テストに依存しない，ソフトウェア欠陥を防ぐ効果的・効率的な形式検証の始め方

～ SPARK を用いて ～

伊藤昌夫
株式会社ニルソフトウェア

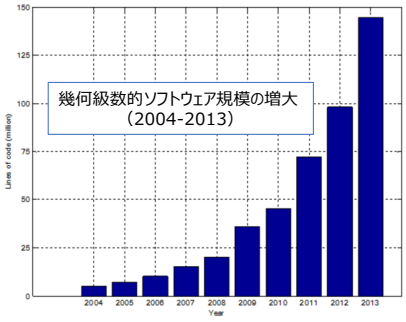
目次

1. 課題

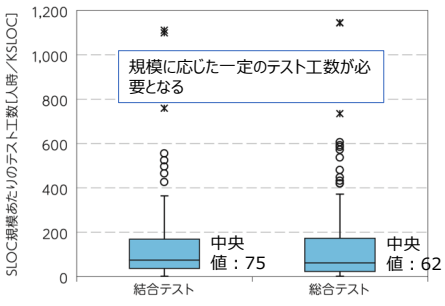
大規模・複雑化する車載ソフトウェア

大規模化する車載ソフトウェアとテスト

- 車載ソフトウェア規模は，年々増大している．同一規模において，一定以上のテスト工数が必要となる．従ってトータルのテスト工数は増大し，如何にソフトウェアをテストするか（或いは，しないで済ますか）は，現在も今後も課題である．



出典：Perspectives in Automotive Embedded Systems: From manual to fully autonomous vehicles



出典：組込みソフトウェア開発データ白書 2017, IPA

複雑さの増大：自動運転車における安全確保の難しさ

- 検証すべき様々な不確定要素が存在する

カテゴリ	ハザード	車両関連規格／仕様
欠陥	• ハードウェア要素の機能失陥 • ソフトウェアの誤り	ISO 26262 ed.2
予定機能の誤った実行	• センサー信号の誤った解釈・センサーデータと処理の組み合わせ誤り • HMIの誤使用	ISO/PAS 21448 (SoTIF)
不確実さの中での状況理解	• 未知の（環境）要素による安全性侵害	N/A

