大学共同利用機関法人 情報・システム研究機構
国立情報学研究所
National Institute of Informatics

AI・自動運転時代の プログラミング技術への期待

@PPL 2020

国立情報学研究所　石川　冬樹
f-ishikawa@nii.ac.jp / @fyufyu
http://research.nii.ac.jp/~f-ishikawa/

研究者としての最近の主な活動

- 自動（運転）車のテスト・検証
  - 物理世界や確率などの連続値や微分方程式を含み複雑なシステムのテスト・検証・品質保証
  - 形式検証・テスト自動生成・最適化・機械学習などの技術を活用・併用する融合アプローチの追求
- 機械学習や広くAIを含むシステムの品質保証
  - コミュニティ活動，概念や技術の整理
  - テスト・検証技術の研究
- 形式手法の活用支援
  - 形式仕様の段階的詳細化における保守や再利用
  - 様々な手法・ツールの活用支援

本日の内容

- 話題提供
  - 研究事例：形式手法関連
  - 研究事例：自動運転のテスト関連
  - 動向紹介：機械学習工学

→ PPLの皆さんと議論したい！

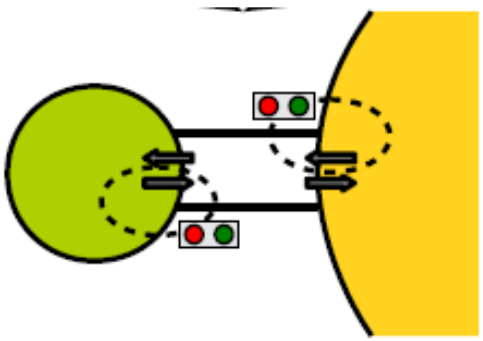
Shameless Advertisement

2つのプロジェクトで連携教員，研究員，
エンジニア，インターン，諸々募集中です！
（JST ERATO-MMSD, JST-MIRAI QAML改め eAI）
お気軽にお問い合わせください！

「プログラミング」の研究？

問題例

- 物理環境を含むシステム全体の中で
  適切にソフトウェア制御を定義
- 例題：大陸からつながる島
  - 「島と橋」に入れる車の台数に上限あり
  - 橋は一方通行（真ん中でお見合いになってはならない）
  - センサーで車の台数を検知しつつ（遅延あり）
    ソフトウェアが信号への制御指示を出し，
    上記の制約（不変条件）が守られるようにする

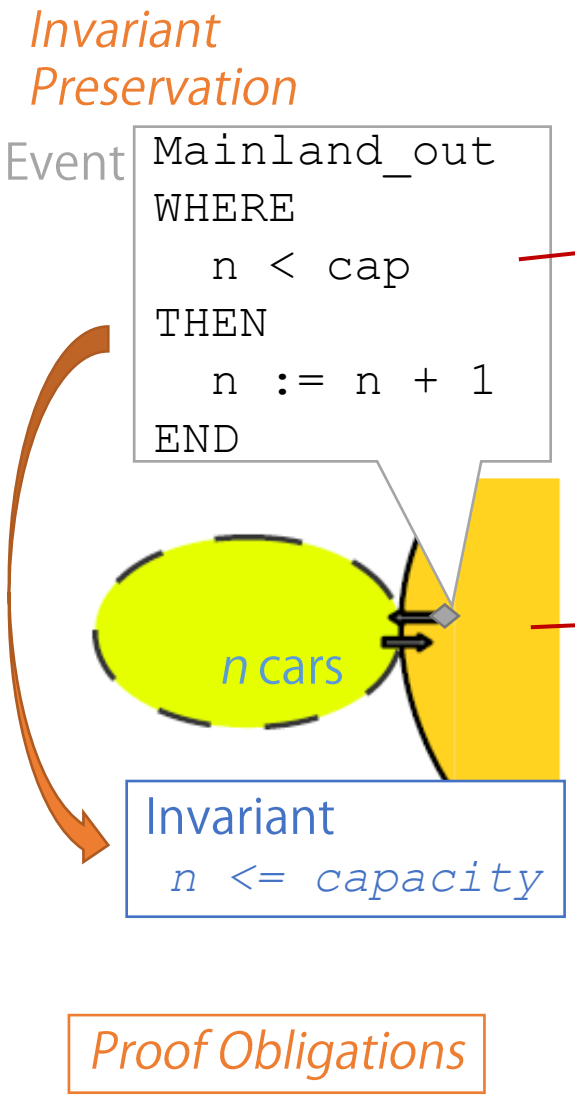


- ➡ 仕様定義の時点で複雑
- ➡ 段階的に完全・整合な仕様を定めたい
  （プログラムを得たい，ではない）

形式手法Event-B
最初の例題

[Abrial, Modeling in Event-B]

