DEOS-FY2009-PP-07

プログラム検証による ディペンダビリティ支援手法

前田俊行
東京大学大学院情報理工学系研究科

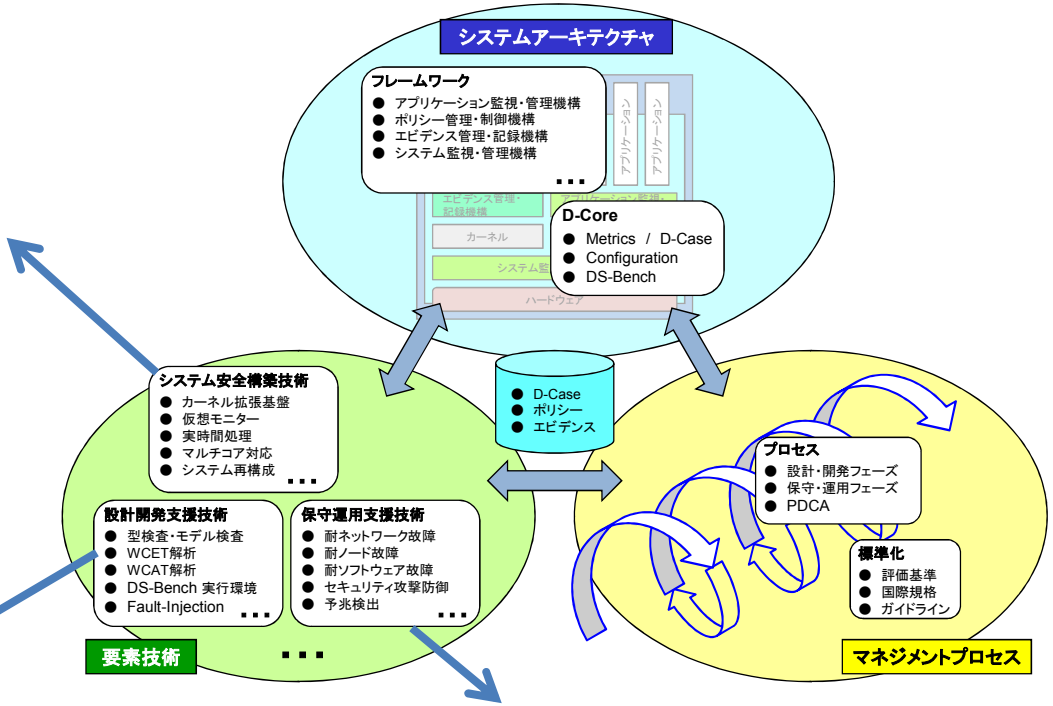
目次

- 研究領域全体における位置づけ・目的
  - ディペンダビリティ支援機構のディペンダビリティの実現
- アプローチ: プログラム検証
- 開放系障害要因とプログラム検証
- プログラム検証ツールの実装方針
  - 型検査器とモデル検査器
- 研究の進捗状況とデモ
- 重要な成果一覧
- 今後の研究の方向性
- まとめ

研究領域全体における位置づけ

システム安全構築技術

開発モジュール	チーム名
仮想モニタ モニタリング(VMO) マルコア制御(VMC)	中島
P-Bus Core 論理分割(LPAR)	石川



設計開発支援技術

開発ツール	チーム名
型検査・モデル検査(TCHK/MCHK)	**前田**
最悪実行時間予測(RETAS)	石川
電力使用量予測(GREEN)	徳田
Fault Injection (D-Cloud)	佐藤
ディペンダブルシステムベンチマーク実行環境(DS-Bench)	石川

保守運用支援技術

開発モジュール	チーム名
動作時間予約機構(TR) 耐故障ネットワーク機構(SCTP+FHO)	徳田
耐故障ネットワーク機構(RI2N/PEACH)	佐藤
アカウンティング機構(ACT)	中島/センター
シングルIPアドレス機構(SIAC)	石川

本研究の目的

- ディペンダビリティ支援機構そのもののディペンダビリティを支援する
  - 新しく導入した支援機構に問題があっては元も子もない
    - 「新しいコードを導入する」
      =「新しいバグが混入する」

本研究のアプローチ

- 「プログラム検証」技術を用いて
  ディペンダビリティ支援機構の
  安全性を検証する

プログラム検証とは

- プログラムを解析することにより
  プログラムがある性質を持つことを
  証明すること

- (例) Java 言語や Java バイトコードの型検査
  - Java プログラムのメモリ安全性等が**証明**される

プログラム検証とは何ではないか?

