SIGSE-RE-WG (2004.7.15)

モデル検査技術に基づくソフトウェア検証の概要

国立情報学研究所
科学技術振興機構さきがけ
中島 震

内容

- モデル検査技術の概要
- 事例の紹介
- おわりに

応用のサーベイ
SPINの概要
モデル検査の概要

参考資料 (中島によるチュートリアル・解説など):
(1) 日本ソフトウェア科学会DSW'04、(2004年2月)
(2) コンピュータソフトウェア、(2004年3月)
(3) 情報処理、(2004年7月)

はじめに

- システムの信頼性を高める技術
  - テスト技術
    - 最終生成物 (プログラム) が対象
    - 網羅性に問題:テストケース
  - 形式手法、形式検証技術
    - デザインが対象 → 開発上流工程での信頼性向上
    - テスト技術より良い網羅性
- 本発表の立場
  - ○:モデル検査技術の概要、使い方の事例
  - ×:モデル検査技術の理論的な側面、モデル検査ツールの仕組み

形式検証技術への期待

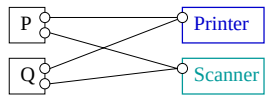
- 形式検証技術
  - 昔: 誤りがないことを示す技術
  - 今: 誤りを早期に発見する技術 → 開発上流工程
- 実用性
  - 形式検証は仕様作成コストが高い
    - 厳密な「仕様」
  - 高安全性システムで実績
    - 原子力発電所の制御、航空管制、汎用CPU、自動運行制御、等
  - 今後の可能性
    - 大量販売、大量消費
    - もし、不良があると、リコールや和解金により、企業の存亡に関る

形式検証技術の現状

- 検証技術の発展
  - 演繹手法 ... 対話的な証明のガイドが必要 → 実用化の障壁
  - モデル検査手法 ... 自動化が可能 → 適用可能性が大、注目技術
    - 不具合がある場合、反例が求まる → 修正の参考情報として有益
- 現状
  - モデル検査ツールが実用化へ
    SMV、SPIN、LTSA、CWB、FDR、SAL/PVS 等
  - 通信プロトコル、分散アルゴリズム、ソフトウエア・アーキテクチャを対象とするモデル検査検証の成功事例が蓄積
  - 「形式検証」は高度な差別化技術 → 内製化の要求
    - 検証ツールに不具合があったら ... !?
  - 細かい改善技法や現実問題への対応といったノウハウが大切
    - 経験しないとわからない ... !!

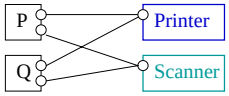
自動検証の方法の例 (1)

- 例題
  - PとQが2つの資源を同時利用、両方が同時に必要
  - 資源獲得の順序によっては デッドロック
  - 例: P -> Printer, Q -> Scanner, P -> Scanner
- 解析のための抽象化
  - 資源アクセスをイベントとするイベント系列



自動検証の方法の例 (2)

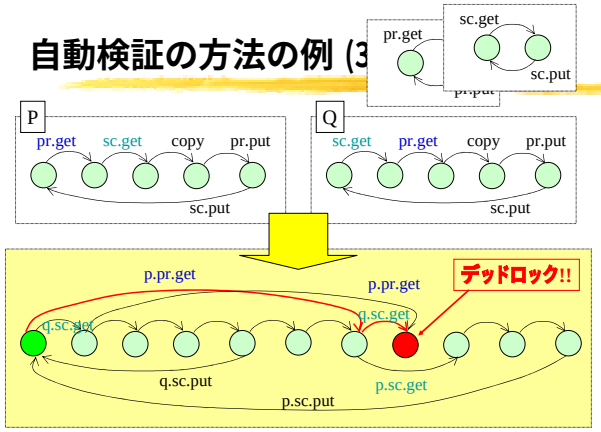
- 仕様の記述例
  - イベントの送受に着目した抽象化



システム記述

```
chan pr = [SYNCH] of { short };  chan sc = [SYNCH] of { short };
proctype P () { endLoop: do :: pr!GET; sc!GET -> copy -> pr!PUT; sc!PUT od }
proctype Q () { endLoop: do :: sc!GET; pr!GET -> copy -> pr!PUT; sc!PUT od }
proctype Printer () { endLoop: do :: pr?GET -> pr?PUT od }
proctype Scanner () { endLoop: do :: sc?GET -> sc?PUT od }
```