形式手法 Event-B



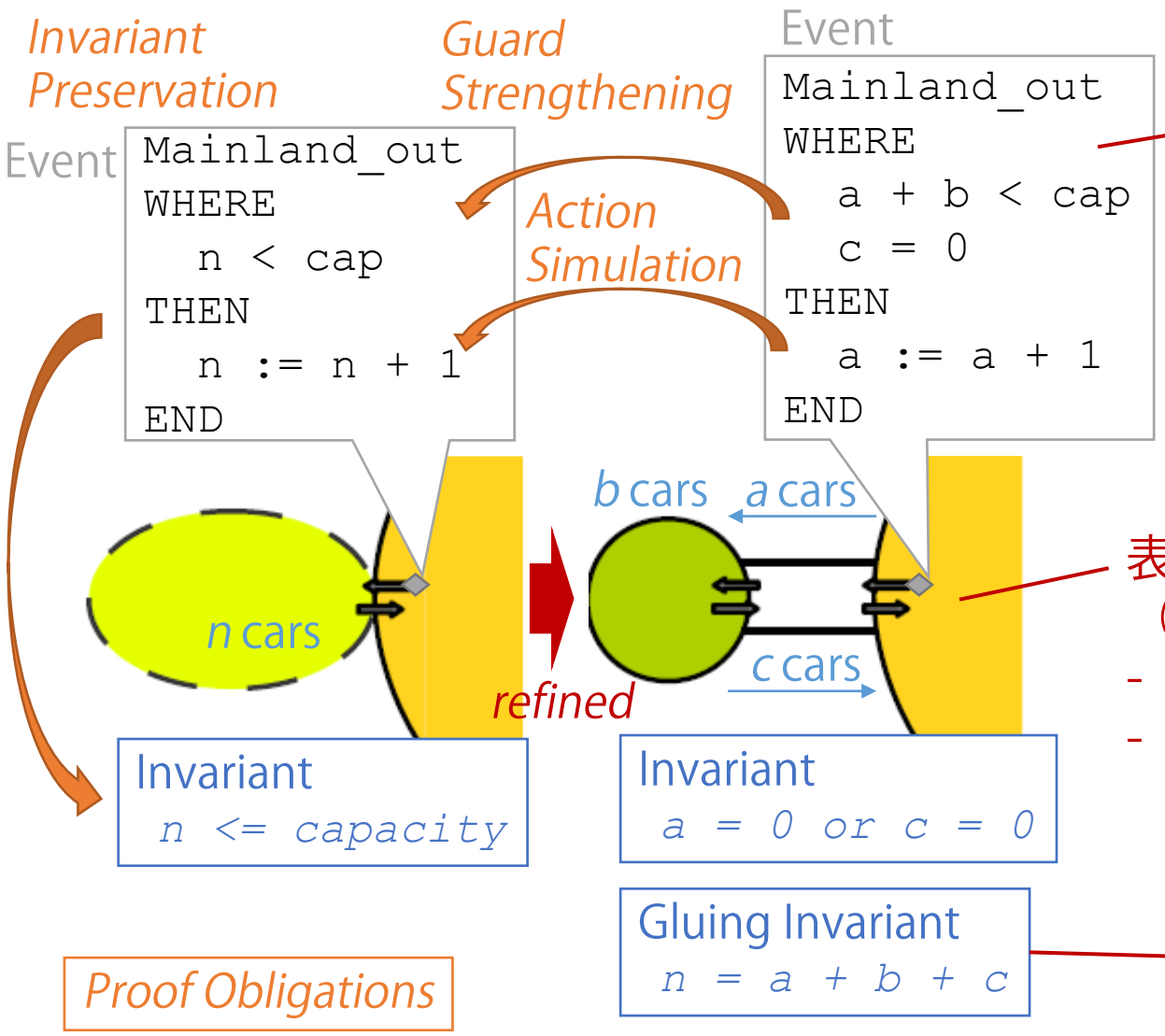
ガード条件の「チート」
- それぞれの車が現状の台数を把握して適切に止まってくれる
- これにより不変条件維持の証明完了

単純な抽象モデルから開始
- 世界の一部だけモデルに含めている
- 不変条件も台数上限だけ

※ 初期化や型定義は省略

[Abrial, Modeling in Event-B]