複雑かつ膨大な動作条件を考慮に入れる必要あり

さまざまな対策が必要になるが … 例えば

（安全）要求の正確な記述

検証手段の多様化（テスト と 形式検証）

2. 一つの解決策

テスト補完としての形式検証

形式検証とは

形式検証とは，「数学的なモデルを用いて，作られたプログラム（実装）が仕様と合致していることを確認する」こと

【補足】
多くの安全系の規格は，形式手法を使用することを推奨しています．
ISO 26262（自動車），EN 50128（鉄道），DO-178C（航空機）

形式検証の主たるアプローチ：

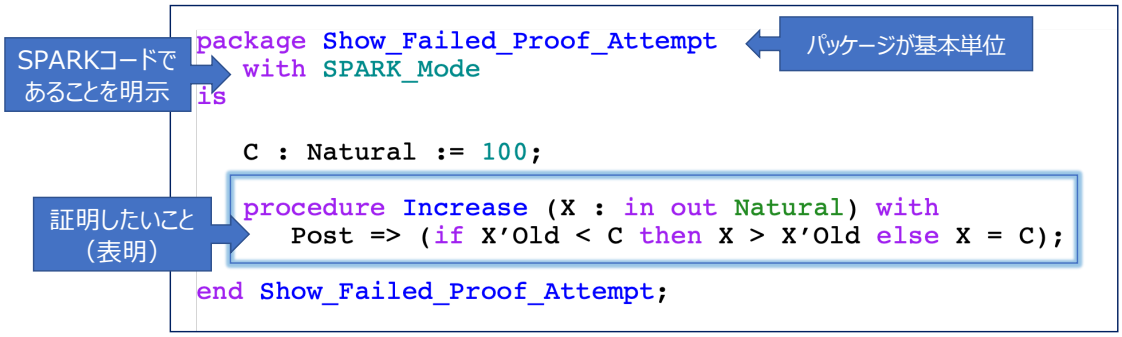
LTL：線形時間論理

- モデル検査	取り得る状態の網羅．望まない状態にならないこと（Safety），望む状態にいつかなること（Liveness）を検出．規模の問題がある．プロパティはLTL式(*)で記述．
- 演繹的アプローチ	プログラムの正しさを証明できる．但し，証明可能な範囲には制限がある（証明器の発展により制限は緩和される）．
- 抽象解釈	静的コード解析．検証プロパティの指定は基本的に不要．偽陽性を検出する場合がある．

Ada … SPARK

- Ada
  - 強い型付けを特徴とする静的なプログラム言語（最初の規格化は1983年）
  - 仕様と実装を分離
  - 言語設計時点で，開発環境も考慮（e.g. APSE）
  - 適用例：Boeing 777
- SPARK
  - Adaの部分的な拡張：「契約」的設計（AdaのAspects利用）
  - フロー分析／実行時検査／正しさの証明（一階の述語論理記述による）
  - 補助手段として，他に契約ケース・ゴーストコード等がある
  - 適用例：Euro Fighter，C130J

SPARK形式検証例：仕様部



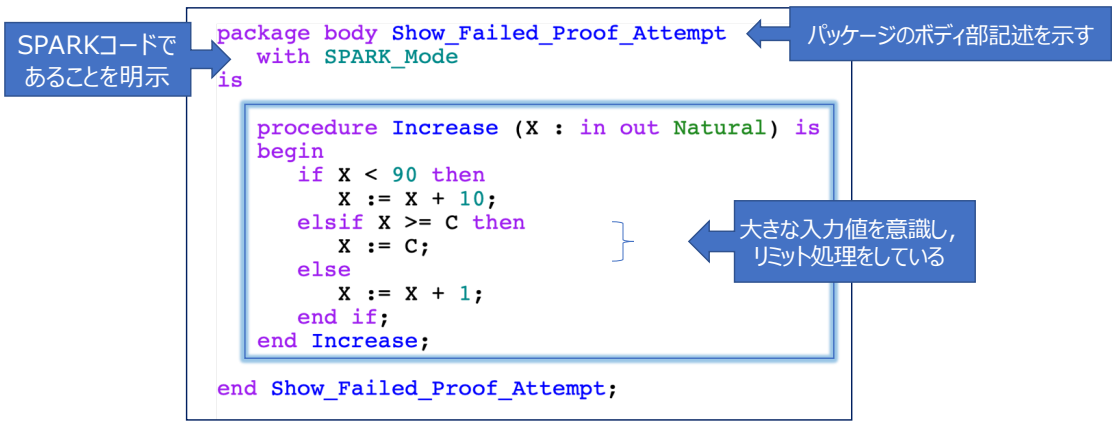

```
package Show_Failed_Proof_Attempt
   with SPARK_Mode
is

   C : Natural := 100;

   procedure Increase (X : in out Natural) with
      Post => (if X'Old < C then X > X'Old else X = C);

end Show_Failed_Proof_Attempt;
```

仕様部：手続き Increase と事後条件

事後条件では，手続きが終了したときに成立しているべき条件を示す．ここでは，入力*X*の値が変数*C*より小さいならば，出力*X*は，入力*X*より必ず大きくなり，それ以外の条件では変数*C*の値に等しいことを仕様として示している．これが，手続き利用者との「契約」となる．
なお，*X'Old* は，この手続きを実行する前の自然数*X*の値のこと．*X* は *in out* パラメータであり，値が変化する

SPARK形式検証例：ボディ部

- 以下にボディ部における実装を示す．先の仕様を満足している必要がある．




```
package body Show_Failed_Proof_Attempt
   with SPARK_Mode
is

   procedure Increase (X : in out Natural) is
   begin
      if X < 90 then
         X := X + 10;
      elsif X >= C then
         X := C;
      else
         X := X + 1;
      end if;
   end Increase;

end Show_Failed_Proof_Attempt;
```

