PPL'06

フォーマルメソッドは夢か道具か

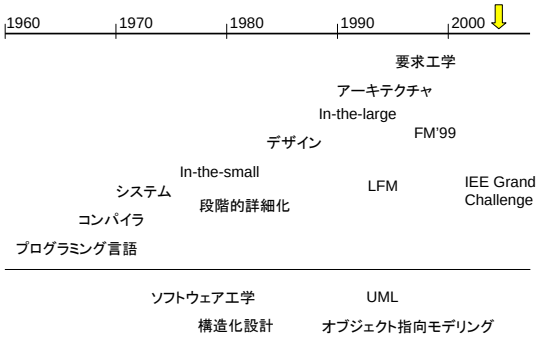
中島震
国立情報学研究所

本資料はPPL'06での発表(カテゴリー4)をベースとしている

- Bメソッドの記述例は来間啓伸氏(日立シス開研)による
- 参考文献リストを追加した

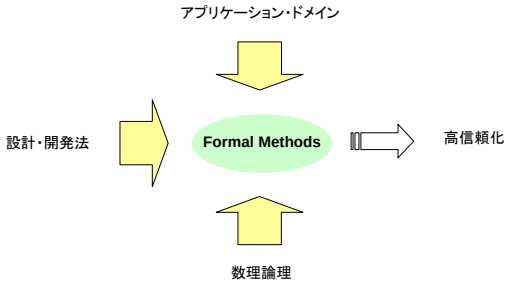
2006年3月10日改版

話の流れ

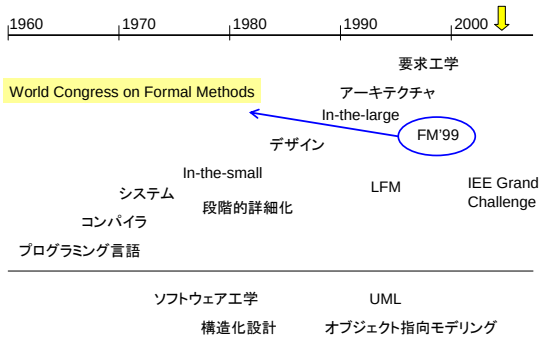


多面的な技術

Not Just Formal



話の流れ



形式手法の見方

- Tony Hoare による整理 1998-99
  - Maturity　技術の成熟さ
  - Convergence　手法の収束
  - Cumulative Progress　蓄積による進歩
- 形式手法の見方:今昔
  - 昔 → 不具合がないことを示す技術、一種の「夢」
  - 今 → 不具合を早期に発見する技術
- 実績と今後の見通し
  - 高安全性システムで多くの実績
    - 原子力発電所の制御、航空管制、自動運行制御、汎用CPU、等
  - ソフトウェアの浸透と大量販売・消費
    - 社会基盤ソフトウェアの脆弱さ → 人間社会の危機管理
    - 出荷後の不良によるリコール → ビジネスの危機管理

形式手法とその系譜

2000

初期　基礎理論　多彩な手法・成功事例　手法の収束　本格化

VDM (Vienna Definition Method)

モデル規範型 → VDM, Z, B, Raise/RSL, …

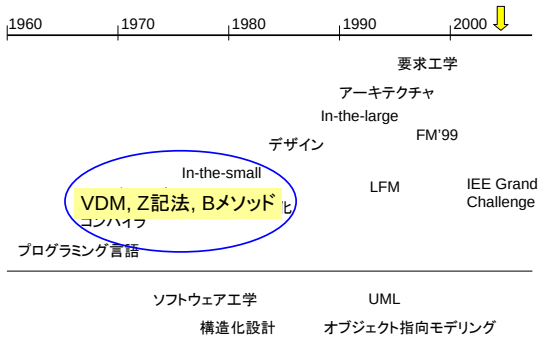
性質規範型 → 代数仕様: Clear, Larch, OBJ, CafeOBJ, CoFI, …

並行システム・インタラクション → CSP, CCS, π 計算, …

時相論理 → CTLモデル検査, SPIN, …

構造化設計　オブジェクト指向技術／UML　分散システム／Web

話の流れ



発端

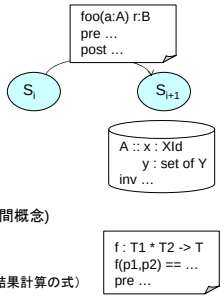
- 目的
  - プログラミング言語の意味の厳密な定義
  - コンパイラの正しさを形式検証
- 代表的な研究
  - Hoare / Dijkstra の公理的意味
    - 最弱事前条件を用いるプログラム検証
  - VDL / VDM (IBMウィーン研究所)
    - PL/I言語の表示的意味
    - コンパイラ（言語処理系）の形式記述と検証