形式手法 Event-B



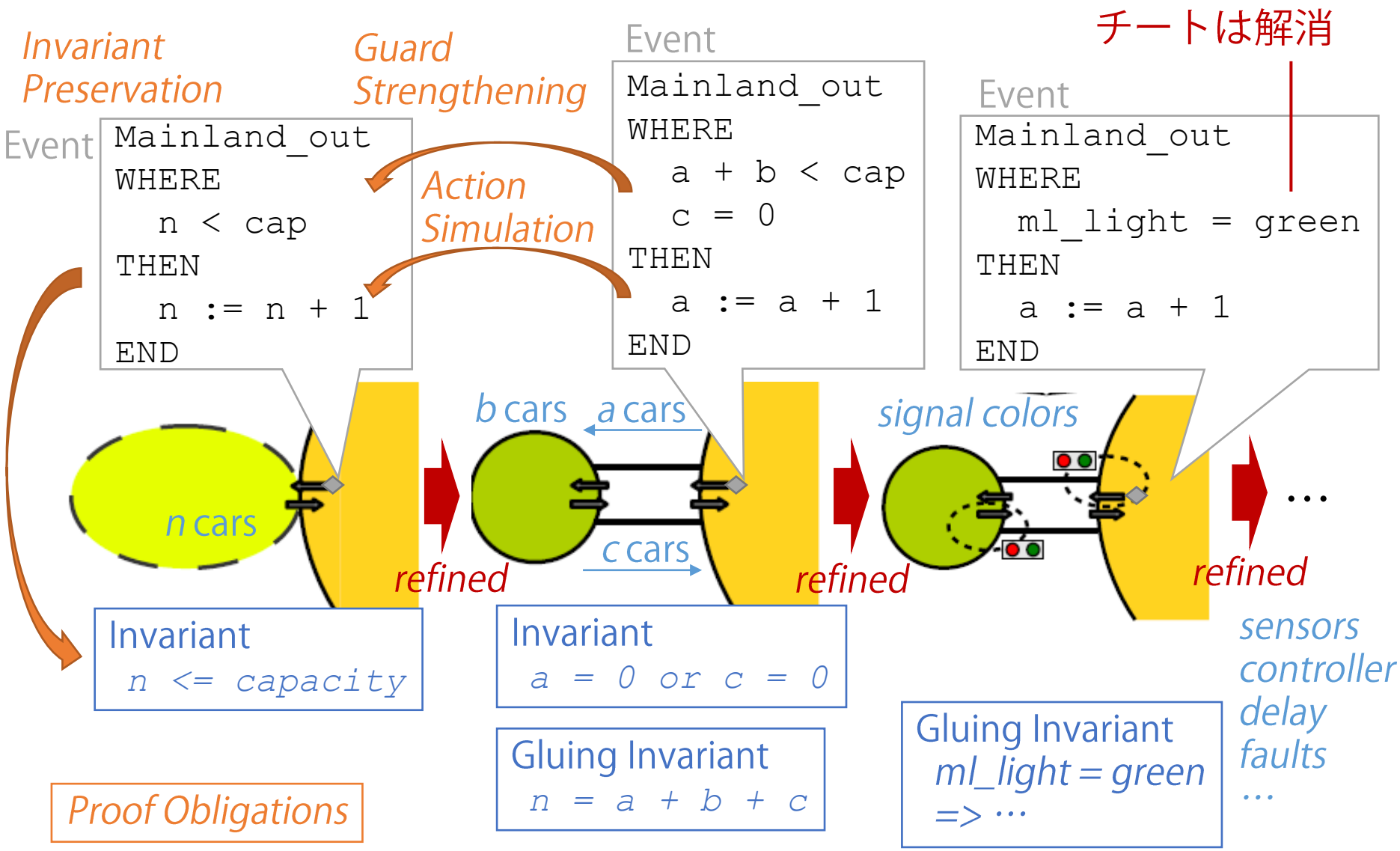
表現の置き換え
- 一方通行制約も追加
- まだ「チート」

表現変数の変化
（今回は置き換え）
- 一方通行についても考慮
- まだ信号やセンサーは考えていない

前のモデルとの対応

[Abrial, Modeling in Event-B]

形式手法 Event-B

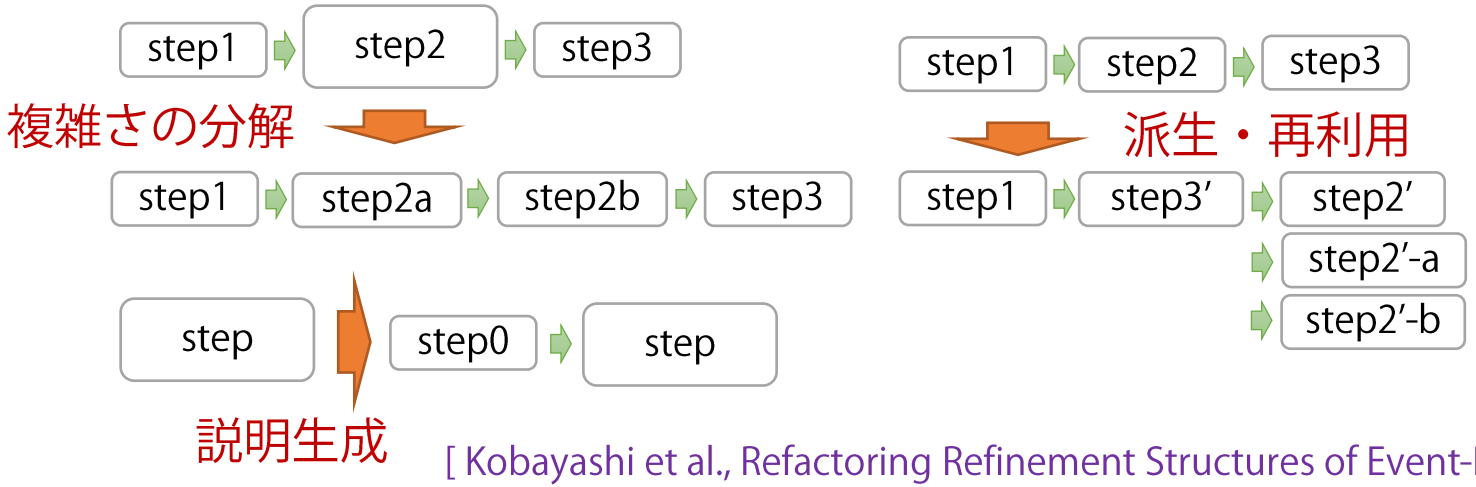


[Abrial, Modeling in Event-B]

