モデル検査技術入門

青木利晃
北陸先端科学技術大学院大学
安心電子社会研究センター

ソフトウェアの信頼性

- 社会システムの様々な場所にソフトウェアが組み込まれている。
  - バンキングシステム、航空管制、携帯電話、自動車、etc.
- 誤りを含むものが少なくない。
  - 社会の混乱、莫大な損害、人的損害を引き起こす。

☛ 正しいソフトウェアを作ることが重要。

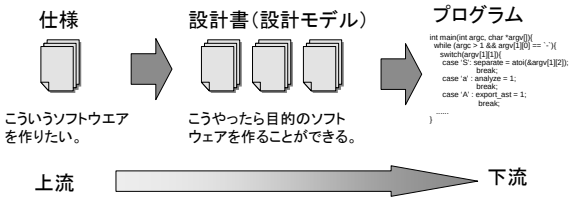
- 現在はテスト手法中心の開発。
  - それでも誤りが取り除けないなら、パラダイムシフトが必要。
- 形式手法/検証
  - 数学的、論理的基盤に基づいたソフトウェア開発。

テストと検証

- テスト(Test)
  - 限られた実行列（テストケース）で誤りが無いことを保障する。
    - 実行列は無限にあるので、重要な部分のみをチェックする。
    - カバレッジ：テストケースが全体のどれぐらいの部分をカバーしたか？
  - テストされていない部分が正しいかどうかは保障されない。
- 検証(Verification)
  - 検証対象が正しいことを保障する。
  - 数学的、論理的基盤に基づいて正しさを証明する。

検証対象

- ソフトウェア開発ではプログラム以外にも様々なモノを作る。



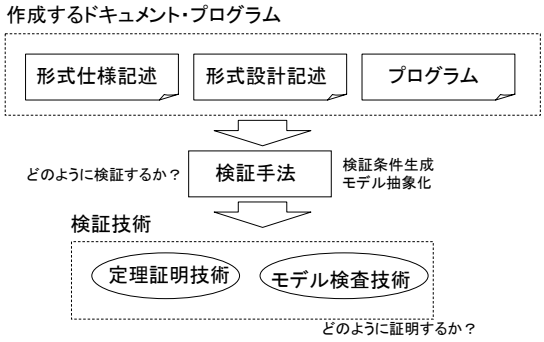
ソフトウェアの検証

- プログラムのチェックだけでは効果が薄い。
  - 仕様の誤り、設計の誤りをプログラムで見つけるのは困難。
- 仕様、設計を厳密に書いて、チェックする。
- 厳密に仕様と設計を書く。
  - 仕様、設計をプログラムのように実行可能な言語で書く。
- 仕様・設計記述言語
  - UML:クラス図, ステートチャート図, etc...
  - Z, VDM, B, etc.

ソフトウェアの検証

- 構築した仕様・設計書・プログラムを検証する。
- 検証手法：
  - モデル検査手法：
    - 有限状態モデル。
    - 状態を網羅的に探索し、性質が成立するかどうか自動的に検査する。
    - Spin, NuSMV, LTSA, UPPAAL, etc.
  - 定理証明手法：
    - 述語論理など。無限の値や状態。
    - 推論規則を用いて、性質を導出・証明。
    - 原理的には決定不能問題→対話的推論。
      - 様々な部分問題の自動化への挑戦。
    - HOL, Isabelle, Coq, PVS, etc.

ソフトウェアの検証



形式検証の現状

- うまく適用ができれば絶大な効果があるが、万能ではない。
  - 決定可能問題と決定不能問題。
  - 理論的に可能、実際に可能。
- 適用事例。
  - Edmund M.Clarke and Jeannette M.Wing: Formal Methods: State of the Art and Future Directions,ACM Computing Surveys, 1996.