    ⇒ Model-theoretic, model-oriented specifications
      - 抽象的な状態を表す数学的対象物とその上の操作

    （⇔ Algebraic, property-oriented specifications
      - 一連の操作間の関係で意味を定義）

VDM

- 仕様の書き方
  - 事前・事後条件（「陰関数」と呼ぶ）
  - データ不変式
- 基礎理論
  - 部分関数の論理（3値論理）
- 2つの言語
  - VDM-SL (ISO標準)
  - VDM++ : VDM-SL + オブジェクト概念 + (時間概念)
- ツール支援
  - 構文・型チェック、インタプリタ!!
    - 実行可能な関数 … 事前条件、関数本体（結果計算の式）
- 代表的な適用事例
  - 国内：Jフィッツ（トレーディング）、FeliCa（おさいふ携帯）、など



Z記法

- 仕様の書き方
  - 手続き －＞ 事前・事後条件
  - 「データ」間の静的な関係
- 基礎理論
  - Z集合論、1階述語論理
- ツール支援
  - 構文チェック、文書化
  - 証明支援ツール
    - サブセットZ記法：Z/EVES, HOL/Z, 等
- 代表的な適用事例
  - CICS (英国IBM、OLTPのAPI)
  - ODPトレーダ、RBAC (ANSI標準勧告)、など
    - 厳密な標準勧告「文書」としての役割

BirthdayBook

known : **P** NAME
birthday : NAME -+-> DATE

known = *dom* birthday

AddBirthday

△BirthdayBook
n? : NAME
d? : DATE

n? ∉ known
birthday' = birthday ∪ { n? ↦ d? }

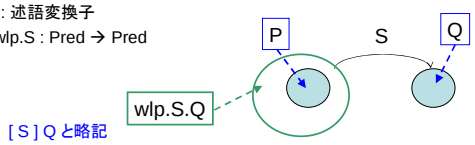
Bメソッド

- 仕様の書き方
  - 「抽象機械」と呼ぶモジュール単位
  - 事前条件、データ不変式
- 基礎理論
  - 集合論、最弱事前条件
- 開発技法
  - 段階的詳細化
- ツール支援
  - 検証条件の自動生成
  - いくつかは自動検証
- 代表的な適用事例
  - パリ地下鉄・自動運行制御

```
MACHINE
  PointSystem
VARIABLES
  granted, used
INVARIANT
  granted : NATURAL & used : NATURAL
                    & used <= granted
INITIALISATION
  granted, used := 0, 0
OPERATIONS
  sp <-- use_a_point =
    PRE used < granted
    THEN sp, used := used+1, used+1
    END;
  ap <-- grant_a_point =
    ap, granted := granted+1, granted+1
END
```

最弱事前条件

- Dijkstra の方法　P ➔ wlp.S.Q
  - P ： 事前条件 Pre-condition
  - S ： 実行文　 Statement
  - Q ： 事後条件 Post-condition
- Sの実行規則
  - wlp をSの種類ごとに定義
  - 実行後にQを満たすような(最弱)事前条件
  - wlp : 述語変換子
    wlp.S : Pred → Pred



[S] Q と略記

Bメソッドの検証条件

- 初期化　[Init] Inv
  [granted, used := 0, 0] (granted : NATURAL & used : NATURAL & used <= granted)
- オペレーション Inv ∧ Pre ➔ [Body] Inv
  (granted : NATURAL & used : NATURAL & used <= granted)
  ∧ used < granted
  ➔ [sp, used := used+1, used+1] (…)

Bの記述例

```
MACHINE    basic
SETS    NODE
CONSTANTS    Edge, ConnectedNodes
PROPERTIES
    Edge : NODE <-> NODE &
    ConnectedNodes : NODE <-> NODE &
    Edge <: ConnnectedNodes &
    (ConnectedNodes ; Edge) <: ConnectedNodes &
    !Graph . (
        Graph : NODE <-> NODE &
        Edge <: Graph &
        (Graph ; Edge) <: Graph
        => ConnectedNodes <: Graph)
ASSERTIONS
    ConnectedNodes = Edge ¥/ (ConnectedNodes ; Edge)
END
```