おまけ：我々の研究事例

…というすべてを最初から見越してキレイにやるのは無理
（さらに後から派生・再利用の要求が生じる）

- **Refinement Refactoring**
  - より広くは Refinement Engineering
  - 既存の証明を崩さずにステップを分解またはマージ
  - ➡ 任意の構造変更可能に（定義不足等にならない形なら）



[Kobayashi et al., Refactoring Refinement Structures of Event-B Machines, FM'16]

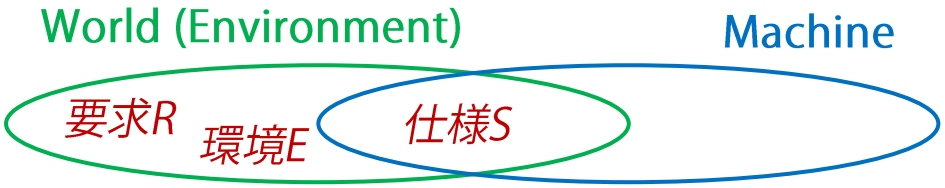
質問

- これ（Event-B，Event-Bが扱う問題）は，
  「プログラミング」・PPLの研究？
  - いや「モデリング」だ？
    ここから「コンパイラ」によりアセンブラが出るとしたら？
  - 「信号が青なら車が進む」は，計算機関係ない？

別の例：どの部分が「プログラミング」？

■Zave/Jacksonのモデル

$$S,\ E \models R$$



マシン（開発の成果物・システム）の仕様 S は，
想定環境（ドメインの性質） E の下で要求 R を満たす

[Zave et al., Four Dark Corners of Requirements Engineering, 1997]

■例：

[画像： http://www.fujitaka.com/]

要求 R	• enter 発生回数 ≦ pay 発生回数
仕様 S	• pay の発生を検知したら，unlock する • push の発生を検知したら， lock する
環境 E	• push と enter は交互にしか起きないと仮定 • lock が起きてから unlock が起きるまでは push は起こせないと仮定

ここの実現だけ扱う？

別の例：これも「プログラミング」

- Chef 言語 (Infrastructure-as-Code)

```
file ‘/tmp/a.txt’ do
    content(“copied:” + IO.read(‘/a.txt’))
    action: create
end
```

Prefixを足してファイル内容をコピー

```
file ‘/a.txt’ do
    content IO.read(‘/tmp/a.txt’)
    action: create
end
```

ファイル内容をコピー

- 各スクリプトは「べき等」：
  途中で落ちたり終了確認がとれなかったりしたとき
  何も気にせず再実行を繰り返してよい
- しかし，これらの逐次合成はべき等でない！！

全可能性をテスト
[Hummar et al., Middleware’13]

中間言語と形式検証
[Shambaugh et al., PLDI’16]

外部スクリプト等の影響で形式検証できない部分だけテスト
[Ikeshita et al., TAP’17]

別の例：これも「プログラミング」

- Scenic

Figure 1. Three scenes generated from a single ~20-line SCENIC scenario representing bumper-to-bumper traffic.

```
1 wiggle = (-10 deg, 10 deg)
2 ego = Car with roadDeviation wiggle
3 c = Car visible,
4         with roadDeviation resample(wiggle)
5 leftRight = Uniform(1.0, -1.0) * (1.25, 2.75)
6 Car beyond c by leftRight @ (4, 10),
7     with roadDeviation resample(wiggle)
```

Figure 8. A scenario where one car partially occludes another. The property `roadDeviation` is defined in `Car` to mean its heading relative to the `roadDirection`.