事例

- Edmund M.Clarke and Jeannette M.Wing
  - Formal Methods: State of the Art and Future Directions,
  - ACM Computing Surveys, 1996.
- CISC (Customer Information Control System)
  - Oxford University and IBM Hursley Lab, 1980年代.
  - Z記法。
  - エラー数の削減、開発初期段階でのエラー検出。
  - 9%の開発コスト削減。

事例

- CDIS: 英国航空管制システム（CCF Display Information System）
  - Praxis, 1992(delivered)
  - 耐故障性分散システム（約100台）
  - VDM＋CCS
  - 非形式的 vs 形式的
    - 生産性：同じか、良いくらい。
    - 品質：飛躍的に向上。バグの割合0.75/1000LOC。他の航空管制システムと比べて2～10倍。
- TCAS (Traffic Collision Avoidance System) II
  - Safety-Critical Systems Research Group, 1990年代
  - RSML

事例

- IEEE Futurebus+
  - Cache coherence protocol.
  - CMU, 1992.
  - SMV
  - プロトコル設計段階で見逃された多くのエラーを発見
    - 1988年からプロトコル開発がはじまったが、それまでは非形式的な手法が使われていた。
- ISDN/ISUP
  - ITU ISDN/ISUP Protocolの検証, AT&T, 1989-1992.
    - 145の要求を時相論理で記述（5人の検証エンジニア）
    - 独自のモデル検査ツールによる証明
      - SDLソースコード：7500行
      - 112のエラー検出（設計段階）、55%のオリジナル設計要求が論理的におかしいことを示した。

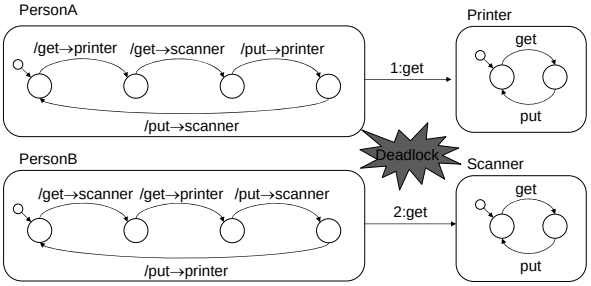
モデル検査技術

- 産業界では、最近、モデル検査に注目が集まっている。
  - 比較的簡単。
  - 自動的に結果がでる。
- すべてを取り扱えないが、有効には働きそう。
  - 組込みソフトウェアでは状態変化・並行性の取り扱いが重要。
  - マルチタスクの振る舞い解析。
    - デッドロック、競合状態の検出。微妙なタイミングの検証。
  - 振る舞い解析のエンジン。
    - ステートチャート図、コラボレーション図などの検証。

