検証ツールの展開と
そのカリキュラムでの利用

小川瑞史（JAIST）
mizuhito@jaist.ac.jp

2007．9．6

形式検証への流れ

- 現実からの要請
  - 古典的なテスト手法だけでは複雑なシステムに対応できなくなりつつある。形式検証による補完が必要。
  - システムが正しいだけでなく、正しさの証拠が社会的要請となりつつある（セキュリティ, EAL7 ISO/IEC15408）
  - 人間は間違える！

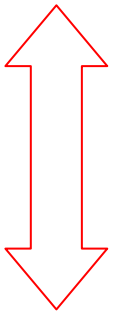
- 形式手法が可能とするサポート
  - 近年の検証ツール・ライブラリの実用性向上（インタフェース・実装の改良）
  - ハードウェアの進化 (2GB以上のメモリ)
  - 自分の誤りに気づくことが最大の利点！

形式検証とは

- システムやプログラム、ならびに満たすべき性質の双方を形式的に記述(＝計算機でチェック可能な形で)
  - 形式的記述の背景は数理論理(推論の規則化)

- 論理の表現力と自動化・効率化のトレードオフ

表現力大



自動化
効率的

  - 帰納法:定理証明系・代数仕様記述(自動化困難)
  - 一階述語論理:充足性検査(実用的には自動化可)
  - 様相論理:モデル検査(自動化)
  - 命題論理:充足性検査(自動化)

本日の目的:検証ツールのユーザの視点からの(粗い)地図

形式検証の成功事例

- CPUデザインでは実用的に用いられている
  - Intel, IBM, AMD, ….
  - 定理証明系＋モデル検査は一つの標準的手法 (Intel は独自システムを関数型言語flで実装)
  - Intel verification labo.: USA/Israel, 30-40 Ph.D's.

- 次の研究のターゲットはソフトウェアの検証？！
  - 午後の事例紹介

- 数学的証明
  - Jordan の曲線定理（信州大ほか）

検証における大いなる誤解

- 検証に通れば正しい
  - 間違いの発見は強力だが、正しさを示したとは言い切れない。
  - モデル検査：間違っていれば反例を提示、正しいときは “ok”のみ。
  - 定理証明系：証明はできた。でも何を証明したか？
    - 簡単に証明できた⇒意図とは異なることを証明
      ⇒インスタンスのないことを証明

- 正しさとはコンセンサス
  - 数学的証明は厳密ではない（たとえば補集合の概念）。証明は数学者間のコンセンサスに過ぎない。
  - 形式化が意図通りかどうかは最後は人間が判断。
  - 要求される正しさはコストとのトレードオフ。

定理証明系(帰納法を含む論理)

- 高階論理 現実主義的 ⟷ 原理主義的
  - *PVS, HOL, Isabelle/HOL, Misar, NuPrl, Coq, Agda, Twelf*
  - プログラムの意味・性質は論理体系の中に記述
  - 帰納法は体系内で記述可能
  - 型(α set)を導入することで Russel の逆理(「自分自身を含まないすべての集合からなる集合」)を回避。

- 一階述語論理＋帰納法(代数的仕様記述)
  - *Café/OBJ, Maude, Larch.*
  - プログラムの意味は(無限)状態遷移として記述
  - プログラムの性質は到達可能性などで記述

難しさは証明力(形式的定義能力、証明構成力)

自動証明ツール(帰納法を含まない論理)

- 命題論理 (CNF)　≒ビット演算器の有界実行
  - NP-完全のはず、しかし実際は高速
  - SAT-solver(充足性検査器), e.g., *miniSAT, zchaff*
  - 応用例: Alloy, NuSMV (bounded MC), TTT (停止性検証)
- 一階述語論理 + 拡張 (array, arithmetic)
  - 決定不能のはず、しかし実際は(半)自動
  - *Simplify, CVC Lite, E, Vampire*
  - 応用例: ESC/Java2, Caduceus (注釈付プログラムの検証)
- 様相論理 (LTL, CTL, safety)
  - モデル検査
  - *SPIN, (Nu)SMV, SLAM (Moped), WPDS+, Maude*
  - 応用例: Bandera/Boger, Java PathFinder (プログラム解析)