静的な情報構造

性質の定義

証明すべき性質

リファイメントの例

```
MACHINE    abstract1
SETS    ND
VARIABLES    Edge
INVARIANT
    Edge : ND <-> ND

INITIALISATION
    Edge := {}
OPERATIONS
    add_edge(n1, n2) =
    PRE  n1 : ND & n2 : ND
    THEN
        Edge := Edge ¥/ { n1 |-> n2 }
    END
END
```

```
REFINEMENT    refinement1
REFINES    abstract1
VARIABLES    CEdge
INVARIANT
    CEdge : ND --> POW(ND) &
    !(x, y) . ( x : ND & y : ND & x |-> y : Edge => y : CEdge(x) ) &
    !(x, y) . ( x : ND & y : ND & y : CEdge(x) => x |-> y : Edge )
INITIALISATION
    CEdge := ND * {{}}
OPERATIONS
    add_edge(n1, n2) =
    PRE  n1 : ND & n2 : ND
    THEN
        CEdge := CEdge <+ { n1 |-> (Cedge(n1) ¥/ { n2 }) }
    END
END
```

リンク不変条件 J

リファイメントの検証条件

```
MACHINE        M
VARIABLES      y
INVARIANT      I
INITIALISATION B
OPERATIONS
   T =
   PRE   P
   THEN  K
   END
END
```

```
REFINEMENT     R
VARIABLES      z
INVARIANT      J
INITIALISATION C
OPERATIONS
   U =
   PRE   Q
   THEN  L
   END
END
```

```
MACHINE        N
VARIABLES      z
INVARIANT      ∃y.(I ∧ J)
INITIALISATION C
OPERATIONS
   U =
   PRE   Q∧ ∃y.(I ∧ J ∧ P)
   THEN  L
   END
END
```

実質的な抽象機械

初期化

[B] I

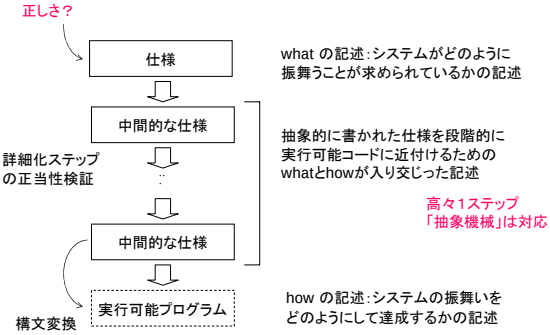
[C] ¬ [B] ¬J

オペレーション

∀y . (I ∧ P ==> [K] I)

∀y, z . (I ∧ J ∧ P ==> Q ∧ [L] ¬ [K] ¬J)

段階的詳細化の技法



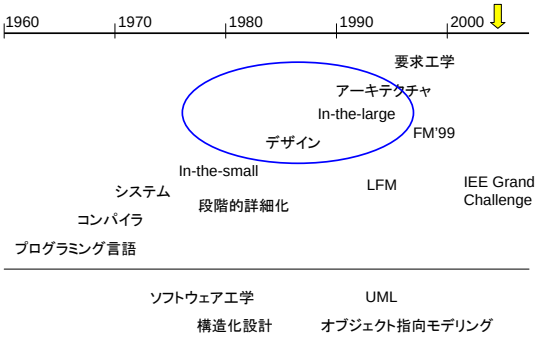
what の記述:システムがどのように振舞うことが求められているかの記述

抽象的に書かれた仕様を段階的に実行可能コードに近付けるためのwhatとhowが入り交じった記述

高々1ステップ「抽象機械」は対応

how の記述:システムの振舞いをどのようにして達成するかの記述

話の流れ

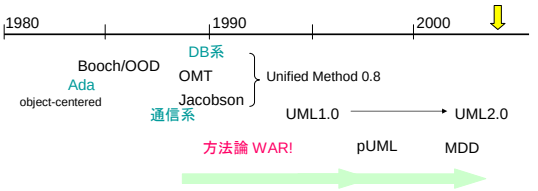


デザイン

- Design in-the-small
  - データ構造やアルゴリズムの選択・設計
    - 操作API → VDM / Z / B 等
    - プロトコルなどの振る舞い仕様 → SMV / SPIN 等
- Design in-the-large
  - ソフトウェア・アーキテクチャの表現ならびに要求仕様との関係
    - 1990年代以降のソフトウェア設計法から得た知見
    - 静的な情報構造 : クラス図
    - 動的な振る舞い : 状態遷移図
    - 大域的な構造 : データフロー図、ADL(アーキテクチャ記述言語)
    - ドメインエンジニアリング : フィーチャー図
  - 既存フォーマルメソッドの使い方を工夫