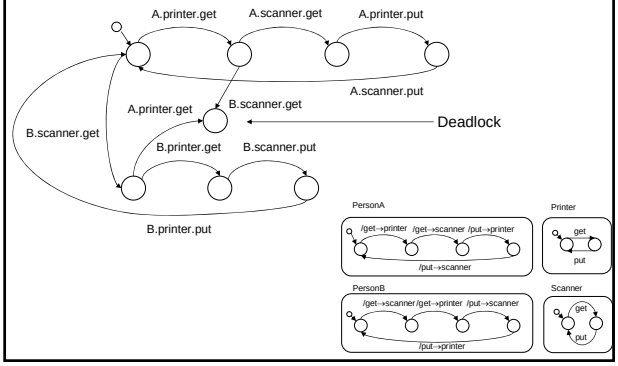
モデル検査技術

簡単な例

- プリンタとユーザ。

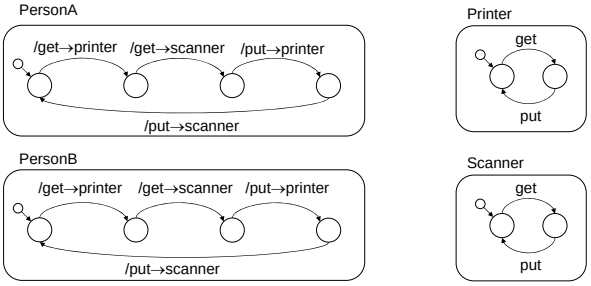


簡単な例



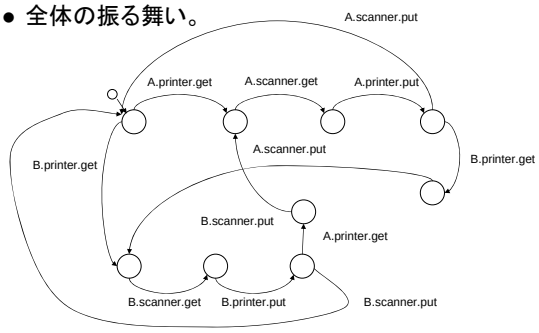
簡単な例

- Deadlockしない例。



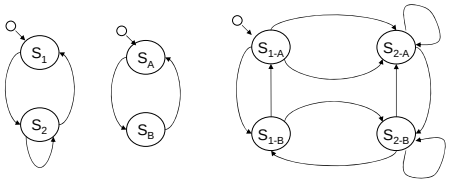
簡単な例

- 全体の振る舞い。



状態の探索

- 並行動作する状態遷移モデルは、それらの可能な遷移を探索することにより、すべての動作の可能性を調べることができる。
- 可能な遷移=並行状態遷移モデルの直積上で表現できる。
  - 状態の数は指数関数的に増える。
    - でも、有限。
    - 効率的なデータ構造：BDD (Binary Decision Diagram)

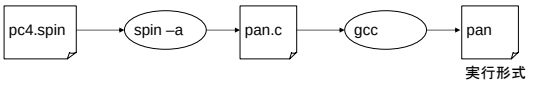


モデル検査ツールSPIN

- SPIN
  - AT&T Bell研が開発したモデル検査ツール。
  - 1980年代に開発、現在も継続的にバージョンアップ。
  - WIndowsやUnix上で動作。フリーソフトウェア。
    - http://spinroot.com/
  - 2002年 ACM System Software Award受賞。
- チャネル通信オートマトンのためのモデル検査ツール。
  - 並行動作する有限オートマトン。
  - オートマトン同士は、チャネルを使って通信をする。
- Promela:チャネル通信オートマトンを記述するための言語。
  - SPINはPromelaによる記述を入力として、網羅的に状態を探索し、その性質を検査する。

モデル検査ツールSpin

- SPINによる実行。
  - Promelaによる記述はシミュレーション実行できる。
    - 1回の実行で1つの実行系列のみ。
- SPINにおけるモデル検査。
  - pan(Protocol ANalyzer)を自動生成。
    - 網羅的に探索するプログラム。
    - spin –a pc4.spin
  - コンパイルして実行。
    - gcc pan.c –o pan
    - ./pan



モデル検査ツール

- 検査結果。

```
pan: invalid end state (at depth 8)
pan: wrote pan_in.trail
(Spin Version 4.0.7 -- 1 August 2003)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             - (not selected)
        assertion violations    - (disabled by -A flag)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 48 byte, depth reached 12, errors: 1
     10 states, stored
      1 states, matched
     11 transitions (= stored+matched)
      0 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)
```