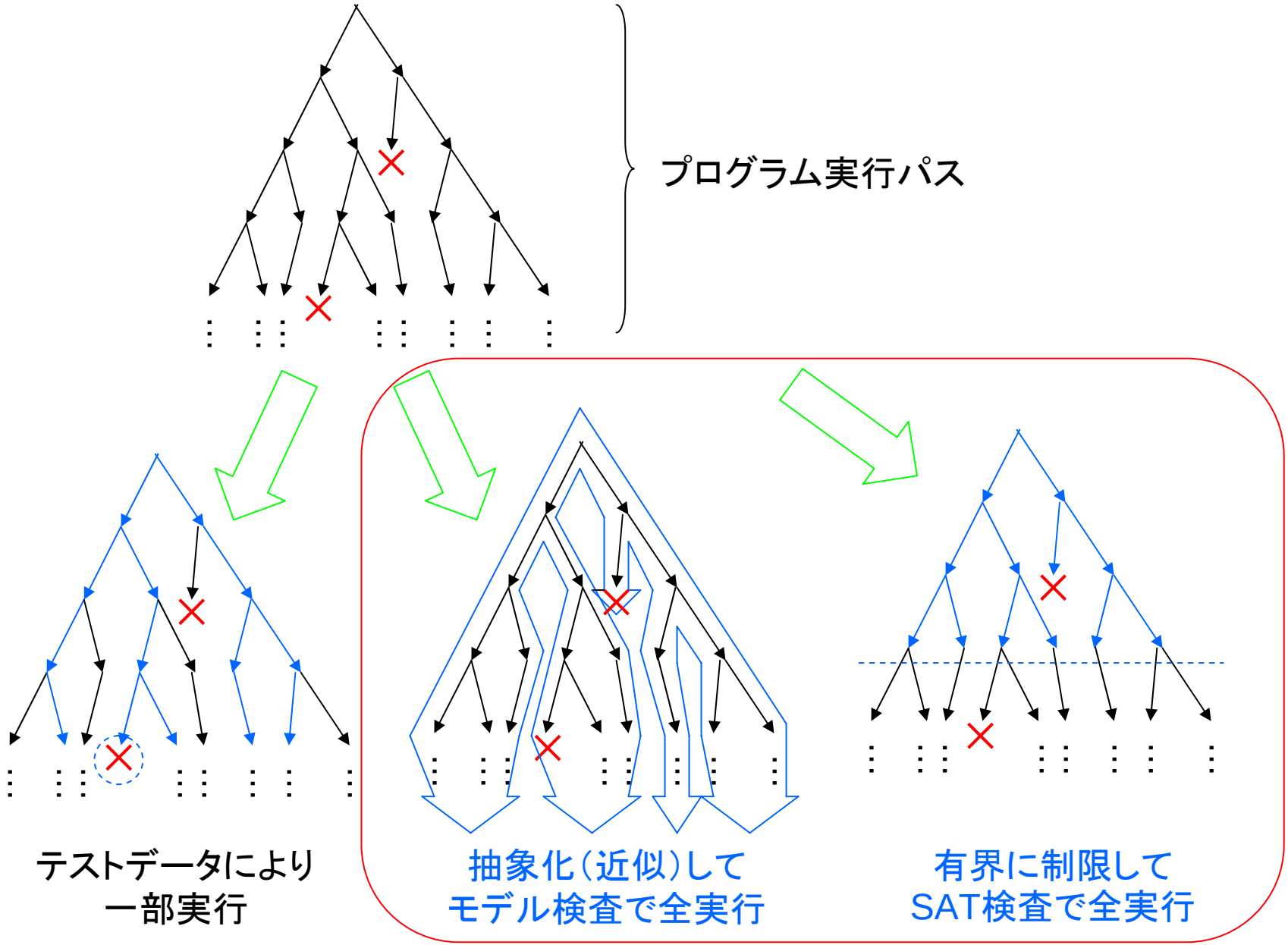
難しさはモデル化(取捨選択)のデザインセンス

論理の表現力と問題の関係

- プログラム解析でいえば …

	命題論理・様相論理	一階述語論理	帰納法
手続内	古典的データフロー解析 （不要コード解析など）	アレイ境界検査 （注釈有）	アレイ境界検査 （注釈無）
⇩			
手続間	スタック検査 関連コード解析		

プログラム検証における自動検証の考え方



SAT-solver

SAT（充足可能性問題）とは?