自動検証の方法の例 (3



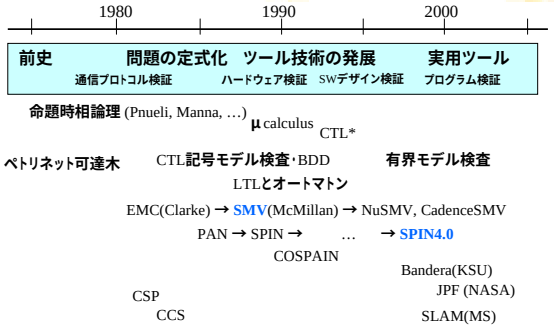
自動検証の方法の例 (4)

- 不具合に至るシナリオ
  - 初期状態から始まりデッドロックに至る最短のイベント系列

p.pr.get → q.sc.get
q.sc.get → p.pr.get

- 要するに
  - ステップ1：解析のための抽象化
  - ステップ2：対象システムの作成
    - 有限状態モデル (LTS)の内部表現に変換
  - ステップ3：状態空間の網羅的な探索
    - 予め決められた「観点」からのチェック：デッドロック、無限ループ、等

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

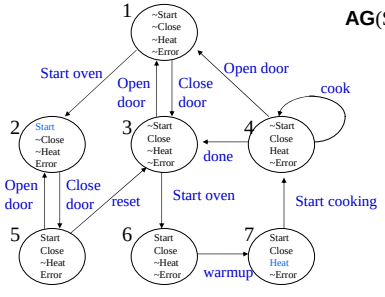
$\{ s \in S \mid M, s \models f \}$

Kripke構造　$M = (S, R, L)$

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

- 関連する技術
  - オートマトンの受理言語の包含関係判定
  - 他の関係 (プロセス代数の観測等価、等) の判定
- 最近は
  - 有限の状態遷移系を対象とする網羅的な探索技法

電子レンジの例



McMillan
Symbolic Model Checking, 1993

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

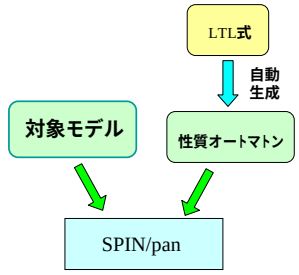
```
SPEC EF((s0 = critical) & (s1 = critical))
SPEC AG ((s0 = trying) -> AF(s0 = critical))
```

M, s |= f

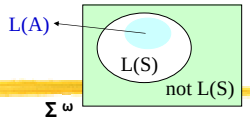
G. Holzmann
The SPIN Model Checker,
Addison Wesley 2004.

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の基礎



- Buchi オートマトン
  - 無限長の語を扱う有限状態オートマトン
  - システムならびに検証性質の双方を同じ枠組みで表現
- 検証とは
  - 受理言語の包含関係

システム　仕様

$L(A) \subseteq L(S)$

$L(A) \cap (\text{not } L(S)) = \varnothing$

never-claim

積オートマトンの空判定

「空」でない場合、その要素が反例

$L(A * (\text{not } S)) = \varnothing$

満たしてはならない性質
(not S … bad behavior)

SPINの時相論理

$(\sigma, j) \models p$　命題pが状態jで成り立つ

$\sigma = (s_0, s_1, s_2, \ldots)$　状態の列

always

$(\sigma, j) \models [] p$

$\forall\, k \geq j,\ (\sigma, k) \models p$

eventually

$(\sigma, j) \models <> p$

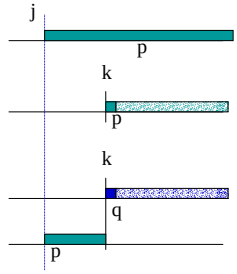
$\exists\, k \geq j,\ (\sigma, k) \models p$

strong until

$(\sigma, j) \models p\ U\ q$

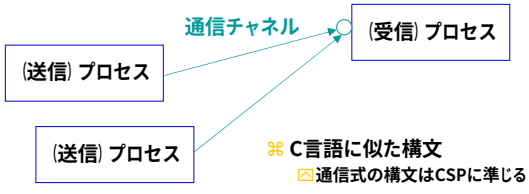
$\exists\, k \geq j,\ (\sigma, k) \models q$