反例

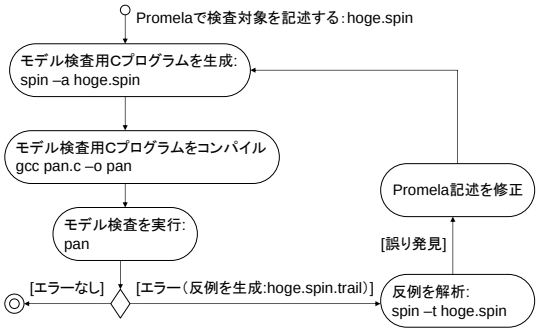
- 正常で無い性質が検出されると、それに導く例を出力する(反例-Counter Example)。

```
 1:     proc  0 (:init:) line  34 "pan_in" (state 1)  [(run Printer(P_port))]
 2:     proc  0 (:init:) line  35 "pan_in" (state 2)  [(run Scanner(S_port))]
 3:     proc  0 (:init:) line  36 "pan_in" (state 3)  [(run PersonA(P_port,S_port))]
 4:     proc  0 (:init:) line  37 "pan_in" (state 4)  [(run PersonB(P_port,S_port))]
 5:     proc  4 (PersonB) line  12 "pan_in" (state -)  [values: 2!get]
 5:     proc  4 (PersonB) line  12 "pan_in" (state 1)  [scanner!get]
 6:     proc  2 (Scanner) line  26 "pan_in" (state -)  [values: 2?get]
 6:     proc  2 (Scanner) line  26 "pan_in" (state 1)  [port?get]
 7:     proc  3 (PersonA) line   4 "pan_in" (state -)  [values: 1!get]
 7:     proc  3 (PersonA) line   4 "pan_in" (state 1)  [printer!get]
 8:     proc  1 (Printer) line  20 "pan_in" (state -)  [values: 1?get]
 8:     proc  1 (Printer) line  20 "pan_in" (state 1)  [port?get]
spin: trail ends after 9 steps
#processes: 5
 9:     proc  4 (PersonB) line  12 "pan_in" (state 2)
 9:     proc  3 (PersonA) line   4 "pan_in" (state 2)
 9:     proc  2 (Scanner) line  26 "pan_in" (state 2)
 9:     proc  1 (Printer) line  20 "pan_in" (state 2)
 9:     proc  0 (:init:) line  38 "pan_in" (state 5)
5 processes created
Exit-Status 0
```

PersonB: scanner!get
Scanner: port?get
PersonA: printer!get
Printer: port?get

終了

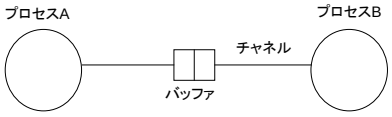
モデル検査のプロセス



モデル検査ツールSpinの概要

Promela

- Promelaで書かれた振る舞いを網羅的にチェックする。
  - Promela:チャネル通信オートマトンを記述するための言語。
- チャネル通信オートマトン。
  - 並行に動作するプロセス（オートマトン）
  - プロセス間でメッセージ通信をするためのチャネル。



Promela

- Promelaでの振舞い記述。
  - データ型とその演算。
  - 繰り返し、条件分岐。
  - 非決定性
    - 複数の振る舞いのうち、どれも実行される可能性がある。
    - 複数の振る舞いがランダムに選択される。

値について

- Promelaでは整数の変数を使うことができる。
  - bit: 0,1
  - byte: 0〜255
  - short: -2^{15}〜$2^{15}-1$
  - int: -2^{31}〜$2^{31}-1$
- 配列
  - 「bit a[5]」、「bit a[5] = 0」、「byte hoge[2]」, ...
  - a[3] = 0, hoge[1] = 10, ...
- レコード型
  - 「typedef Point{ int x; int y}」
  - Point p; p.x = 0; ....
- 演算子
  - +, -, *, / : 足し算、引き算、掛け算、割り算
  - % : 余り
  - < , > , <= , >= : 比較
  - &&, ||, !, ==, !=: 論理積、論理和、否定、等しい、等しくない、

様々な性質の検証

Spinによりチェックできること。

- 表明。
  - assert(条件式)
- 到達性
  - 終了してはいけないところで終了するかどうか。
  - デッドロックなど。
- 進行性
  - 頻繁に訪れる：いつかは訪れる。
- 性質オートマトン
  - 望ましい性質、もしくは、望ましくない性質を状態遷移モデルで記述する。
- LTL式
  - 論理式に時間を表現する演算子（様相演算子）を追加。

表明

- Promelaによる振る舞い記述の中に表明(assertion)が書ける。
  - assert(条件式)
    - その時点で条件式が成立してほしい。
    - 成立しないときはエラーを出力。
- 表明が破られると、破られる状況へ導く実行列を反例として出力する。