UML

- 概要
  - オブジェクト指向モデリングのためのダイアグラム記法ファミリー
- 標準化の観点
  - 表記法の統一 ⇒ UMLツール間のデータ互換性
  - モデル(デザイン) ← メタモデル(表記法) ← メタメタモデル



UMLと形式手法

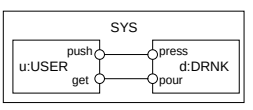
- UMLの問題点
  - ダイアグラム → 意味が曖昧
  - ファミリー言語 → ダイアグラム間の関係が曖昧
- 形式化の試み … pUML
  - 個々のダイアグラム
    - クラス図 + OCL
    - ステートダイアグラム(STD)
  - 複数ダイアグラムの整合性

  } UMLツールとして成功とは言い難い

  } Executable UML の人々は大きな関心
- 標準化の文書
  - 発想 → 決めるべきは最小、自由度を残す (⇔ 言語マニュアル)
- 利用の実際
  - プロジェクトごとに利用方法を決定 ← 方法論をカスタマイズする能力

アーキテクチャ

- 分散システム
  - 構成コンポーネントが自律動作 → 並行性の問題
  - 大域的な制御の構造(振る舞い仕様)をおさえる
- ADL(アーキテクチャ記述言語)
  - コンポーネント:並行プロセス
  - コネクタ:通信・連携基盤
- 代表的なADL
  - Wright/CSP、Darwin/FSP
- ADLを用いない研究
  - モデルチェッカの応用
    例:EJB基盤の解析、等

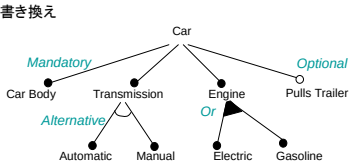


|| SYS = (u:USER || d:DRNK)
/{ u.push/d.press, u.get/d.pour }

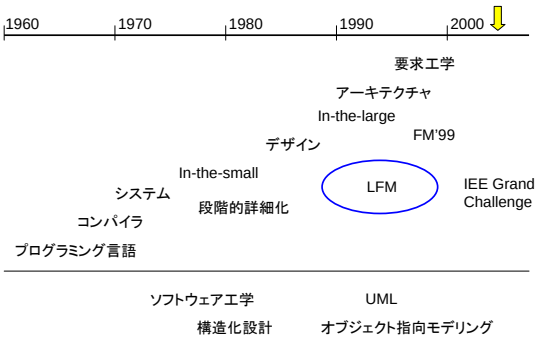
フィーチャー・ダイアグラム

- 表現の対象
  - 要求工学・ドメイン工学、SPL(ソフトウェア・プロダクトライン)
- 表現形式
  - 「フィーチャー」木構造
  - 要求仕様=コンフィギュレーション (整合性のあるフィーチャーの集まり)
- フォーマルメソッドの適用
  - 制約つき木構造の厳密な表現
  - (何らかの)標準形への書き換え

  ⇒ Z、Alloy等の研究事例



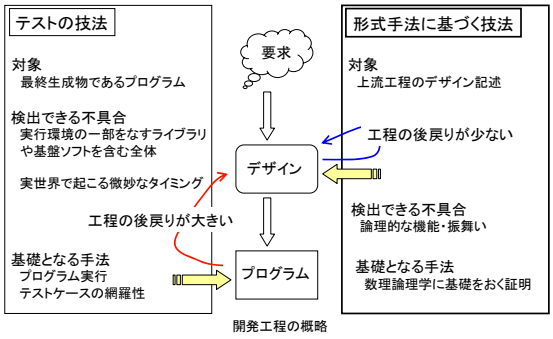
話の流れ



LFM

- Light-weight Formal Methods
  - 自動ツールを用いて不具合を早期に発見
    - false negatives 正しいものを不具合として報告する
    - false positives 不具合を発見できない
  - ⇔ 完全性・健全性を重視する立場
- LFMの立場
  - 論客たち
    - D.Jackson, J.Wing, など
  - フォーマルメソッドはデザインを対象とする系統的な「デバッグ」ツール
    - ⇔ テスト技法は開発の最終成果物（プログラム）が対象
- 同調する意見
  - 完全なシステムの記述を得ることは困難
    - モデルチェッカ ⇒ 有限状態 ⇒ 事前の抽象化・近似
    - そもそも、「完全な」記述が不可能？ … 動作環境