ボディ部：手続き Increase の実装

SPARK形式検証例：実行結果

- 実装が仕様を満足しているかを確認する（メニューから，”Prove All Sources …” を選ぶ）
- OKであれば，この手続きはテストが不要になる

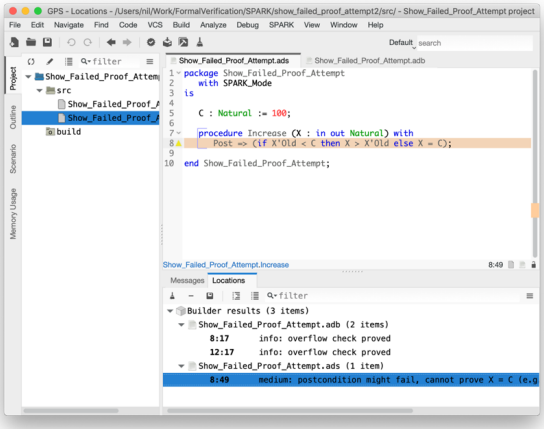
実行結果

```
show_failed_proof_attempt.ads:8:49: medium:
postcondition might fail, cannot prove X = C
(e.g. when C = 0 and X = 10 and X'Old = 0)
```

上記メッセージの意味：
事後条件が成立しない場合があります．

例えば，C=0, X = 10, X’Old = 0 の場合です．

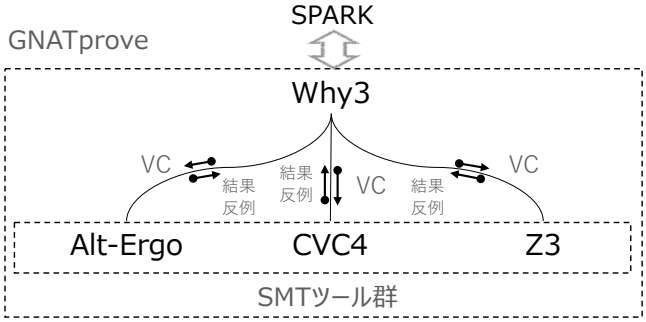
反例



GPS（IDE）画面

なぜ，SPARKは，形式検証に有利なのか

- 強い型付け言語Adaがベースになっており，仕様を適切に記述できる
  - 範囲チェックエラー,ゼロ割,オーバーフロー等は表明の記述なしに検査できる
- 表明表現は，言語の一部である
- 証明器を最初から開発環境に取り込んでいる



VC: Verification Condition （検証条件）
SMT: Satisfiability Modulo Theories

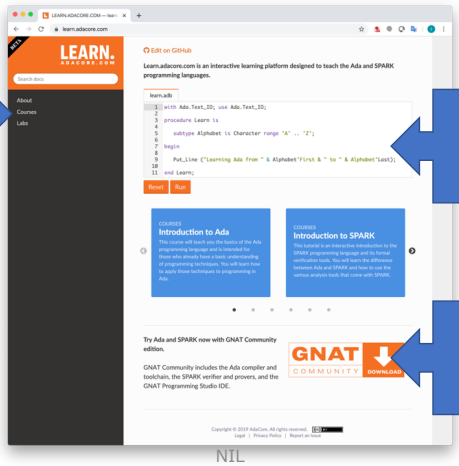
【注意】
証明できるのは，各SMTツールの能力の範囲においてです．
また，非線形の演算子は，通常扱うことができません．

今日から始めるためのポータル

https://learn.adacore.com/

オンライン
コース

- Introduction to Ada
- Introduction to SPARK
- Ada for the C++ or Java Developer
- SPARK Ada for the MISRA C Developer
- Getting Started with GNAT Toolchain