$k > \forall\, i \geq j,\ (\sigma, i) \models p$



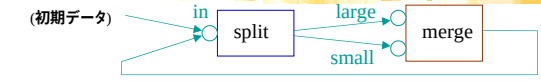
仕様記述言語 Promela

- チャネル通信オートマトン
  - 非同期通信　(同期通信も可能)
  - 一方向、名前つき、データ型つき



- C言語に似た構文
  - 通信式の構文はCSPに準じる

Promelaの雰囲気：みかけ




```
proctype split ()
{ short cargo;
  do
  :: in?cargo ->
     if
      :: (cargo >= N) -> large!cargo
      :: (cargo < N) -> small!cargo
     fi
  od
}
```

```
init {
  run split(); run merge();
  in!13; in!10; in!18;
  in!15; in!20  }
```

```
chan in = [SIZE] of { short };
chan large = [SIZE] of { short };
chan small = [SIZE] of { short };
```

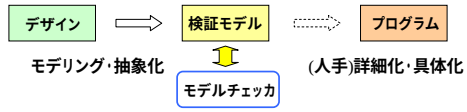
```
proctype merge ()
{ short cargo;
  do
   :: if  :: large?cargo  :: small?cargo fi;
      in!cargo
  od
}
```

内容

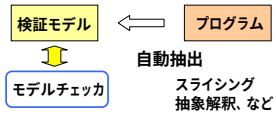
- モデル検査技術の概要
- **事例の紹介**
- おわりに

ソフトウェア検証：2つの見方

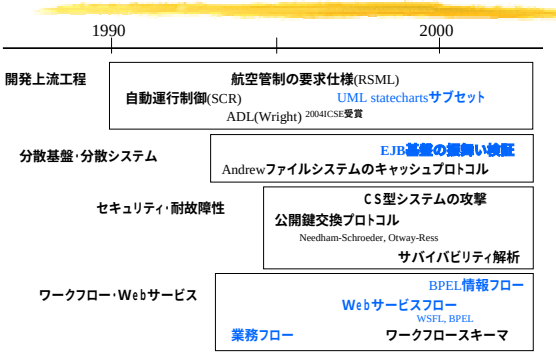
- "Classic" Model-Checking ⇒ デザイン検証



- "Modern" Model-Checking ⇒ プログラム検証



デザイン検証への応用事例

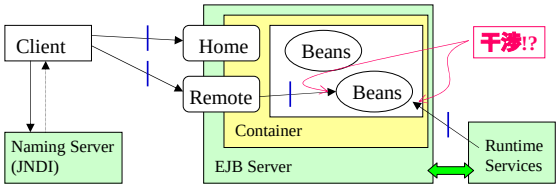


具体的な適用事例

- **EJBコンポーネント基盤の振舞い検証**
  - S.Nakajima and T.Tamai :
    Behavioural Analysis of the Enterprise JavaBeans Component Architecture, Proc. SPIN2001, pp.163-182 (2001).
  - 中島、玉井：
    EJBコンポーネントアーキテクチャのSPINによる振舞い解析、コンピュータソフトウェア、Vol.19, No.2, pp.2-18 (2002).
- **本事例を紹介する目的**
  - 既存の仕様書からSPINを用いた検証方法の手順
  - 形式仕様ならびに検証の過程から得る知見

コンポーネント基盤の振舞い仕様検証

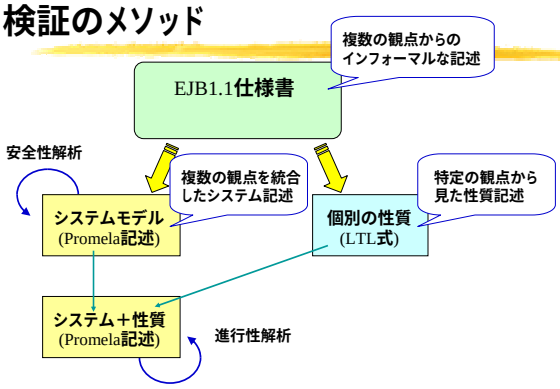
- **EJBコンポーネントフレームワークの検証**
  - EJB (Enterprise JavaBeans™): 分散サーバのコンポーネント基盤
  - 仕様書が規定する振舞い（メソッド起動系列）を形式化し自動検証
  - クライアント要求と実行時サービスが干渉しないことが正しさの基準
    → 曖昧さや場合分けの漏れがあることを発見