2つの技法

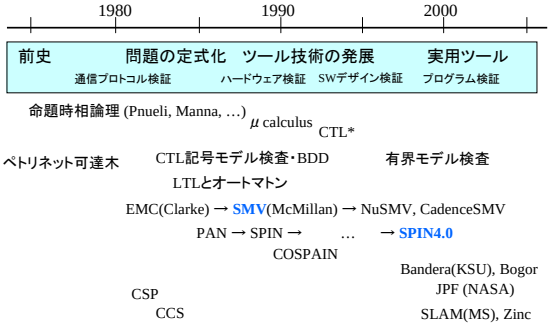


開発工程の概略

モデルチェッカ

- 振る舞い仕様の検証
  - 並行分散システム等の制御が複雑なシステムを対象
    - 振る舞い＝外部からみたイベント系列
    - イベント＝通信メッセージの送受信、メソッド起動、. . .
  - 検証対象のシステムを「有限」に限定することが前提
    - 「動作環境」のモデル化が大切
- 関連する設計技術
  - UMLステートダイアグラム (Statecharts)
  - Java マルチスレッド・プログラム
  - 協調／コラボレーション
    - オブジェクト指向フレームワーク、デザインパターン
    - 分散システムを対象とするソフトウェア・アーキテクチャ

おおまかな流れ



モデル検査技法

- 歴史的には
  - 時相論理 (CTL) の式 f を満たす状態（の集合）を求める問題

$$\{ s \in S \mid M, s \models f \}$$

Kripke構造 $M = (S, R, L)$

S：状態の集合
R：遷移関係
L：$S \rightarrow 2^{AP}$

各状態で成り立つ命題

- 検証問題
  - システムの振舞い ： Kripke構造
  - システムが満たすべき性質 ： CTLの式 f
  - システムが性質を満たすこと ： 空でない状態集合

モデル検査ツール：SMV

- SMV
  - CMUで開発・公開、もともとハードウエア検証が目的

```
ASSIGN
init(state0) := noncritical
next(state0) :=
  case
    (state0 = noncritical) : trying, noncritical;
    (state0 = trying) & (state1 = noncritical)
       : critical;
    (state0 = trying) & (state1 = trying)
   & (turn = turn0) : critical;
    (state0 = critical) : critical, noncritical;
    1 : state0;
  esac
```

状態遷移マシンによるシステム記述

CTLによる性質記述

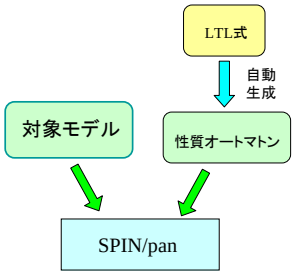
```
SPEC EF((s0 = critical) & (s1 = critical))
SPEC AG ((s0 = trying) -> AF(s0 = critical))
```

M, s |= f

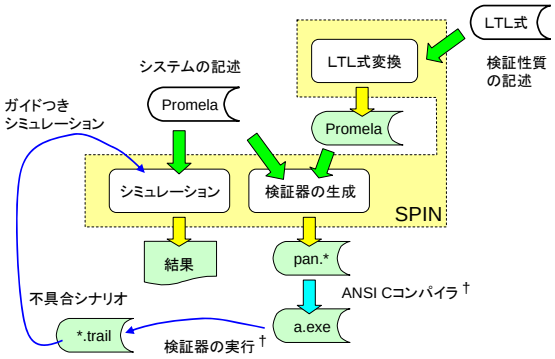
McMillan
Symbolic Model Checking, 1993

モデル検査ツール：SPIN

- 対象モデルの記述
  - 言語：Promela
  - チャネル通信オートマトン
- 性質記述
  - 性質オートマトン
  - LTL (線形時相論理) 式
- ツール機能
  - シミュレーション実行
  - **モデル検査**
    - 状態空間の網羅的な探索