CNFの命題論理式を与えたとき
$(\pm x_1 \vee \ldots \vee \pm x_n) \wedge (\pm y_1 \vee \ldots \vee \pm y_m) \wedge \ldots$
全体を真とする論理変数への真偽代入を発見せよ。

$\omega_1 = x_1 \vee x_2 \vee x_3$

$\omega_2 = \neg x_1 \vee x_2$

$\omega_3 = \neg x_2 \vee x_3$

$\omega_4 = \neg x_1 \vee \neg x_2 \vee \neg x_3$

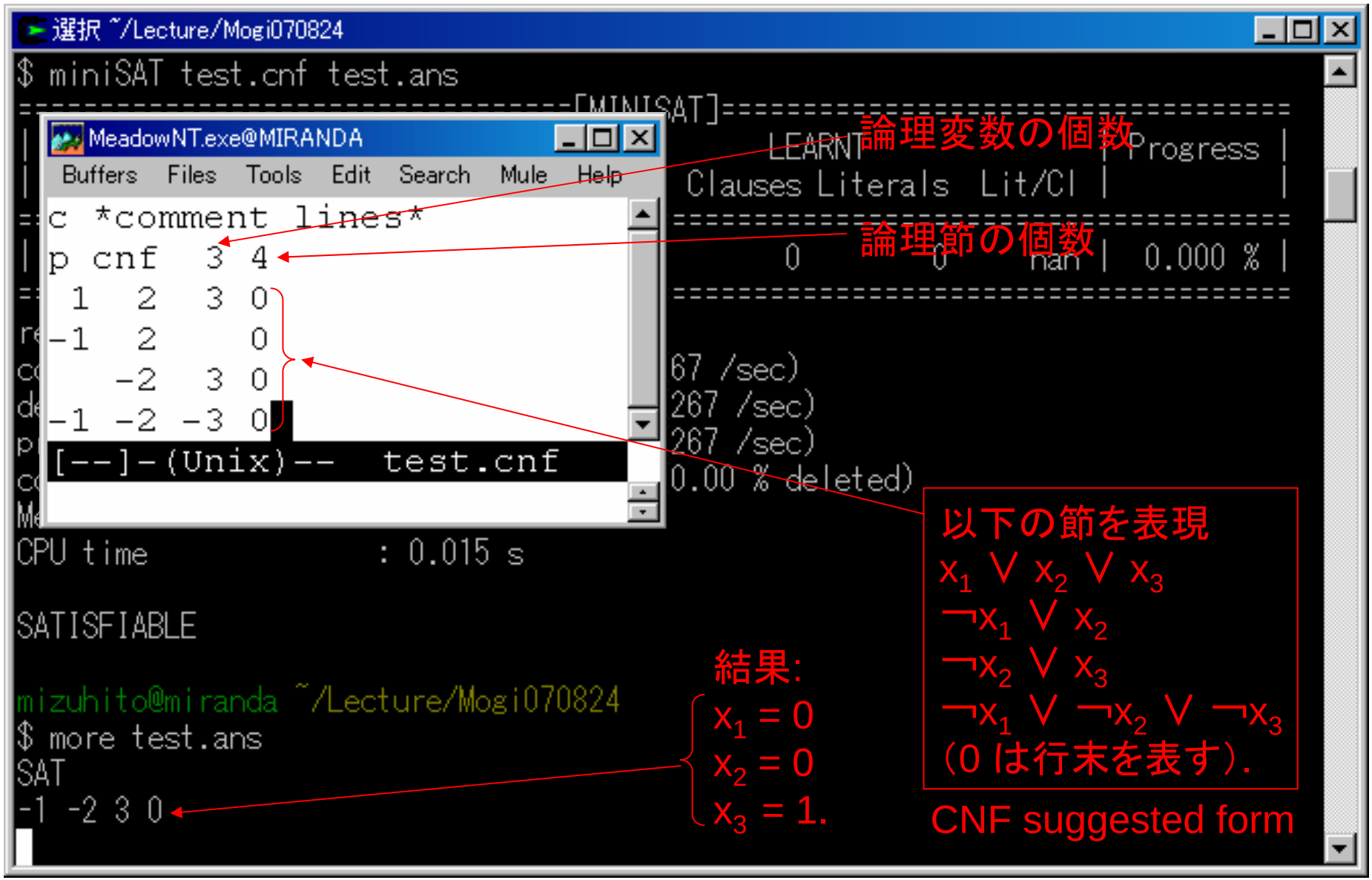
$A = \{x_1=0, x_2=0, x_3=1\}$

$\omega_1, \omega_2, \omega_3, \omega_4$を真とする代入

原理：総当り（しらみつぶし）の探索

3SAT（節の長さが3）は*NP*-完全, しかし実際には（たいがい）
高速に解ける（現実の問題は意外と扱い易い！）

miniSATの実行例

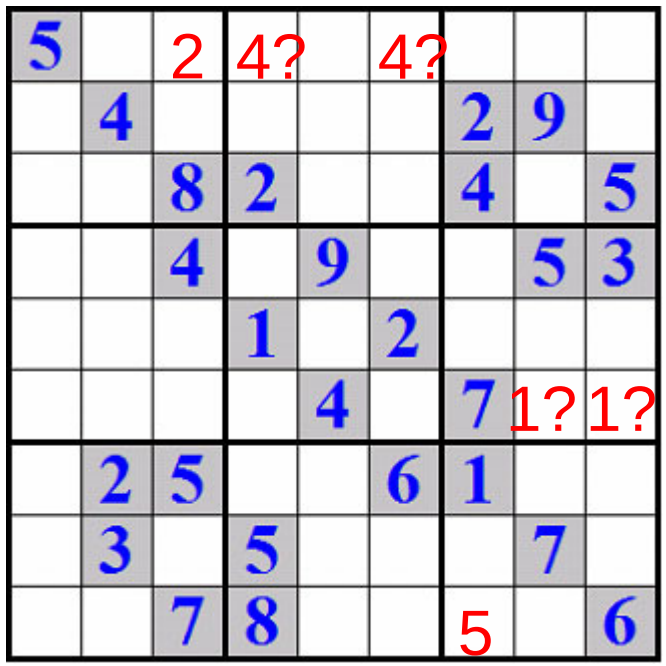


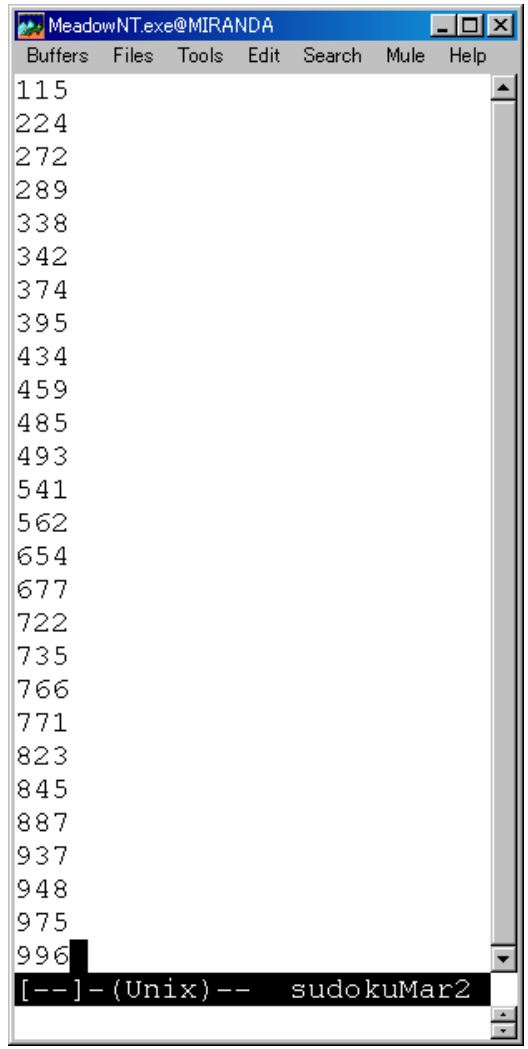
SATでの記述例：数独パズル（講義資料より）

各マスに以下の条件を満たすように *1*～*9*の数字を入れる。

- 各行に
- 各列に
- 各 *3**3 小行列に

1～*9* はそれぞれちょうど一度現れる。





SUDOKU as a SAT problem

Lynce, I., Ouakline, J., AI Math 2006

- x_{ijk} は k が (i, j) マス（i 行j 列）にあることを表す。

- 各(i, j) マスは1から9のいずれかが一つ入る。
  $(x_{ij1} \vee x_{ij2} \vee \ldots \vee x_{ij9}) \wedge (x_{ij1} \rightarrow \neg x_{ij2})$ (i.e., $\neg x_{ij1} \vee \neg x_{ij2}$), …

- 各 i 行には1から9がちょうど一度現れる。
  $(x_{i1k} \vee x_{i2k} \vee \ldots \vee x_{i9k}) \wedge (x_{i1k} \rightarrow \neg x_{i2k})$ (i.e., $\neg x_{i1k} \vee \neg x_{i2k}$), …