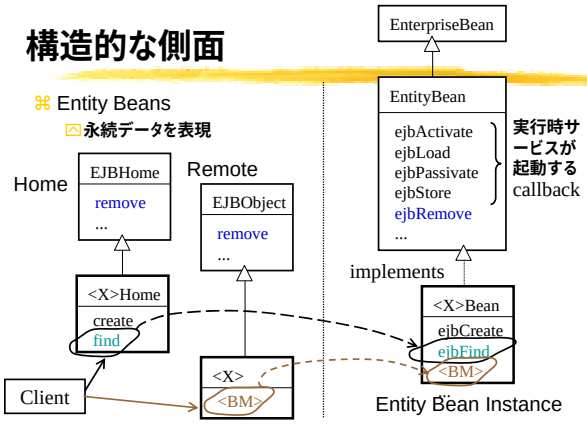
研究の背景

- **コンポーネント技術**
  - 狙い：分散アプリケーション、オブジェクト指向再利用
  - 核技術：コンポーネントモデル、統合フレームワーク基盤
  - 具体例：COM+、EJB、OMG CORBA Component、...
- **ドキュメント**
  - インフォーマル：自然言語、ダイアグラム
  - 問題点：曖昧さ、矛盾、誤り、等
- **形式検証への期待と適用例**
  - 仕様の厳密な表現、自動検証による不具合の発見
  - Sullivan&Jackson (COM, Z/Alloy), Sousa&Garlan (EJB, Wright・CSP)

検証のメソッド



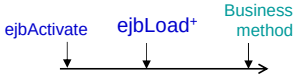
構造的な側面



- Entity Beans
  - 永続データを表現

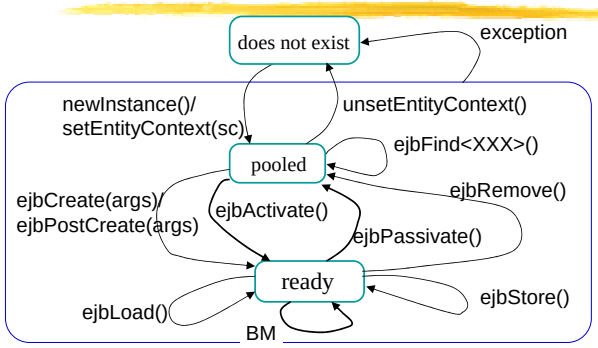
振舞いの側面

- 状態遷移ダイアグラム
  - Entity Beansのライフサイクルを明示
- オブジェクト相互作用ダイアグラム
  - 参加オブジェクトを明示したシナリオ
- 自然言語
  - Java APIの説明
  - 時間的な制約条件



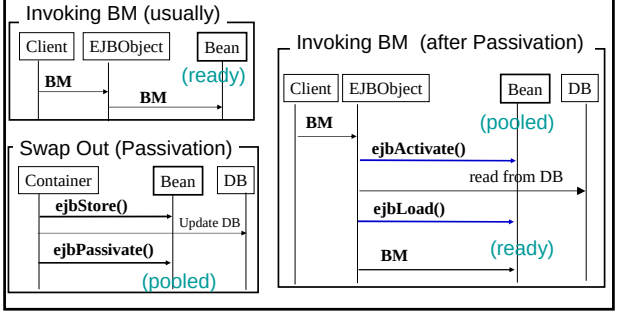
The Container invokes at least one ejbLoad between ejbActivate and the first business method in the instance