[Fremont et al., Scenic: A Language for Scenario Specification and Scene Generation, PLDI'19]

ここまでのまとめ

「プログラムを作るのがプログラミング研究」で止まらず閉じずに

「問題解決」の研究を一緒にしませんか？

- 「計算機に指示を直接出すこと」にとらわれない
- 「世界のあらゆるモノ・コトを動かすこと」
  （「やりたいこと」「世界に仮定すること」も扱うはず）

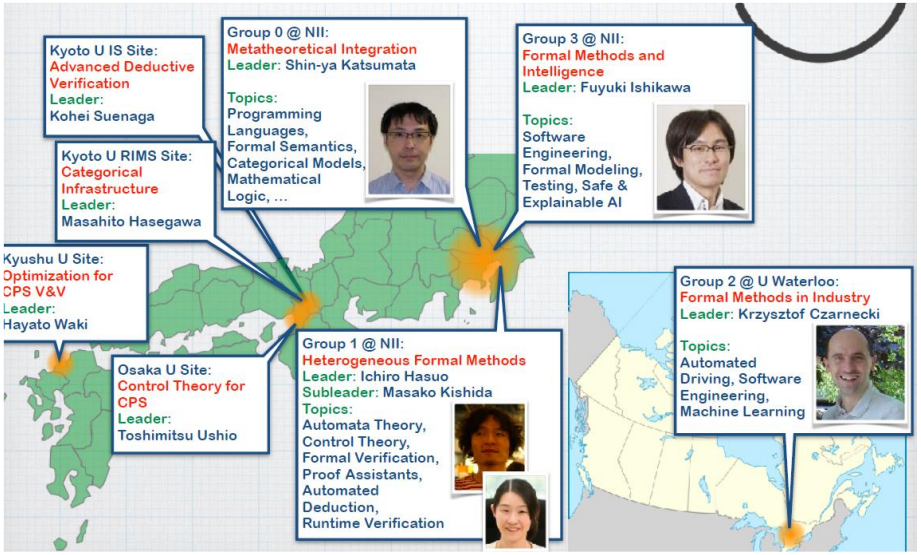
世界にはPPLの皆さんなら「もっとうまくできる」ことがたくさん転がっている！！

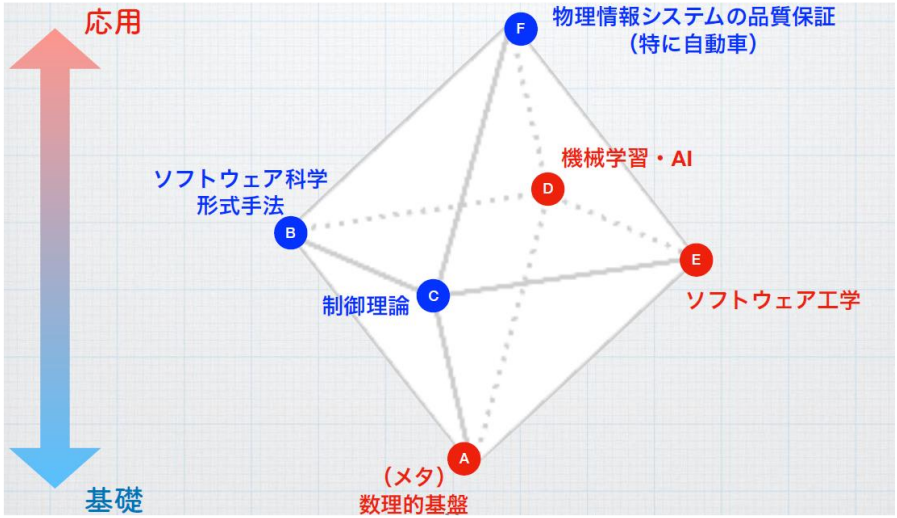
自動運転のテスト

JST ERATO-MMSDプロジェクト

http://group-mmm.org/eratommsd/

- Cyber-Physical Systemsの信頼性
  - ソフトウェアに対する形式手法・テスト・信頼性技術を，連続系を扱うように発展
  - 特に自動運転（後述する「AIの安全性」も）





下記にある NII 蓮尾先生のスライドより
https://group-mmm.org/eratommsd/ja/vvav-symposium-201905/

一般ソフトウェアと自動車システム

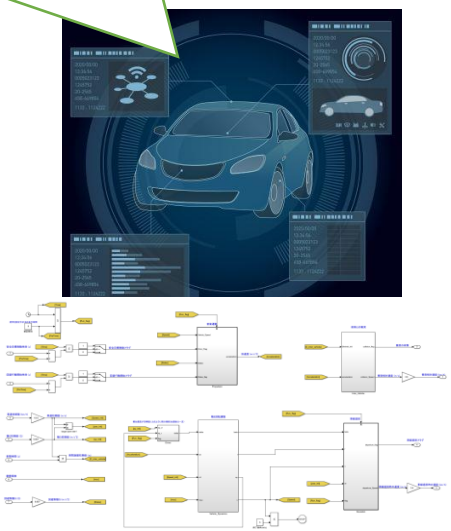
■自動車システムを同様な形でテストする？

入力

ユーザの挙動
- アクセルペダル
- ブレーキペダル

環境
- 初期速度
- 歩行者の位置・動き
- 路面状況
- …

シミュレーションモデル
例：自動ブレーキ付の車の挙動



出力
- 衝突有無
- 衝突速度

正否判定だけでなく
連続値で結果を評価

連続値をとる無数の状態

現実的ではない値の組も

プログラムとは
異なる論理
（微分方程式等）

物理挙動を扱う形式手法： 例

- Differential Dynamic Logic

微分方程式による状態変化を含む
「プログラム」

$$x <= m \land C(x,v) \; [a := -b; \; (x' = v; \; v' = a)] \; x <= m$$

Precondition on
position *x* and velocity *v*

Deaccelerated by braking -b

Not go beyond
the position *m*

$C \equiv v^2 <= 2b(m-x)$
（推論規則を当てはめて
最終的には判別式を利用）

[Platzer, Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics, 2010]