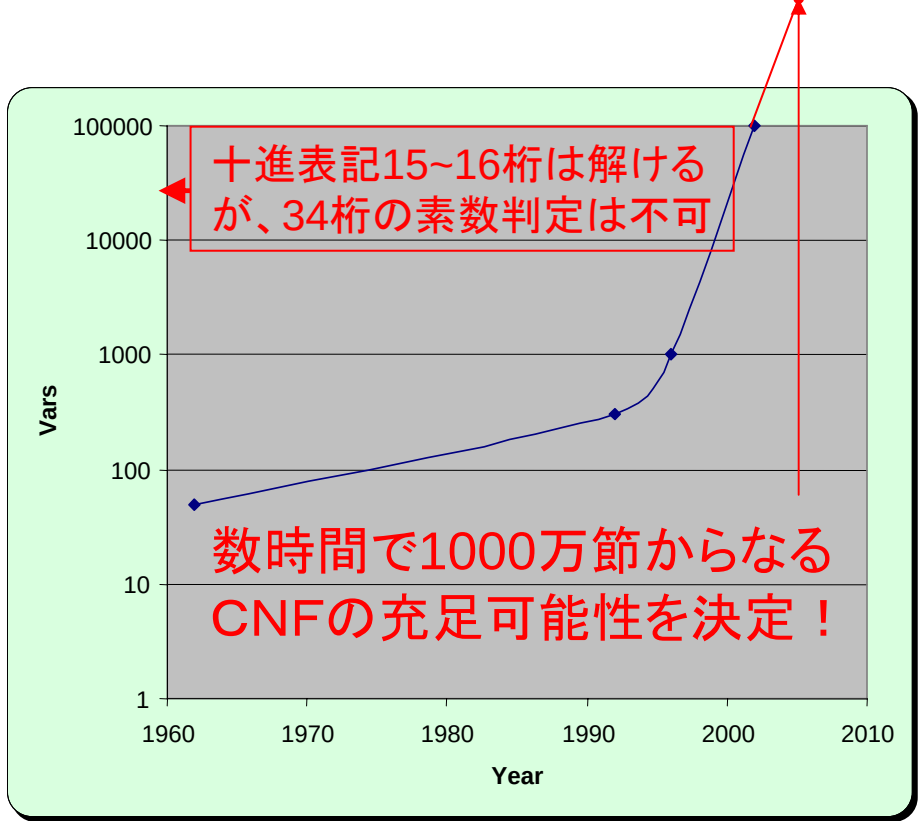
- 各 j 行には1から9がちょうど一度現れる。
  $(x_{1jk} \vee x_{2jk} \vee \ldots \vee x_{9jk}) \wedge (x_{1jk} \rightarrow \neg x_{2jk})$ (i.e., $\neg x_{1jk} \vee \neg x_{2jk}$), …

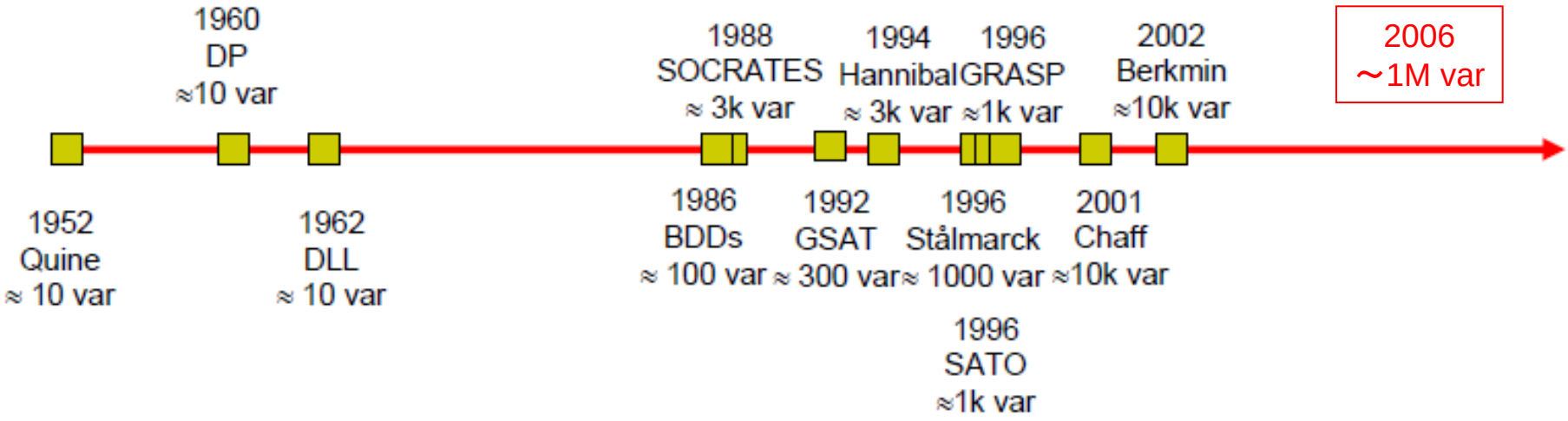
- 各 3×3部分行列には1から9がちょうど一度現れる。
  $(x_{11k} \vee x_{12k} \vee \ldots x_{33k}) \wedge (x_{11k} \rightarrow \neg x_{12k})$ (i.e., $\neg x_{11k} \vee \neg x_{12k}$), …
  ただし$1 \leqq i, j, k \leqq 9$