- いわゆるバグ発見器ではない
  - プログラムがある性質を満たしているかどうか
    を検査することを考える
    - 通常のバグ発見器はその性質に関する
      プログラム中の全ての問題を検出することはできない
      - 結局その後人手で検査したりテストしなければならない
    - プログラム検証器はその性質に関する
      プログラム中の全ての問題を検出することができる
- いわゆるテストツールではない
  - テストツールにはテスト漏れの可能性がある

プログラム検証の利点

- プログラム実行時に生じ得る問題を
  実行前に検出することができる
  - 「生じ得る問題」を「性質」とみなして検証すればよい
  - 例えば以下のようなことが可能となる
    - 不正なメモリアクセス防止、バッファオーバーフロー防止、
      想定外の入力に対する誤動作防止、ウィルス侵入防止、
      仕様に従わない API 利用の防止

開放系障害要因とプログラム検証

- 事前に全く想定していなかった問題を
  直接プログラム検証することは難しい
  - プログラム検証は基本的に
    既知の「ある性質」
    が満たされているかを検査するもの
  - 開放系障害要因が生じたときに
    プログラム検証はどういう支援ができるか?

開放系障害要因が生じたときの問題解決の流れ

1. 問題の原因を特定する
2. プログラムを修正する
3. 修正の正当性を検証する
4. 運用する
    1. へ戻る

1. 問題の原因を特定する

- 基本的にはディペンダビリティ支援機構 (P-Component) のロギング機能を用いる
  - ロギング機能が正しく実装されているか を検証することで間接的な支援が可能

2. プログラムを修正する

- 1. で特定した問題を「性質」として表して
  プログラム検証を行う
  - 事前に想定できなかった状況による「性質」にも
    利用環境の変換による「性質」にも対応できる

- 検証結果にもとづいて
  プログラムを修正する

3. 修正の正当性を検証する

- 2. で修正したプログラムに対して
  プログラム検証を行う
  - 基礎的な安全性 (メモリ安全性など) が
    修正によって破られないことを確認する
  - 1. で確認された問題 (性質) が
    修正によって確かに解決されていることを確認する

なぜプログラム検証ツールが使われてこなかったか

- C 言語に対応した検証ツールが少ない
  - Blast, CCured, Fail-safe C, Deputy, $CTAL_0$, …
- 「性質」が何となく分かっていてもそれを「検査仕様」として表現できない
  - このため、テストや人手によるコードレビューに頼らざるを得ないのが現状

本当に使ってもらえる検証ツール を実現するための我々のアプローチ

- C 言語プログラムを直接検証できるようにする
  - 他のプログラミング言語で書き直すなどもってのほか
- 検証前にソースコードを なるべく変更しなくて済むようにする
- 検査仕様記述の負担を減らす
  - 基礎的な性質に関しては自動的に検証するようにする