物理挙動を扱う形式手法

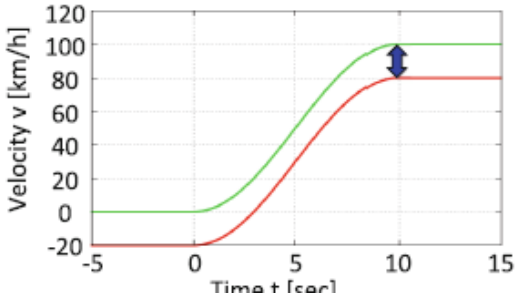
- 様々な手法・ツール
  - 例： 先ほどの「プログラム」に対する定理証明器 KeYmaera [http://symbolaris.com/info/KeYmaera.html]
  - 例：ハイブリッドオートマトンに対する到達性解析ツール Flow* [https://flowstar.org/]
  - ・・・
- が，スケーラビリティ（そもそも決定可能性）や扱うスキルが心配

➡ *ある種の「テスト」（実行の繰り返し）によるアプローチへ*

形式手法分野での一潮流：反例探索

■検証式の定量化

シミュレーション結果例（２サンプル）



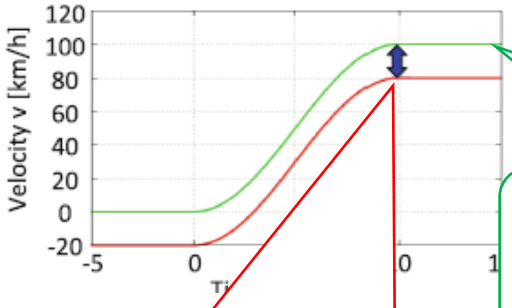
イベントBが起きてから10秒以内に
速度は80km/h 以上になる

[Fainekos et al., Robustness of temporal logic specifications for continuous-time signals, TheoCompSci'09]
Figure from [Akazaki et al, Time Robustness in MTL and Expressivity in Hybrid System Falsification, CAV'15]

形式手法分野での一潮流：反例探索

■定量検証式の定量的な充足評価

シミュレーション結果例（２サンプル）



イベントBが起きてから10秒以内に
速度は 80km/h 以上になる

OK！10秒後，求められたよりも20km/h余裕を持って

ロバストな充足

OK！10秒後，ちょうどギリギリ求められた値クリア

危うい充足（違反に近い）

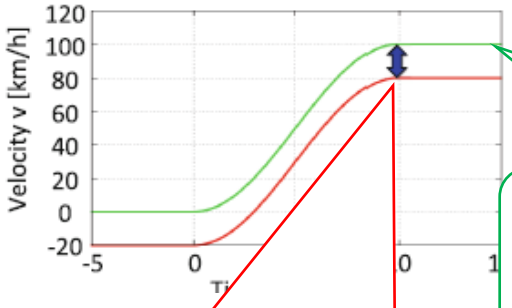
（時間についても同様な議論ができる）

[Fainekos et al., Robustness of temporal logic specifications for continuous-time signals, TheoCompSci'09]
Figure from [Akazaki et al, Time Robustness in MTL and Expressivity in Hybrid System Falsification, CAV'15]

形式手法分野での一潮流：反例探索

■定量検証式に対する最適化を用いた反例探索

シミュレーション結果例（２サンプル）



イベントBが起きてから10秒以内に
速度は 80km/h 以上になる

OK！10秒後，求められたよりも20km/h余裕を持って

ロバストな充足

OK！10秒後，ちょうどギリギリ求められた値クリア

危うい充足（違反に近い）

（時間についても同様な議論ができる）

➡「ロバスト度合い」の最適化問題に帰着

[Fainekos et al., Robustness of temporal logic specifications for continuous-time signals, TheoCompSci'09]
Figure from [Akazaki et al, Time Robustness in MTL and Expressivity in Hybrid System Falsification, CAV'15]

ソフトウェアテスティング分野の一潮流

- サーチベースドテスティング（Search-based）
  - 最適化技術（特に進化計算・メタヒューリスティック）により「欲しいもの」を表すスコアを最大化するようなテストスイートやテストケースを生成
    - 例：「車が人に最も近づくような危ういテスト」ケースを生成
    - 例：「前バージョンと比べ出力が大きく変わる」入力を発見
    - 例：「高カバレッジで数が小さい」回帰テストスイートを生成

[S. Ali et al., Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation, 2010]
[http://www.evosuite.org/]