729変数、約8200節、10ms程度で発見

自動証明ツールの競技会

- SAT competition
  www.satcompetition.org/
- SMT-COMP (2005~):
  SAT Modulo Theories
  アレイ、整数・実数演算
  非解釈関数記号の等式





- CASC (CADE ATP):一階述語論理自動証明 (1996~)

モデル検査

モデル検査が有効な場合

- 間違いがみつかった場合に有効
  - “verified” となった場合は bug injection をして、そのbug が正しく検出されるか、また芋蔓式にbug が増えないか、などをチェック。
  - システムをモデルを作る際に間違いに気づく

- 成功パターン（ケーススタディ）:
  - Bugのあるライブラリや関連するコードを選択
  - （人手で）関連のあるコードのみ（スライス）を抽出
  - この部分を（人手で）モデル化してモデル検査

モデル検査の原理

- モデルの生成する実行列が仕様で許されている実行列に含まれる
  - $L(M) \subseteq L(S) \Leftrightarrow L(M) \cap L(S)^C = \phi$
  - これを形式言語の問題として解く

- 最近のモデル検査の実装は SAT solver を用いる（bounded model checking)
  - 理論的には計算量大。でもBug がない場合は時間がかかってもよい。ある場合は早く見つけてほしい。
  - BDD等に比べ、メモリ使用量が漸増的。
  - NuSMV は zchaff を利用