到達性

- プロセスが途中で停止する場合には、停止することを指定しなければならない。
  - 停止することがいつも悪いわけではない。
- なにも指定しないと、途中で停止する場合には、エラーを出力。
  - デッドロックの検出。
  - デッドロックにいたる実行列を出力する。

進行性検査

- 頻繁に訪れるポイントにprogressラベルをつける。
  - 頻繁に訪れる：いつかは訪れる。
    - 無限の実行列において、無限回訪れる。
    - その場所を訪れないような無限の実行列が無い。
- 進行性の反例：Progressラベルを通らないようなサイクルを出力。
  - そのサイクルをぐるぐる回れば、progressラベルを通らないような無限の実行列を作ることができる。

時相論理

- 述語論理や命題論理では時間について（直接的に）言及できない。
- 線形時相論理(Linear Temporal Logic)
  - 時間の進み方が一直線になる世界での性質の定義。
  - cf) CTL(Computational Tree Logic)
    - 時間の進み方が枝分かれになる世界での性質の定義。
- 通常の論理式に時間を表現する演算子 ◊,□,U, Xなどをつけて修飾する。
  - 例) □(P⇒◊Q∧R), □◊¬(P∨X Q), etc.

様相演算子

LTLの直感的意味

sometime: ◊p

always: □p

next time: X p

until: p U q

マルチタスクソフトウェアの検証への応用例

マルチタスクソフトウェア

- RTOSを用いたソフトウェア開発。
  - マルチタスクソフトウェア。
    - タスクによる並行処理の直接的な記述。
  - 動作やタイミングの解析が必要。
    - デッドロック、飢餓状態、競合状態、etc.
- モデル検査ツールSpin。
  - 動作やタイミングの検証。
  - タスクの概念に近い非同期プロセスを取り扱うことができる。
  - Promela
    - 優先度、セマフォ、Sleep/Wakeupなどの振る舞い制御に対応する概念が無い。

マルチタスクソフトウェア

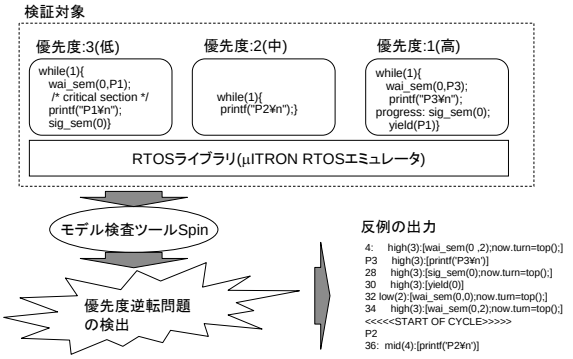
- 振る舞い制御（優先度、セマフォ、Sleep/Wakeup）の取り扱い。
  - 振る舞い制御を無視して、並行に動作するものとして検証。

  →実装されると解消されるような実行列に対してエラーを検出。
  - 振る舞い制御をPromelaで実現する。

ライブラリ化

- Sleep/Wakeup, セマフォなどの実現法は再利用できる。
  - 実現している部分をコピー＆ペーストすればよい。
- 優先度に関しては、単純化すれば簡単に実現できる。
  - 優先度による複雑な処理を含む場合は、単純化が難しい。
  - 以上の操作を組み合わせると、実現法が複雑になる場合もある。
- RTOSライブラリ：RTOSで行われている処理をエミュレートして、タスクの実行を制御する。
  - μITRON用のライブラリを実装済み。