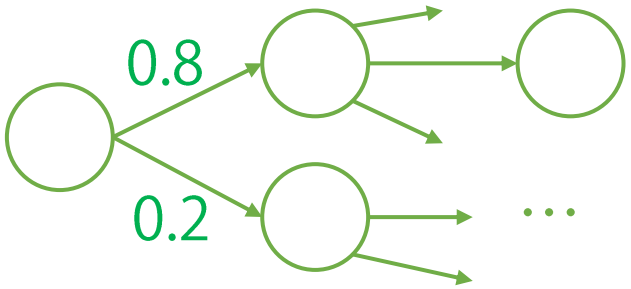
おまけ：サーチベースドテスティングのレベル

- 2018年のJava Unit Testing Competition
  - 比較用に複数ツールを組み合わせた「最強ツール」は，人が作ったものよりよいテストスイートを10秒で生成
  - ここでの評価基準は，コードカバレッジとミューテーションスコア（人工バグの検出率），テストケース数
- Facebookでの事例
  - モバイルアプリのContinuous Integrationに組み込み
  - テスト数が少なく，コマンド数が短く，多くのクラッシュを報告するようなテスト一式をまとめあげる
  - バグ自動修正ツールも連動

[https://github.com/PROSRESEARCHCENTER/junitcontest/blob/master/README.md#6th-junit-contest]
[Molina et al, Java Unit Testing Tool Competition - Sixth Round]
[https://code.fb.com/developer-tools/finding-and-fixing-software-bugs-automatically-with-sapfix-and-sapienz/]

形式手法分野での別の一潮流

- 確率モデル検査
  - マルコフ過程などの確率モデルに対する検査

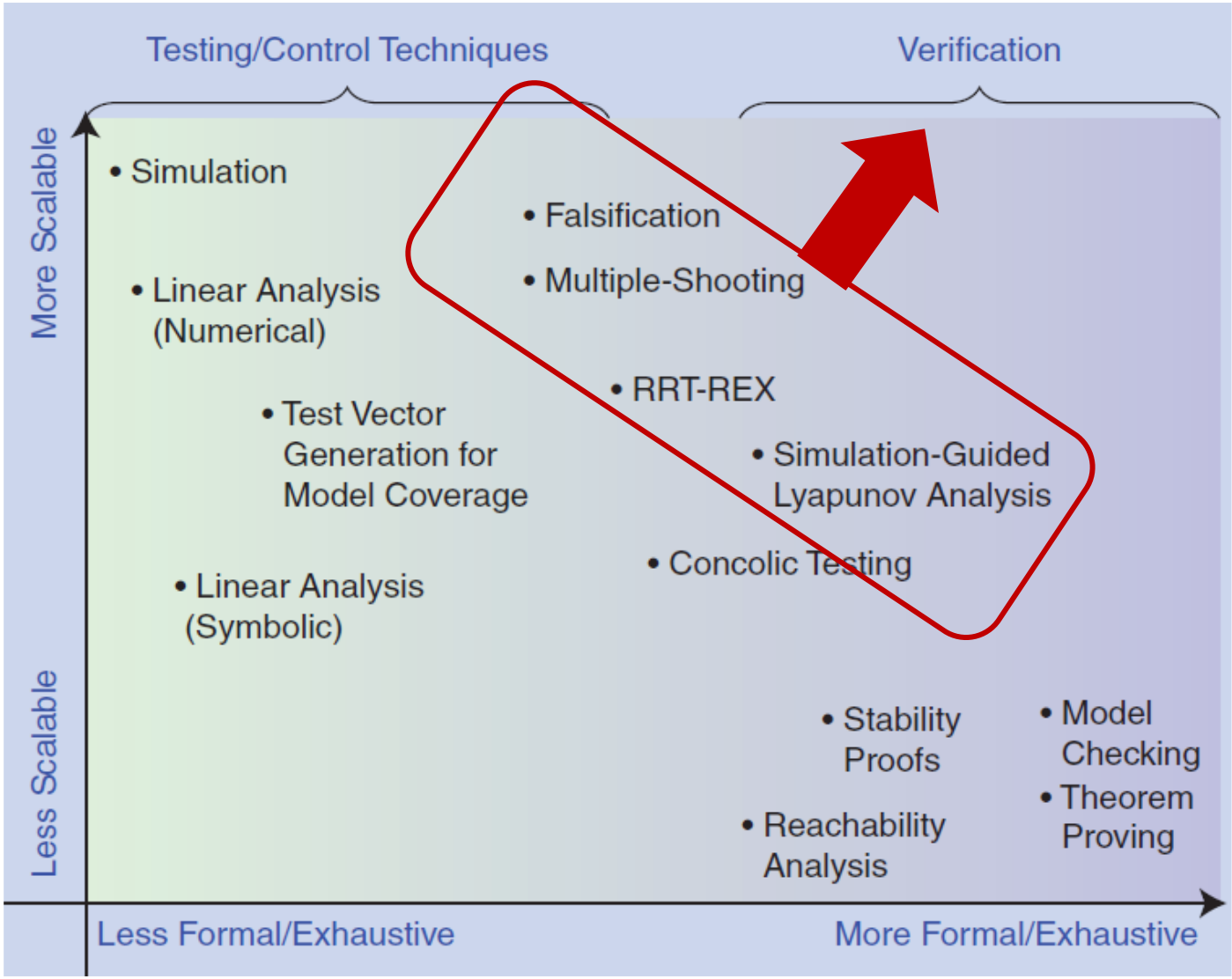


- 発生確率の計算は高コスト

- 検証項目に対する**仮説検証**をしてしまう
  （統計的モデル検査）
  - とにかく大量に実行してしまえばよい
  - ドメイン知識・掘り出した知識で加速可能