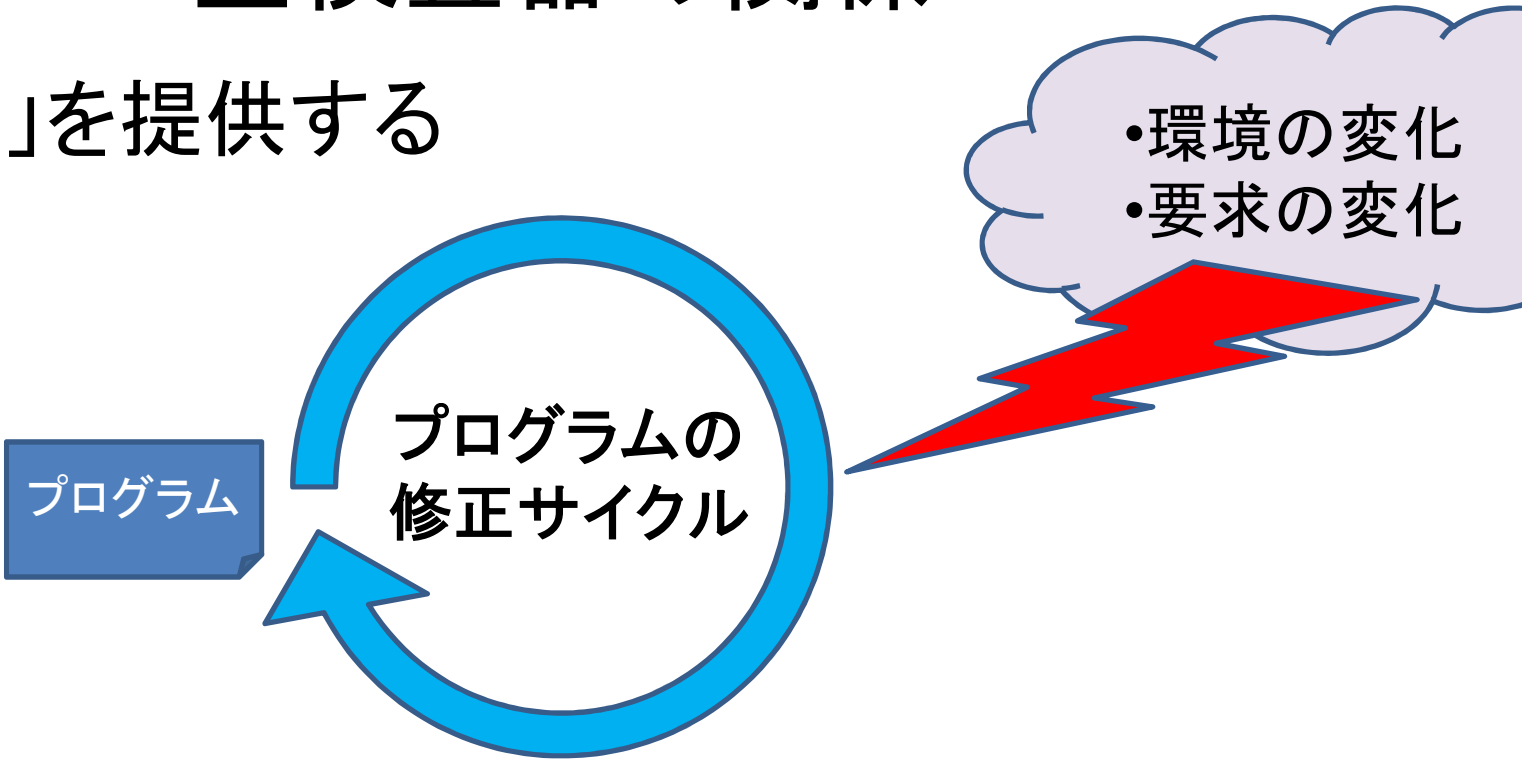
具体的には、2種類の検証ツールを作成中

2 種類の検証ツール

	型検査器	モデル検査器
検証できる安全性	メモリ安全性など基礎的な安全性	同期操作の安全性など高度な安全性
検証の対象	Cのソースコード 機械語プログラム	Cのソースコード
仕様記述	ほとんど必要なし	必要あり (検証したい性質を assertion として書く等)
実行時間	短い	長い