オンライン
実行

コミュニティ版
ダウンロード

まとめ

- 自動車に組みこまれるソフトウェアは，規模が年々増加している．今後の自動運転車の開発においては，さまざまな要素を考慮する必要があり，車載ソフトウェアは，より大規模・複雑化することが想定できる．従って，検証の負荷もますます高くなる．
- SPARKを用いることで，容易に，かつ正確に形式検証を行うことができ，検証に要する工数を大幅に減らすことができる．
  - SPARKには，一貫した記述法とツール支援がある
- もちろん，全てが形式検証可能なわけではない．しかし，大きな工数を要するテストと棲み分けすることで，トータルでコストや信頼性を向上させることが可能となる

付録A：SPARK応用例

1. 衛星フライトシステム
https://www.adacore.com/customers/flight-software-for-lunar-icecube-satellite

2. 航空管制システム
https://www.adacore.com/customers/uks-next-generation-atc-system

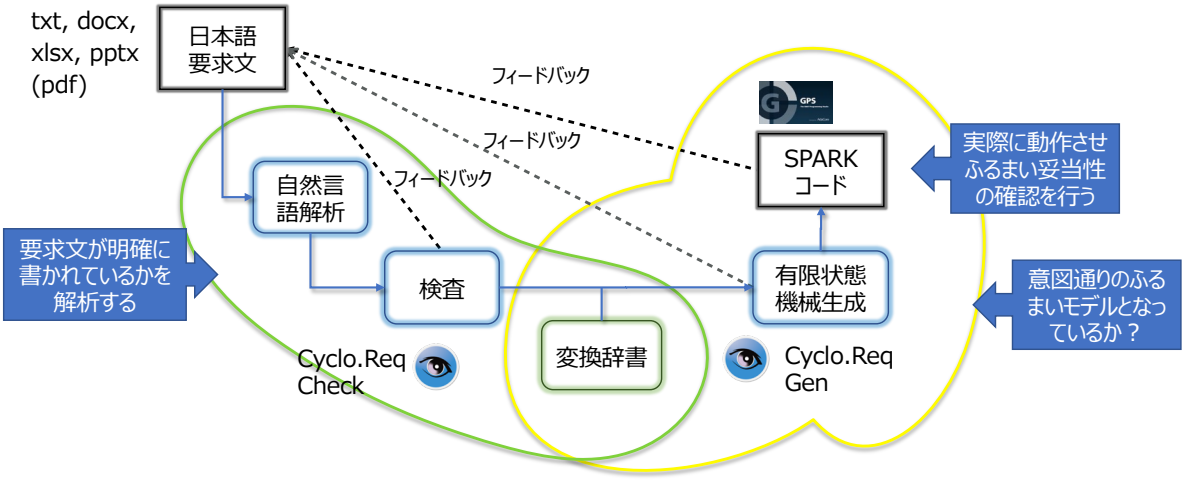
3. Security Workstation
https://www.adacore.com/customers/multi-level-security-workstation

4. NVIDIAのSOC（ADAS・自動運転車向け）
https://www.adacore.com/press/adacore-enhances-security-critical-firmware-with-nvidia

https://blogs.nvidia.com/blog/2019/02/05/adacore-secure-autonomous-driving/

付録B：Cyclo.Req

ふるまいモデルに着目し，要求段階でSPARKを利用しているツール．要求記述・妥当性の確認を支援する



Reference

1. Y. Moy, E. Ledinot, H. Delseny, V. Wiels and B. Monate, "Testing or Formal Verification: DO-178C Alternatives and Industrial Experience," in IEEE Software, vol. 30, no. 3, pp. 50-57, May-June 2013.
2. RTCA, Formal Methods Supplement to DO-178C and DO-278A, RTCA DO-333, 2011.
3. D. グリース，筧 捷彦 (訳)，プログラミングの科学，培風館, 1991.
4. SPARK 2014 ユーザーガイド（日本語版）
   https://www.adacore.com/uploads/techPapers/spark2014_ug.pdf
5. 日本 Ada 協会 https://ada.jp/

ありがとうございました