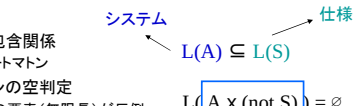


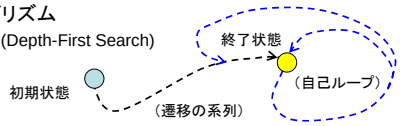
SPINツールの構成



SPINのモデル検査

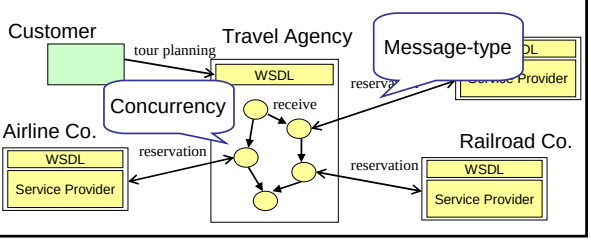
- 検証の問題
  - 受理言語の包含関係
    - Buchi オートマトン
  - 積オートマトンの空判定
    - 受理言語の要素（無限長）が反例
- 判定の方法
  - 初期状態から到達可能な終了状態を求める
  - 当該の終了状態が自己ループの要素であることを確認する
- SPINのアルゴリズム
  - Double DFS (Depth-First Search)



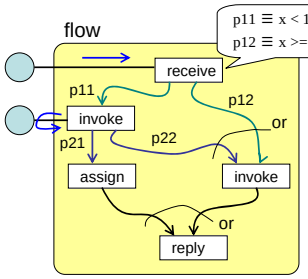


WS-BPEL の振る舞い解析

- 正しさの観点
  - Message-type Conformance
  - Distributed Collaboration ← モデル検査法の応用
- 方法
  - WS-BPEL から振る舞い仕様を抽出しPromelaに変換



環境モデルと抽象化



x：入力メッセージに含まれる値
→ 解析時には未確定

変数の解析・抽象化

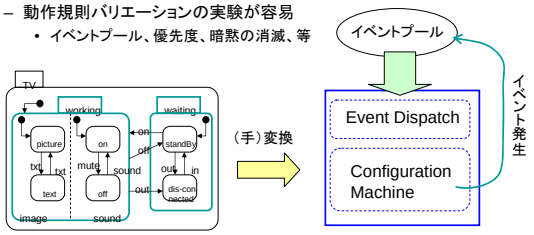
Promela断片

```
…
customer ? request;
if :: p11 = true :: p11 = false fi;
p12 = !p11;
…
```

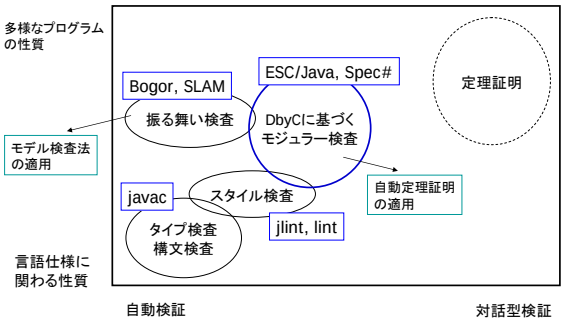
```
<receive partnerLink="customer" portType="loanServicePT" operation="request" variable="request" …>
    <source linkName="receive-to-assess" tC= "getVariableData('request','amount')&lt; 10000" />
    <source linkName="receive-to-approval" tC= "getVariableData('request','amount')&ge;10000" />
</receive>
```

セマンティック・プロトタイピング

- UML/STD (Statecharts) のモデル検査
  - 動作規則の基本は Synchrony Hypothesis
  - 多種多様なバリエーション → 目的に合わせた解釈
- Promela/SPINによるインタプリタ
  - 動作規則バリエーションの実験が容易
    - イベントプール、優先度、暗黙の消滅、等



プログラムの静的検査・検証



拡張静的チェッカ

- Extended Static Checker
  - ESC/Modula-3 → ESC/Java (DEC SRC)
  - ESC/Java2
- そもそも
  - LFMのプログラム検証ツール
    - 注釈仕様 ⇔ プログラム本体
    - Hoare / Dijkstra アプローチ
- ESC/Java
  - プログラムの書き方 : Design by Constract
  - 注釈仕様 : JML （のサブセット）
  - 適度な軽さで自動チェック
    - ループの取り扱い → 展開方式で簡単化
    - オブジェクト不変式 → 怪しい
    - クラス継承、例外処理、実行時の性質 → 考慮