開放系障害要因と型検査器の関係

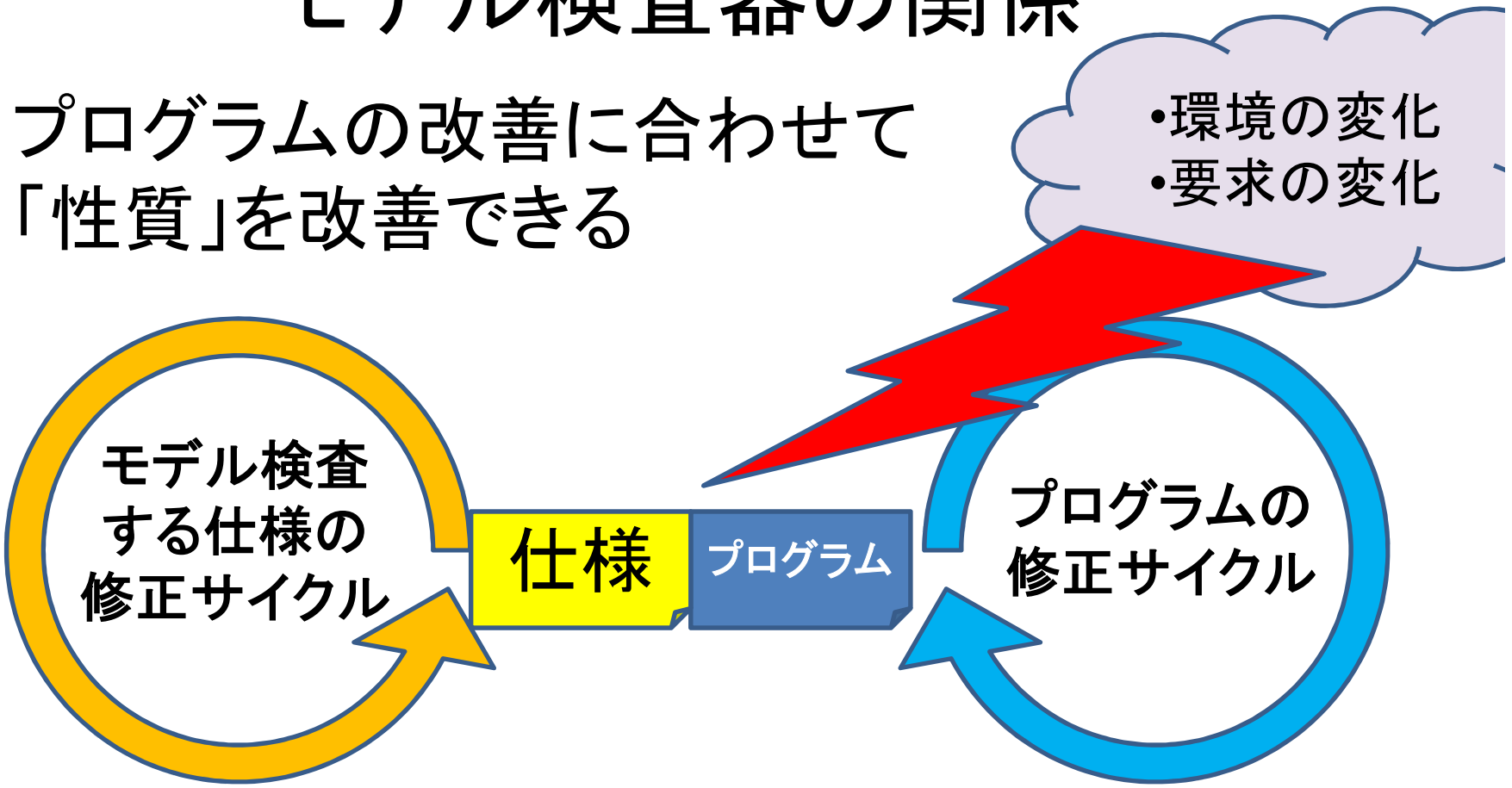
- 「土台」を提供する



型検査器によって
基礎的な安全性を
確保し続けられる