モデル検査における選択

- システムからモデルへの変換を
  - 人間が行う→柔軟な設計情報なども用いたモデル化
  - 自動的に行う→インテリジェントコンパイラ

    プログラム解析＝抽象化＋モデル検査
- モデルを
  - 最初に全部生成する→モデル自体を記述
  - On-the-fly に生成する→モデルの生成規則を記述

    Maude
- 状態数が
  - 有限状態（有限モデル検査）→手続き内解析
  - 無限状態（プッシュダウンモデル検査）→手続き間解析

    Moped, Weighted PDS

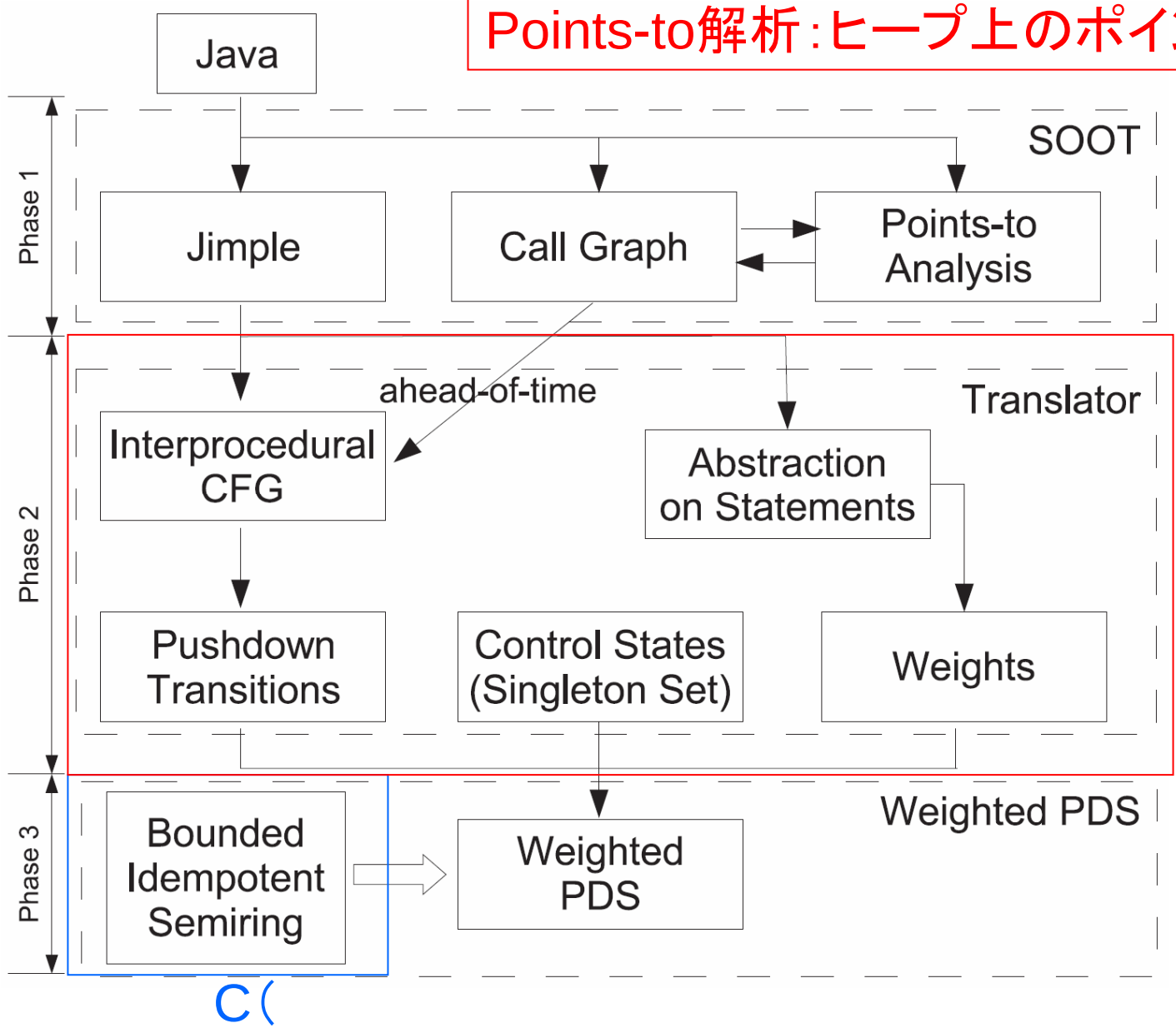
SPINの次にくるモデル検査系？

プログラム解析＝モデル検査＋抽象化

- 既存ツールの利用によるラピッドプロトタイピング
  - Java の場合の前処理：Java bytocode変換や SOOT (Jimple コンパイラ)
  - 後処理の解析エンジンとしてモデル検査系
    - JavaPathFinder（NASA)
    - Bandera（カンザス州率大学）
    - SMV (M2が研究開始から3ヵ月）,
    - Weighted PDS（D2 が研究開始から約1年）

