, Alloy)
  - 記述:プログラム同様の構造(変数・操作)
  - 検証: 多様(テスト, 証明, 有界モデル検査)
- モデル検査のための手法・ツール
  (SPIN, LTSA, SMV, UPPAAL, CSP/FDR)
  - 記述: (特に並行プロセスの)状態遷移(や通信)
  - 検証: モデル検査

補足：実装における活用

- プログラムにおいて不変条件，事前・事後条件を明確化し検証
  - 例： JML（Java Modeling Language）
    - JMLUnitによるテスト，ESC/Javaによる証明
- プログラムにおいてモデル検査
  - 例： Java PathFinder

抽象モデルよりも詳細になる

- ポインタ，非null条件，配列インデックス，・・・
- 状態爆発等，ツールの構築がより難しい

目次

- 形式手法(フォーマルメソッド)とは
- 形式手法における様々なアプローチ
- 「数理論理学」?
- まとめ

数理論理学

- 形式手法の基礎理論
  - 勉強・理解しておくと・・・
    - 様々な言語で用いられている記法が理解できる
    - 「異なる書き方だが同じ意味」といった定理などに基づいて，簡潔に書いたり，人の記述を読んだりすることができる
    - ツールの内部で何が起こっているかをある程度知り，結果の意味や限界を判断することができる
    - （定理証明のような高度な検証を使うことができる）
    - （自前の専用ツールを作ることができる）

命題

- 命題：真偽が一意に判定できる記述
  - 真（true，T，⊤）・偽（false，F，⊥）
  - 例
    - 9は3の倍数である（真）
    - 15は4で割り切れる（偽）
    - 4で割り切れる自然数はすべて偶数である（真）
    - メソッドm1を実行して戻り値が得られるならば，その値は引数aの値を2倍にした値である
      （m1の記述が与えられれば真偽が判定できる）

システムの満たすべき性質等を，明確に定義し，その真偽を判定（真となるのか検査）したい

複合命題

- 既存の命題から新しい命題を作る
  - 『「9は3の倍数である」かつ「12は3の倍数である」』(両方とも成り立つか？ → 真)
  - 『「6は3の倍数である」または「8は3の倍数である」』(少なくとも片方成り立つか？ → 真)
  - 『「7は3の倍数である」は成り立たない』(真)
  - 『「xが4の倍数である」ならば「xは偶数である」』(前者が真のとき必ず後者が真となるか → 真)
    - 注:『「7が4の倍数である」ならば「7は偶数である」』(これは真と解釈すると決まっている)

命題論理の記号

演算	名称	読み方	真となる条件
¬A	否定	Aでない	Aが偽
A∧B	論理積	AかつB	AとBがともに真
A∨B	論理和	AまたはB	AとBの少なくとも片方が真
A⇒B	含意	AならばB	AとBがともに真であるか，Aが偽
A⇔B	同値	AとBは同値	AとBの真偽が一致

明確に一つに定まった記号を用いて，既存の命題から新たな命題を構成できる

論理演算の法則

- 例：命題A，Bについて下記2つの命題を考える

  ¬（A∧B）　　¬A∨¬B

  - AがTでBがTのとき，
    - A∧B がTなので，¬（A∧B）は⊥
    - ¬Aが⊥，¬Bが⊥なので，¬A∨¬B は⊥
  - AがTでBが⊥のとき（中略）ともにT
  - Aが⊥でBがTのとき（中略）ともにT
  - Aが⊥でBが⊥のとき（中略）ともにT

  A，Bがどんな命題でもこれら命題の真偽は一致

  ¬（A∧B）≡ ¬A∨¬B と書く

論理演算の法則(の一部)

- A∧B ≡ B∧A　A∨B ≡ B∨A　交換法則
- A∨(A∧B) ≡ A
  A∧(A∨B) ≡ A　吸収法則
- A∧(B∧C) ≡ (A∧B)∧C
  A∨(B∨C) ≡ (A∨B)∨C　結合法則
- A∨(B∧C) ≡ (A∨B)∧(A∨C)
  A∧(B∨C) ≡ (A∧B)∨(A∧C)　分配法則
- ¬(A∧B) ≡ ¬A∨¬B　ド・モルガン
  ¬(A∨B) ≡ ¬A∧¬B　の法則