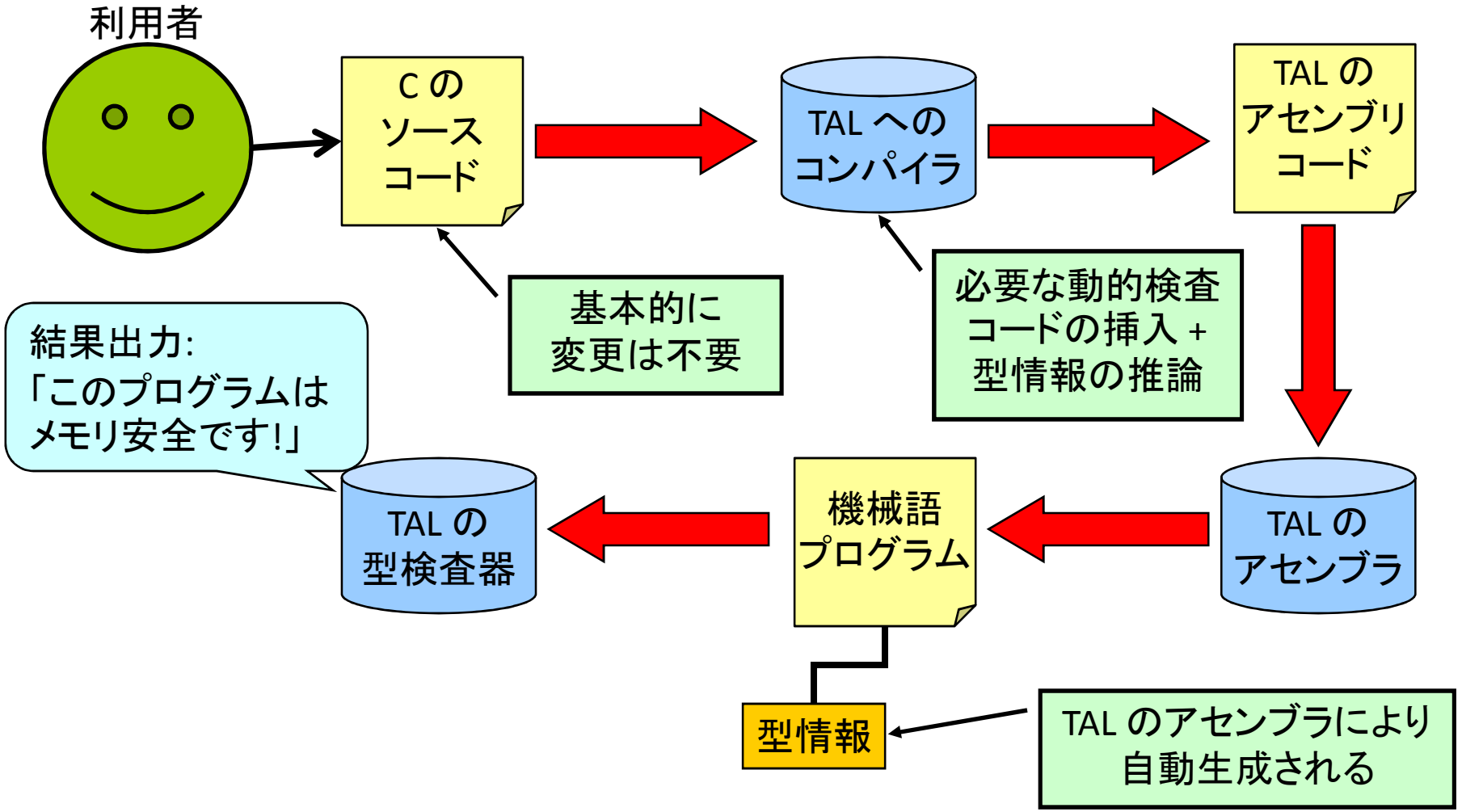
開放系障害要因と モデル検査器の関係

- プログラムの改善に合わせて「性質」を改善できる



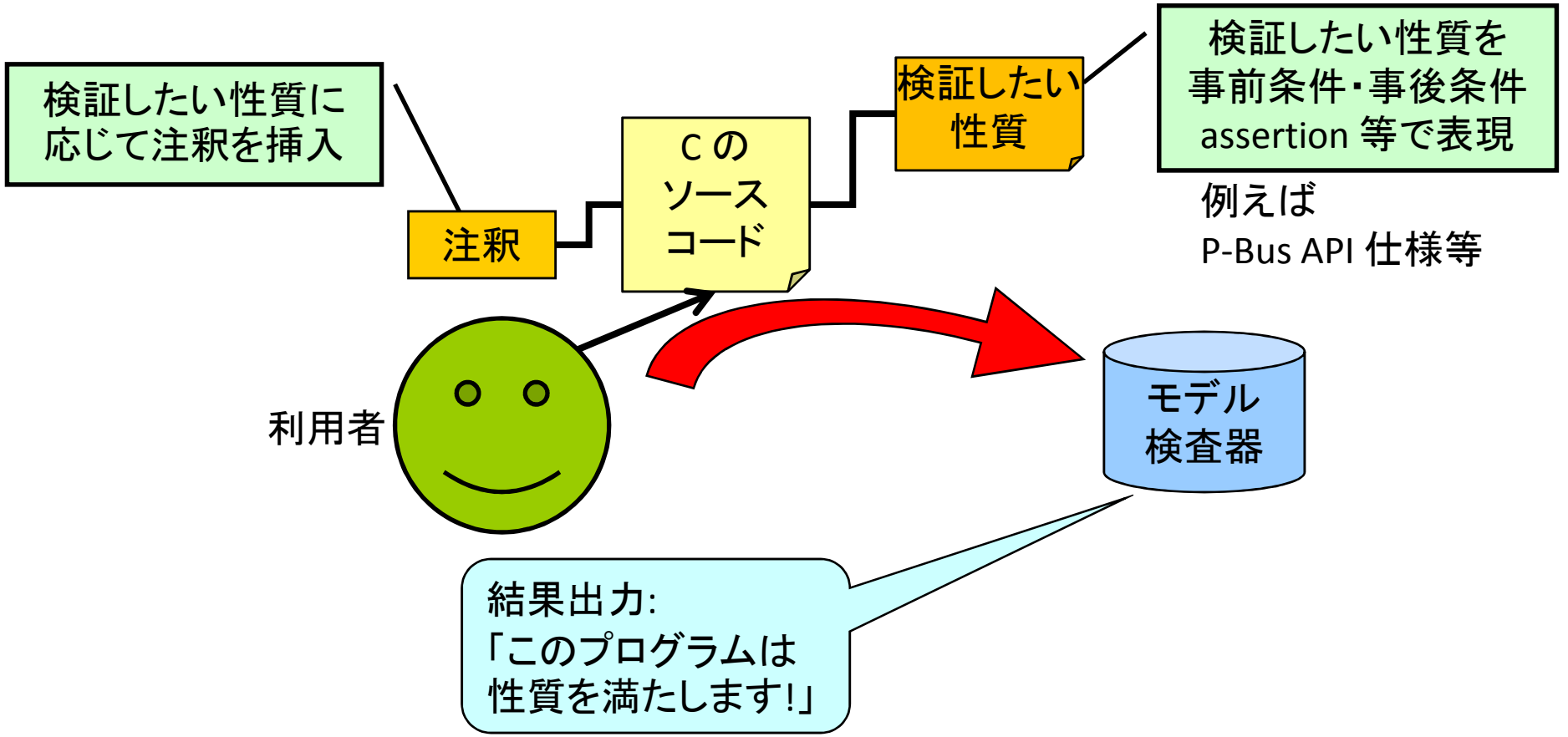
モデル検査器