Entity Beans のライフサイクル



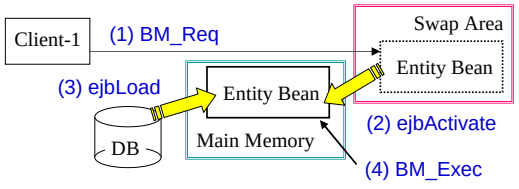
関連シナリオ

- BM 起動 と Passivation サービス



振舞い性質

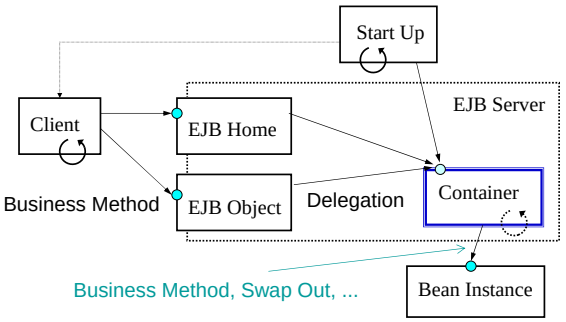
- スワップアウトされたEntity BeanにBM起動



[] (Swapped && BM_Req -> <>ejbActivate)

[] (ejbActivate && <>BM_Exec -> (!BM_Exec *U* ejbLoad))

Promela プロセス構成



検証した性質

⌘ドキュメント化されている時間的な制約

The Container invokes at least one ejbLoad between ejbActivate and the first business method in the instance

The Container invokes at least one ejbStore between the last business method on the instance and ejbPassivate

➡ True (satisfied)

⌘進行性 (leads to)

[] (M_Client -> <>M_Bean)

進行性 (1)

⌘create

[] (create -> <>ejbCreate) ➡ True

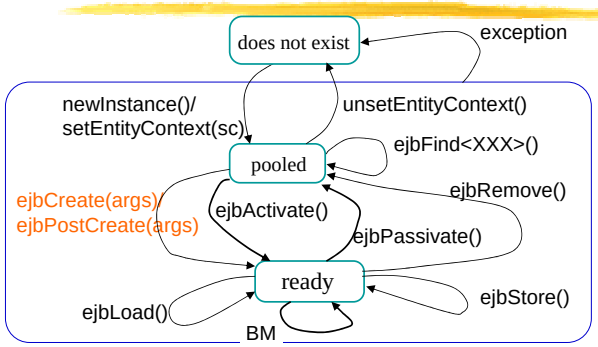
⌘ejbCreate is followed by ejbPostCreate

[] (create -> <>(ejbCreate && <>ejbPostCreate))

⇨ False

ejbCreateが例外を発生する場合がある

Entity Beans のライフサイクル



進行性 (2)

⌘Business Method

[] (BM_Req -> <>BM_Exec) ⇨ False

Beanが pooled 状態の場合、複数メソッドが起動される可能性がある (ejbActivate, ejbLoad)

さらに、ejbActivateが例外を発生する場合がある

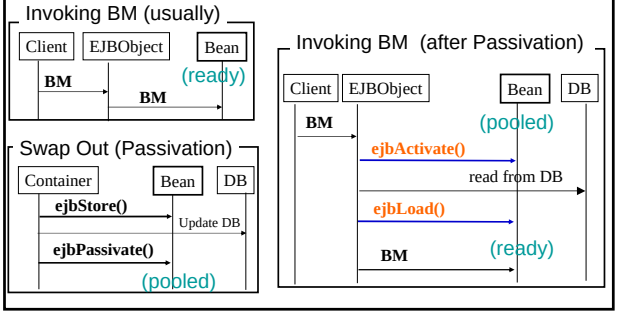
[] (BM_Req -> <>(BM_Exec || Exception)) ➡ True

! (<>(BM_Req && <>BM_Exec)) ⇨ False

トレースは予想通りのメソッド起動系列を示す

関連シナリオ

⌘ BM 起動 と Passivation サービス

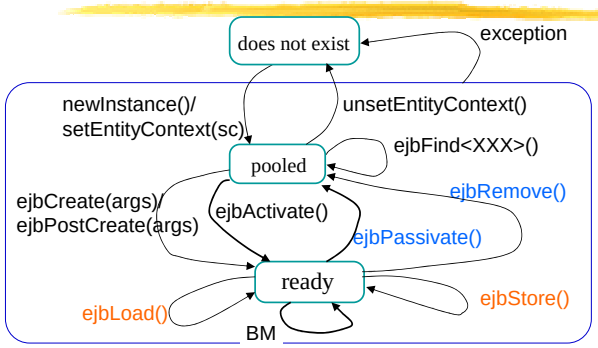


進行性 (3)