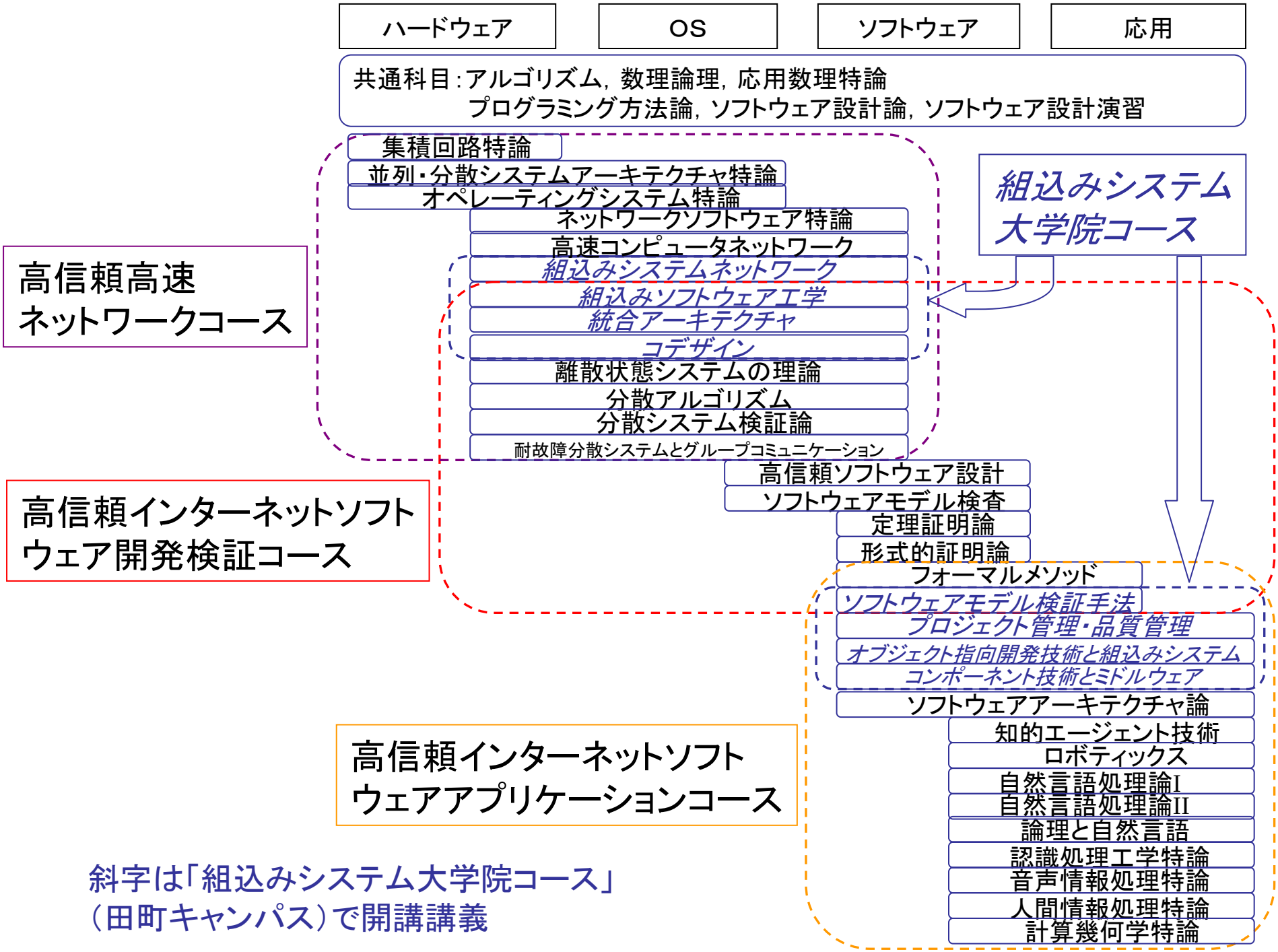
既存ツールを用いたJavaプログラム解析の実装例

Points-to解析：ヒープ上のポインタ解析



Java
(2000行)

カリキュラムでの取り組み


ハードウェア
OS
ソフトウェア
応用
共通科目：アルゴリズム，数理論理，応用数理特論
プログラミング方法論，ソフトウェア設計論，ソフトウェア設計演習
集積回路特論
並列・分散システムアーキテクチャ特論
オペレーティングシステム特論
ネットワークソフトウェア特論
高速コンピュータネットワーク
組込みシステムネットワーク
組込みソフトウェア工学
統合アーキテクチャ
コデザイン
離散状態システムの理論
分散アルゴリズム
分散システム検証論
耐故障分散システムとグループコミュニケーション
組込みシステム
大学院コース
高信頼高速
ネットワークコース
高信頼ソフトウェア設計
ソフトウェアモデル検査
定理証明論
形式的証明論
フォーマルメソッド
ソフトウェアモデル検証手法
プロジェクト管理・品質管理
オブジェクト指向開発技術と組込みシステム
コンポーネント技術とミドルウェア
高信頼インターネットソフト
ウェア開発検証コース
ソフトウェアアーキテクチャ論
知的エージェント技術
ロボティックス
自然言語処理論I
自然言語処理論II
論理と自然言語
認識処理工学特論
音声情報処理特論
人間情報処理特論
計算幾何学特論
高信頼インターネットソフト
ウェアアプリケーションコース
斜字は「組込みシステム大学院コース」
（田町キャンパス）で開講講義

現在までに導入した形式手法に関連する講義

- H15年度新規開講（以降隔年開講）
  - I636 分散システム検証論（二木）
  - I637 高信頼ソフトウェア設計（片山）
  - I432 離散状態システムの理論（平石）
- H16年度新規開講（以降隔年開講）
  - I639 分散アルゴリズム（Defago）
  - I640 ソフトウェアモデル検査（小川）
  - I641 組込みソフトウェア工学（岸）
- H17年度新規開講（以降隔年開講）
  - I642 定理証明システム論（小川）
  - I643 形式的証明論（Vestergaard）
  - I644 耐故障分散システムとグループコミュニケーション(Defago)

H14~18年度
人材養成プログラム
「高信頼インターネット
ソフトウェア開発検証」
H16~20年度
21世紀COEプログラム
「検証進化可能電子社会」

来年度からの統廃合

- 修士課程への拡張
  - I4XX「ソフトウェア検証論」(青木):検証ツール(モデル検査系 SPIN,定理証明系 HOL)を中心に。
  - I211「数理論理学」(小川):一部にSAT solver(命題論理・一階述語論理)を取り入れる
  - I639,I644 は内容を改訂し、I4XX「分散システム」とI4XX「性能評価」へ。
- 博士課程向けの講義内容の整理(最新の研究成果の紹介)
  - I643「形式証明論」は定理証明系
  - I636「分散システム検証論」は代数的仕様記述
  - I640「ソフトウェアモデル検査」はモデル検査と SAT solver
- 重複した内容の統廃合
  - I637(廃止),L642の一部→I4XX「ソフトウェア検証論」
  - I642,I643 は統合してI643 に一本化

注:600番台は博士課程(留学生が多い)向けで英語で行う