型検査の全体像

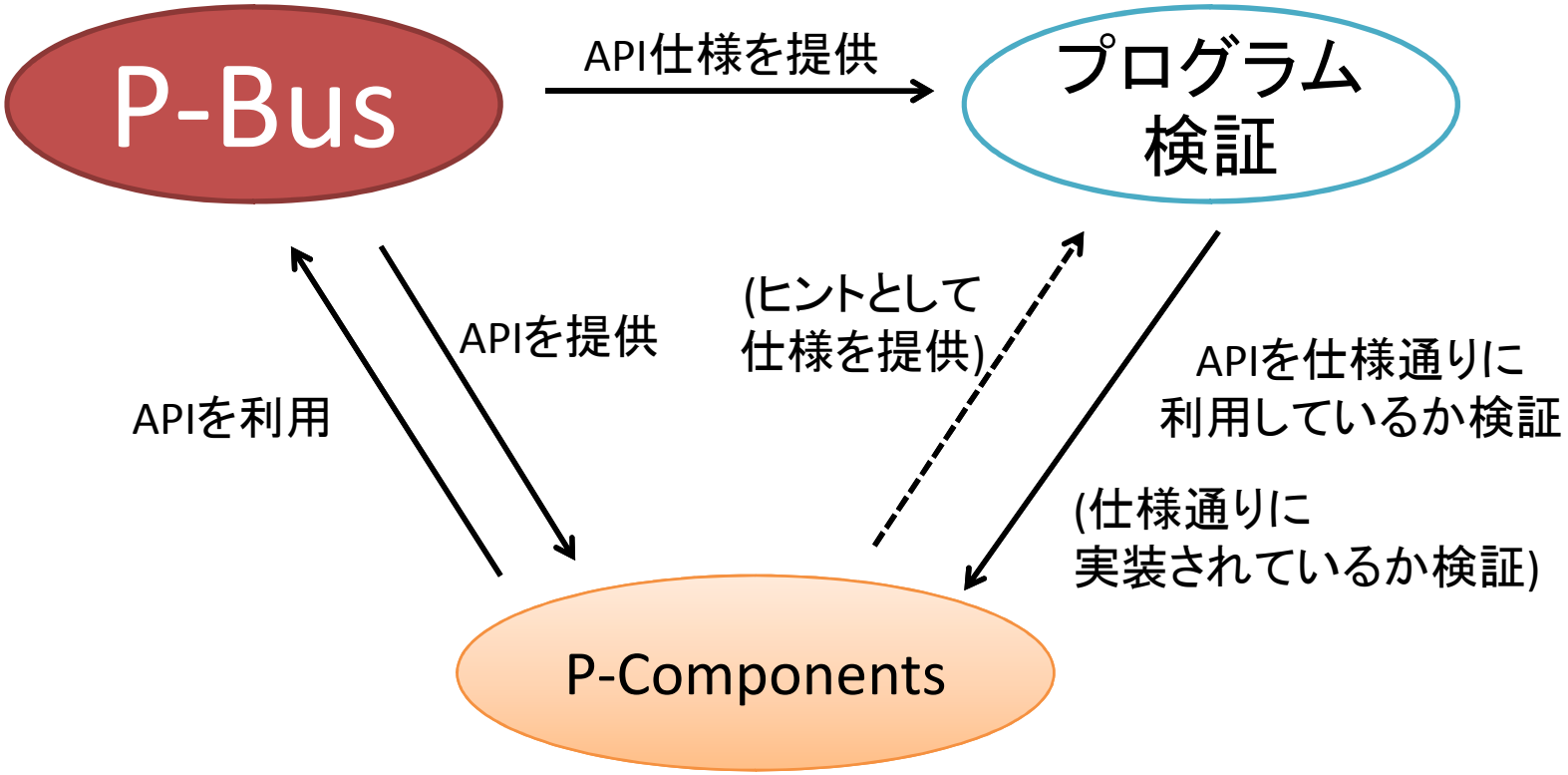


TAL (= 型付きアセンブリ言語)
アセンブリ言語・機械語のレベルで型検査が可能

モデル検査の全体像


検証したい性質に
応じて注釈を挿入
注釈
C の
ソース
コード
検証したい
性質
検証したい性質を
事前条件・事後条件
assertion 等で表現
例えば
P-Bus API 仕様等
利用者
モデル
検査器
結果出力:
「このプログラムは
性質を満たします!」