RTOSライブラリによる検証



現状と課題

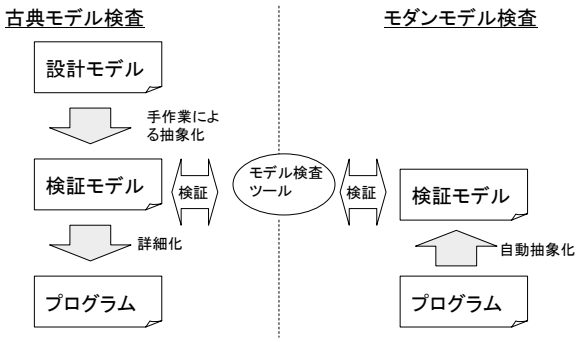
形式検証の現状

- うまく適用ができれば絶大な効果があるが、万能ではない。
  - 決定可能問題と決定不能問題。
  - 理論的に可能、実際に可能。
- 適用事例。
  - 欧米の大規模システム、標準化、航空・宇宙など。
    - 人的、時間的、金銭的コストをかけることができる。
  - 民需の組込みソフトウェアへの適用は実験的。
    - コストの問題。

モデル検査の適用

- すべてを取り扱えないが、有効には働きそう。
  - 組込みソフトウェアでは状態変化・並行性の取り扱いが重要。
  - マルチタスクの振る舞い解析。
    - デッドロック、競合状態の検出。微妙なタイミングの検証。
  - などなど。
- 問題なのは、記述能力。状態爆発問題。
- 2つのアプローチ
  - 古典モデル検査
  - モダンモデル検査

古典 vs. モダンモデル検査



古典モデル検査

- 解きたい問題を、モデル検査ツールで扱えるように書く。
  - 手作業。
  - どのような問題を解きたいか？
  - その問題は、どのように書けば、網羅的探索で検出できそうか？
- 状態爆発がおきないように適切に抽象化する。
  - 解きたい問題がひっかかるように抽象化する。

モダンモデル検査

- プログラムからのモデルの抽出とモデル検査。
  - 自動抽象化。
    - データマッピング、述語抽象化
    - 整数や小数に関する推論が必要。
      - 定理証明システムを併用。
      - 決定可能問題に限定。
- ツール。
  - SLAM（マイクロソフト、C言語）
  - BLAST（Barkley Univ.、C言語）
  - Bandera(Kansas City Univ.、Java）

モデル検査適用の課題

- 「モデル検査技術の研究」と「モデル検査の適用技術の研究」
- モデル検査の適用技術。
  - 実開発と検証技術の乖離を生める。
    - 作成しているモノと扱えるモノの間の乖離。
  - 適用可能な部分と適用する手法の同定。
  - 関連技術との連携。
    - テスト、レビュー、プロセス、開発方法論、etc.
  - 大規模システムの取り扱い。
    - 分離分割、焦点の当て方、抽象化。

取り組み

- モデル検査技術の実開発への応用法。
- ギャップを埋める。
  - モデル検査技術で扱える記述と実開発で作成するドキュメントの間にはギャップがある。
  - モデル検査技術のカスタマイズ。
  - ドキュメントの形式化。
- 現在の取り組み。
  - RTOS上のソフトウェアの検証。
  - UMLで書かれた設計モデルの検証。
  - ノウハウの形式化と蓄積法。
  - モデル検査技術習得のためのセミナー。

RTOS上のソフトウェアの検証

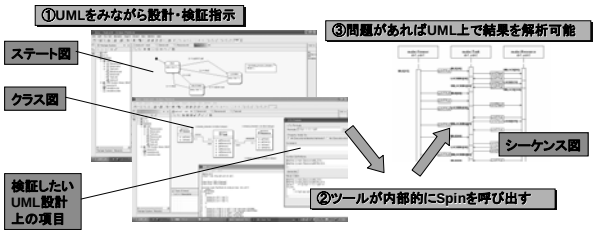
- 典型的な問題を検出できることを確認。
  - セマフォによるデッドロック、優先度逆転問題、生産者-消費者問題、読み手-書き手問題、など。
- 現在、中規模のソフトウェアに適用、評価中。
- その他
  - 周期、処理速度などをSpinで扱う手法。
  - RTOS自体の検証。