Java + JML

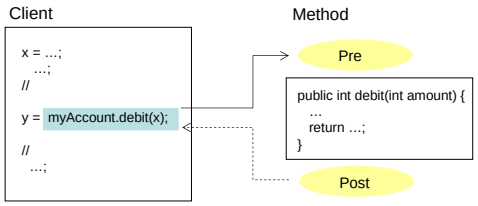
```
public class BankAccount {
  static final int ERR = -1;
  private /*@ spec_public */ int bal = 0;
//@ invariant bal >= 0;
//@ invariant bal < MAX;

//@ requires x >= 0 && x < MAX;
//@ ensures bal == x;
  public BankAccount (int x) { bal = x; }

//@ requires x > 0;
//@ assignable bal;
/*@ ensures (¥old(bal)+x < MAX)  ==> (bal == ¥old(bal) + x) && ¥result == bal);
  @ ensures !(¥old(bal)+x < MAX)  ==> ¥result == ERR;
  @*/
  public int deposit (int x) {
      if((x > 0) && (bal < MAX – x)) return setBalance(bal + x);
      else return ERR;
  }
```

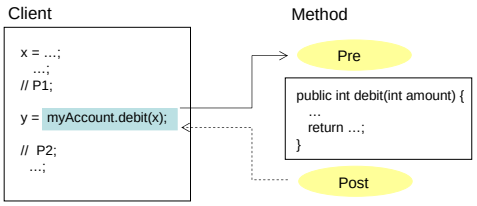
Design by Contract (DbyC)

- Client must ensure Pre and may assume Post
- Method may assume Pre and must ensure Post

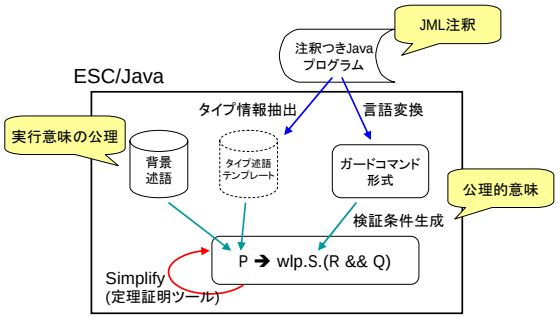


モジュラー検証

- Client : P1 ➔ Pre, Pre && Post ➔ P2
- Method : Pre ➔ wp.Body.Post



ESC/Javaツールの構成

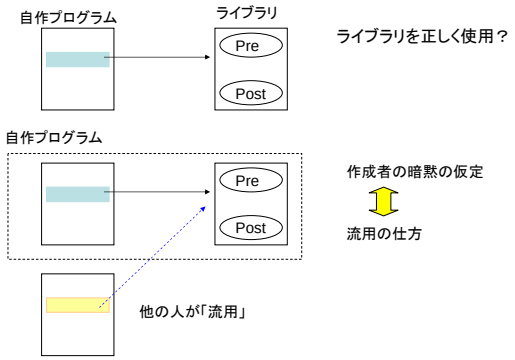


背景述語の中身

- Javaプログラム共通の性質
  - オブジェクトのメモリ内表現:メンバ変数、
  - タイプ関係:唯一性、サブタイプ関係、
  - 組込みのタイプに関する性質:bool, nat, int
  - オブジェクトのメモリ内での存在(allocation state)
- ユーザ定義クラスに依存する性質
  - ユーザ定義タイプに関わる性質
  - ユーザ定義クラスのメンバ変数に関わる性質

⇨ 公理スキーマをインスタンシエート

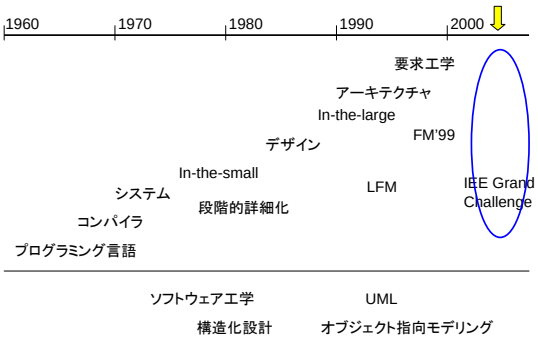
使い方の例



ESC/Java2の面白さと限界

- 手間に応じて詳細な検証が可能
  - 注釈仕様なし －＞ 背景述語により基本的な性質を解析可能
  - 注釈を付与 －＞ プログラム作成者の意図を反映
- 検証能力の限界
  - 検知できない不具合があり誤って正しい結論 (false positive)
  - 実際にありえない状況での不具合を警告 (false negative)
- 対象とする性質の限界
  - マルチスレッドJavaプログラム
    - 並行システム特有のデッドロック等の不具合
    - メソッドの実行系列(振る舞い仕様)に関する解析が必要
      ➔ DbyCの考え方になじまない