プログラム検証の P-Component への適用



仕様記述の例

```
/*@
    requires (size > 0);
    ensures (¥result != NULL ==> ¥valid(¥result, size));

    behavior Wait:
        assumes (flags == PBUS_ALLOC_WAIT);
        requires (pspc_atomicity == 0);

    behavior Nowait:
        assumes (flags == PBUS_ALLOC_NOWAIT);
*/
void * pbus_alloc(size_t size, pbus_alloc_flags_t flags)
{
    ...
```

仕様記述コメントブロックの開始

引数size>0が必要

関数の返り値がnon-nullなら帰り値の値から*size*バイトの範囲のメモリはアクセス可能

関数がブロックしうる場合の条件

関数がブロックしうるための条件

関数がアトミックなコンテキストで呼ばれていないこと

関数がブロックしない場合の条件

関数がブロックしないための条件

*size*バイトの連続したメモリブロックを確保して返す関数

P-Bus 仕様で対象している性質

- 型検査
  - メモリ安全性
    - プログラムが不正なメモリ操作を行わない
  - 制御フロー安全性
    - プログラムが不正なコード実行を行わない
- モデル検査
  - ロックの整合性
    - 確保していないロックを解放しない
    - ロックを2回確保しないなど
  - 実行コンテキストの整合性
    - ロックを確保したままスリープしないなど
- etc.

現在P-Bus 仕様で対象としていない性質

- 資源消費量安全性
  - メモリリークが生じないことなど
- タイミング制約
  - リアルタイム性の保証など
- etc.

研究の進捗状況とデモについて

- 現在の進捗状況:
  型検査器・モデル検査器ともにプロトタイプ実装を行い
  幾つかの P-Component (RI2Nなど) の検証を試みた
    - ただし現状では検証対象のソースコードに若干の修正が必要
  - 実際に幾つか問題を発見することができた
    - 既存の商用の解析ツールでは検出できない問題を発見することができた
    - モデル検査に要する時間は今のところ
      約3000行の検証対象 (RI2N) に対して数分～約30分

- 全体デモ
  - 今回使われているディペンダビリティ支援機構のうち
    RI2Nコンポーネントがプログラム検証されています

- 個別デモ
  - 型検査器・モデル検査器の効果や動作、
    P-Bus 仕様記述等について紹介します
    - 他の解析ツールとの比較なども紹介する予定です

重要成果リスト

- 査読付き国際会議発表
  - Takahiro Kosakai, Toshiyuki Maeda and Akinori Yonezawa, "Compiling C Programs into a Strongly Typed Assembly Language“, In Proc. of ASIAN'07, pp. 17-32, 2007.
  - Toshiyuki Maeda and Akinori Yonezawa, "Writing an OS Kernel in a Strictly and Statically Typed Language", Lecture Notes in Computer Science 5458, pp. 181-197, 2009.
- 公開ツール (http://web.yl.is.s.u-tokyo.ac.jp/~tosh/dsdt/)
  - $CTAL_0$: Compiler from the C programming language to Typed Assembly Language
  - L^3Cover: Framework for building program verifiers for Low-Level Languages

今後の研究の方向性

- ディペンダビリティ支援機構の検証等を更に進める
  - 検証器の有用性を確認する
- 検証器の完成度を向上する
  - 検証器自体の正確性・安定性の向上
  - 検証精度や性能の向上
  - 検証器の使い勝手の向上
    - GUIや統合開発環境との連携など
- 開放系障害要因の発生時の対応を検討する
  - プログラム検証と実行時ディペンダビリティ支援機構の連携による対応支援方法について検討を進める
- 現在まだ P-Bus 仕様で対応していない性質の検討

まとめ

- 本研究の目的
  - ディペンダビリティ支援機構そのものの
    ディペンダビリティを確保する
- 本研究の手法
  - プログラム検証によって
    ディペンダビリティ支援機構 (P-Component) の検証を行う
    - C 言語に対応した二種類のプログラム検証ツールを設計・実装する
      - 型検査器とモデル検査器
- 現在の進捗状況
  - プログラム検証ツールのプロトタイプ実装を行い
    実際に幾つかの P-Component の検証を試みた
    - 幾つかの問題を発見することができた