- remove
  - [] (remove -> <>ejbRemove) ⇨ False
    - ejbLoad と ejbStoreが**ライブロック**（**無限ループ**）
  - **弱い公平性の下で解析** ⇨ False
    - ejbRemove と ejbPassivate **が干渉**
    - Bean **は** ejbPassivate**によって** pooled **状態へ遷移**（**実行時サービスが非同期に実行**）

⇨ **EJB1.1 仕様書の不具合**

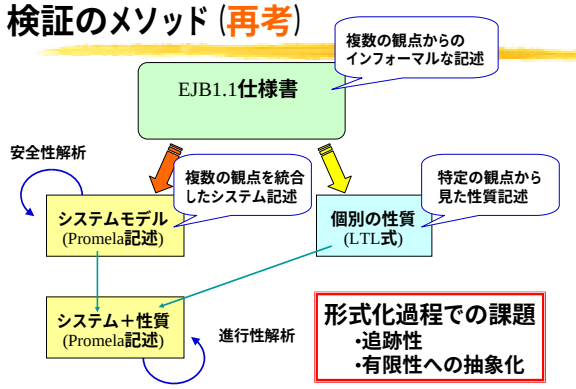
Entity Beans のライフサイクル



検証作業の実際

- **形式化フェーズ**
  - **目的：整合性のある Promela 仕様記述**
  - **曖昧さの除去、システム記述の洗練**
- **検証フェーズ**
  - **Promela 記述の“デバッグ”**
  - **簡単な性質の検証 (例： 例外発生を考慮しない場合)**
  - **不具合の可能性を発見 (例： remove と ejbPassivate)**
  - **“ライブロック”個所は構築の際の自由度に相当**
- **検証過程 ＝ 試行錯誤を必要とする繰り返し過程**
  - **システム記述の洗練、性質記述の洗練**

検証のメソッド（再考）



内容

- **モデル検査技術の概要**
- **事例の紹介**
- **おわりに**

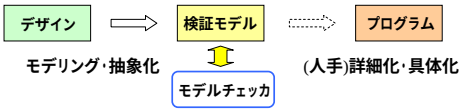
おわりに

- **形式検証技術の適用経験から**
  - **検証可能な規模の問題 → 計算機・検証アルゴリズムの進歩**
  - **検証対象のモデル化・抽象化が鍵 → 難しい**
  - **成り立つべき性質の網羅性・完全性 → これも難しい**

  モデリングが課題
- **高まる重要性**
  - **ソフトウエアの広がり → 従来にない本質的な難しさ**
    - **分散、並行・並列、通信特性、ノンストップ、オープンさ、動的改変、等**
    - **基盤ミドルウエア のみでは対応不可能**
  - **開発上流工程でのデザイン検証による信頼性向上**
- **今後**
  - **プログラム検証の重要性が増す？**

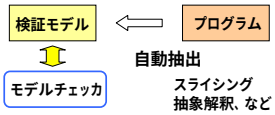
ソフトウェア検証：2つの見方

⌘ “Classic” Model-Checking ⇒ デザイン検証



モデリング・抽象化　(人手)詳細化・具体化

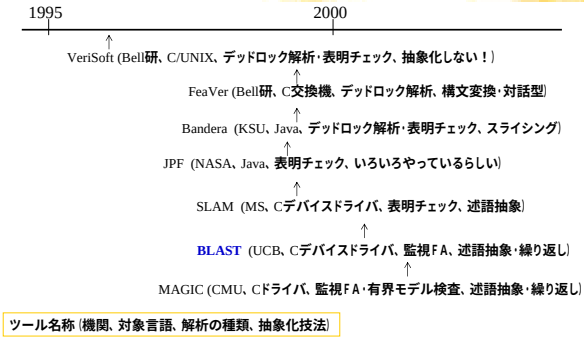
⌘ “Modern” Model-Checking ⇒ プログラム検証



自動抽出

スライシング
抽象解釈、など

プログラム検証ツール



情報源

⌘ 教科書
- Clarke, Grumberg, and Peled : Model Checking, The MIT Press 1999.
- Holzman : The SPIN Model Checker, Addison Wesley 2004.
- Magee and Kramer : Concurrency, Wiley 1999.

⌘ 代表的な国際学会
- IEEE/ACM ICSE
- ACM SIGSOFT FSE
- FME

⌘ フォーマルメソッド関連研究ホームページのURL
- http://www.afm.sbu.ac.uk/

おわりです