見た目が違う命題の意味(真偽)が同じと判定したり意味を保って命題の形を変えたりしたい

論理演算の法則(の一部)

- A⇒B ≡ ¬A∨B
  (「ならば」の意味)

- A⇒B ≡ ¬B⇒¬A
  (対偶と元の命題との真偽は一致する)

  なお,
  - A⇒B の逆は　　B⇒A
  - A⇒B の裏は　　¬A⇒¬B
  - A⇒B の対偶は　¬B⇒¬A

  (元の命題とその逆やその裏の真偽は一致するとは限らない, 逆の対偶は裏)

補足：実際の分析にて

- 例：とあるシステムのフラグf1，f2に関し満たすべき性質を明確に書き出す

  どんな状況でメソッドm1を実行しても，その実行が完了した時点で　f1=1 ⇒ f2=0　が成り立つ

  - 以下の記述も同じ意味であることがわかる

    どんな状況でメソッドm1を実行しても，その実行が完了した時点で　¬(f1=1)∨(f2=0)　が成り立つ

  - この性質は，*m1*の実行完了時点で *f1=1* ではないときについては何も言及していない
    - *f1=1* にする処理が抜けていてもこの性質は真

補足：集合論（のイメージ）

- 集合論：要素（データ）の集まりについて述べるための論理
  - 「予約記録の集合recordsの中から，予約対象がevent1に関するものを集め，その予約を行っているユーザを集めた集合」

    $\{ r.user \mid r \in records \land r.event = event1 \}$

  - 上記集合をSとしたとき，このユーザの中でゴールド会員集合Gにも含まれているユーザ

    $S \cap G$

補足：一階述語論理（のイメージ）

- 一階述語論理：
  - 予約記録の集合recordsに含まれている予約記録はすべて，明日以降のイベントを対象としている

    $\forall r \in records \,.\, isLater(\, r.date,\, getDate()\,)$

  - 「抽選予約操作の引数usersの中に，まだ予約を行っていないユーザが少なくとも1人存在する」

    $\exists u \in users \,.\, (\, \neg\, isRegistered(u)\,)$

∀は「すべてにおいて」，∃は「あるものが存在して」

補足：時相論理（のイメージ）

- 時相論理
  - 「常に，フラグf1とf2の両方が1になっていることはない」

    □ ((¬ f1 = 1) ∨ (¬ f2 = 1))

  - 「常に，リクエストイベントreqが発生すると，その後いつかレスポンスイベントresが発生する」

    □ (req ⇒ ◇ res)

□は「いつも」，◇は「いつか」

目次

- 形式手法(フォーマルメソッド)とは
- 形式手法における様々なアプローチ
- 「数理論理学」?
- まとめ

形式手法の思想(再)

厳密な文法・意味論，
理論的裏付けを持つ言語を用い

- 仕様や設計を「きちんと」書こうとする過程により
  - 曖昧さを解消することになる
  - 不正確さ，不整合も表面化する
  - システムに対する理解が深まる
- 仕様や設計を「きちんと」書いた結果により
  - 誤解なく情報を共有できる
  - 科学的・系統的な分析・検証ができる
  - 分析・検証をツールに支援させることができる

分析・検証内容は様々

（特に「作ってしまう」前に）
各成果物の質を保証し，引き継いでいく

形式手法を扱う難しさ

- モデル化（抽象化）と適用方針
  - 「問題になっている・なりそうなところ」を同定，検証・分析の目的・内容を定義
  - 目的に対して十分かつ効率的に記述，検証・分析
  - 効率的かつ系統的に前後の成果物と連動

  結局は「基準を定めそれを満たす」ことをそれぞれがやるための道具（基準を定めるのは人間）
- ノウハウ・専門知識，体制，開発プロセス全体
  - 「今のもの」とは違う（「より難しい」とは限らない）
  - 手法・ツール，適用方針に依存する