話の流れ

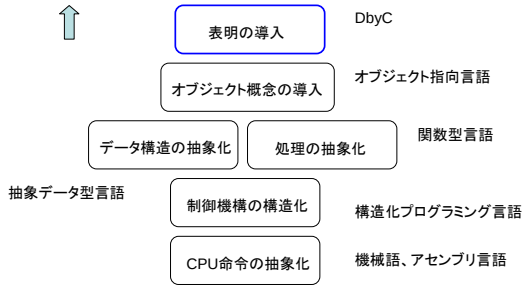


夢の道具

- 英国 Grand Challenge for Computing Research
  - T. Hoare and R. Milner
- Dependable Software Evolution
  - 15年間プロジェクト (2003 －)
    - 理論研究から実用ツールまで
  - Verifying Compiler
    - 表明つきプログラムを対象とする静的検証コンパイラ
- 具体例？
  - The Singularity Project (マイクロソフト研究所)
    - Sing# (表明つきプログラミング言語Spec#の拡張) によるOS開発

プログラミング言語の発展

プログラム作成方法の「言語化」



産業界での話題

日経エレクトロニクス
2005年12月19日号

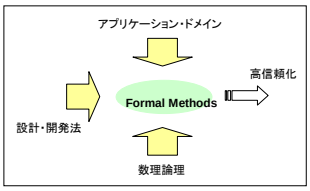
形式手法に関する特集記事

VDM
モデル検査、など

おわりに

- 取り巻く環境
  - システムの不具合が多発 → 安全性への危機意識
  - 戦略的な技術：メモリ、スパコン、インターネット、その次
- 産業界の見方
  - 信頼性向上を目的とする手法のひとつ、他手法との組み合わせ・使い分け
  - IEC 61508 (機能安全規格) → SIL4 で形式手法の利用を強く推奨
- アカデミックの世界
  - The Verifying Compiler
  - どこまで「夢」に近づけるか？

広い範囲の研究テーマ



参考文献（1）

- VDM
  J. フィッツジェラルド, P. ラーセン：ソフトウェア開発のモデル化技法, 岩波書店2003
  荒木、張：プログラム仕様記述論, オーム社2002
- Z記法
  J. Spivey : The Z Notation (2ed), Prentice Hall 1992.
- Bメソッド
  J.-R. Abrial : The B Book, Cambridge 1996.
  S. Schneider : the b-method, palgrave 2001.
  来間：形式仕様記述ーBメソッド, 近代科学社（2006発刊予定）
- SPIN
  G. Holzmann : The SPIN Model-Checker, Addison-Wesley 2004.
- ESC/Java
  中島：ソフトウェアツール紹介 ESC/Java2, コンピュータソフトウェア（2006掲載予定）

参考文献（2）

- フォーマルメソッド一般
  C. Jones : Scientific Decisions in Characterise VDM, FM'99, LNCS 1708, 1999.
  J. Rushby : Mechanized Formal Methods: Where Next?, FM'99, LNCS 1708, 1999.
  D. Jackson and J. Wing : Light-weight Formal Methods, IEEE Computer, 29(4), pages 16-30, April 1996.
  M. Huth and M. Ryan: Logic in Computer Science (2ed.), Cambridge 2004.
- フォーマルメソッドに関する章を含む邦書
  玉井：ソフトウェア工学の基礎, 岩波書店2004
  米田, 梶原, 土屋：ディペンダブルシステム, 共立出版2005
- 本発表者によるチュートリアル・サーベイ
  中島：オブジェクト指向デザインと形式手法、コンピュータソフトウェア (2001)
  中島：モデル検査法のソフトウェア開発への応用、コンピュータソフトウェア (2006)

→ この2編の参考文献リストも参考にして下さい。