UMLによる設計記述の検証

- UMLによる設計記述の検証。
  - 対象：クラス図とステートチャート図を用いた記述。
  - Spinで扱える形式に自動的に変換して検証。
  - 反例をシーケンス図として出力。
- ソフトウェア工学的側面に焦点を当てる。
  - 形式検証技術をどのように適用すべきか？
  - プロダクトライン開発、繰り返し開発のための機能の盛り込み。

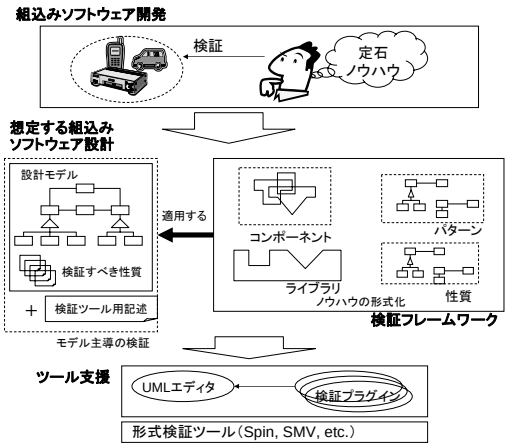
UMLによる設計記述の検証

- UMLのクラス図とステート図からPromela（Spin）のコードを生成。
  - ソフトウェア工学的な側面を重視。
    - プロダクトライン、繰り返し開発、etc.



ノウハウの形式化と蓄積

- モデル検査技術を使えるようになるのは容易。
- どのように適用するかが問題。
  - 問題毎に、振る舞いの書き方、検証する性質の書き方などが異なる。
  - 問題と解決法の整理が必要。
- 検証フレームワーク：ノウハウの形式化と蓄積を行うための枠組みの提案。



普及活動

- モデル検査技術習得のためのセミナー。
  - 状態遷移モデルの基本的な概念から、マルチタスクソフトウェアの検証への応用をカバー。
  - これまでに十数回開催。
    - 2～3日間（8時間×2～3）。
    - 独自、IPA/SEC, 日科技連, 人材養成
    - 資料の洗練。
- 現在は、モデル検査を使うための内容。
  - 実開発への適用法を含む内容へ拡張。

SEC-JAIST共同研究

- テーマ
  - 組込みソフトウェア高信頼性実現のための形式的手法実用化のための調査研究
- 目的
  - 形式的手法の実用化に向け、その技術を産業界に認知させ、どのような対象に対して適用することが効果的であるか を検討することを目的とする
- 内容
  - 業界が実際の適用性を検討できる事例を対象に、形式的手法の適用モデルケースを作成する

SEC-JAIST共同研究

- 05年度:　基本検討フェーズ
  - 例題による評価等を通じたフィージビリティスタディで有効性の見通しを得た
- **06年度:　技術調査検討フェーズ**
  - **Act-1: 適用手法に関する調査(上期)**
    - ソフトウェアに対する形式検証技術の適用技術に関し文献調査などを行い、技術動向・内容を把握する。
  - **Act-2:リアルな事例に基づく適用性検討（通年)**
    - 組込み分野の人にとって説得力のある事例を取り上げ形式的検証を適用することで、適用手法、適用の効果、適用上の課題や注意点などを明らかにする
  - **Act-3: 組込みへの適用に関する技術整理(下期)**
    - 上記の調査、事例研究を踏まえ、組込みへの形式的手法適用に関する知見を整理する。学会発表等も検討。
- 07年度:　実用化に向けた技術チューニングフェーズ

おわりに

- モデル検査技術はすでに実用段階になっている。
  - ツールが公開されているので、まずは使ってみては？
- モデル検査技術の適用法の検討。
  - モデル検査と実際の開発のギャップを埋める。
  - ソフトウェア工学の各技術との連携。
- ノウハウの蓄積。
  - 海外だけでなく日本のいくらかの企業も使い始めている。
  - ノウハウは公開されないので、各企業が地道に蓄積するしかない。