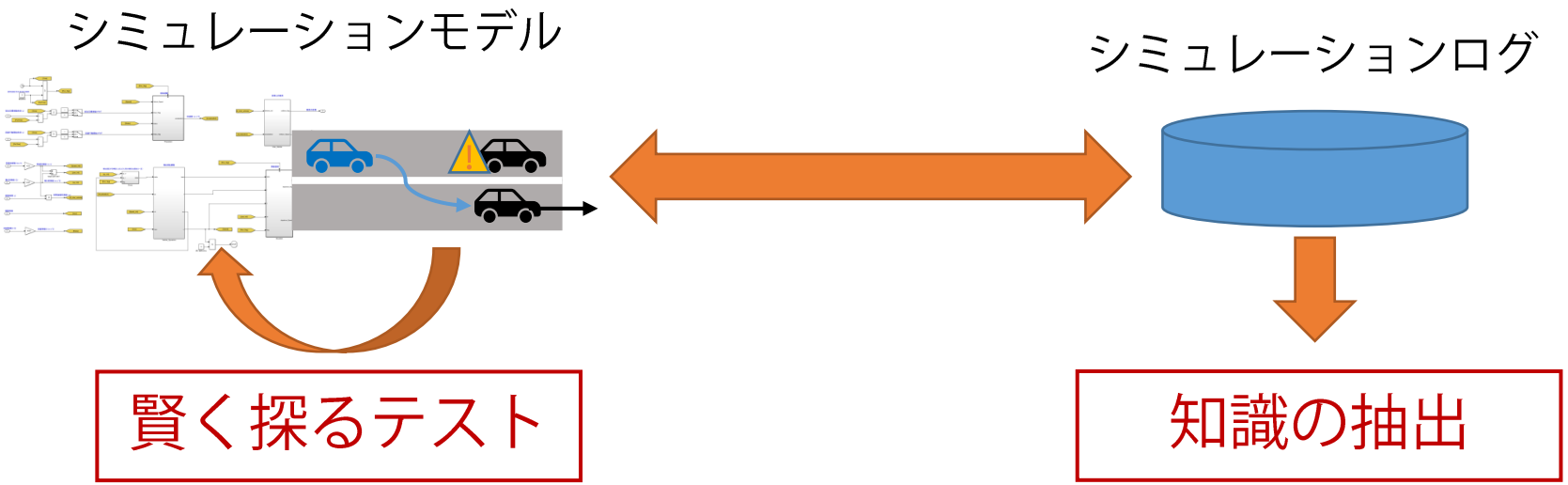
[Jha et al., A Bayesian Approach to Model Checking Biological Systems, CMSB'11]
[Zuliani et al, Bayesian statistical model checking with application to Stateflow/Simulink verification, FMSD'13]

さらに制御分野の潮流



[Simulation-Based Approaches for Verification of Embedded Control Systems, IEEE Control Mag.'16]

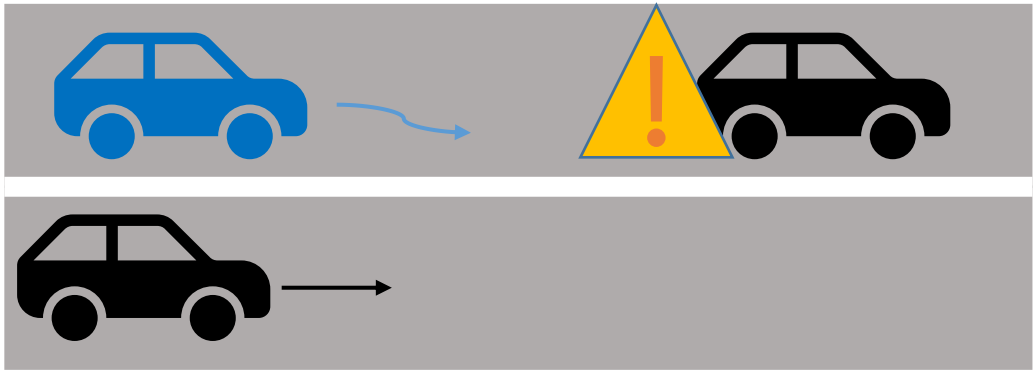
我々の研究事例2点



1. 経路計画プログラムにおける衝突発生シナリオの検出
2. 安全性に影響する要因の説明

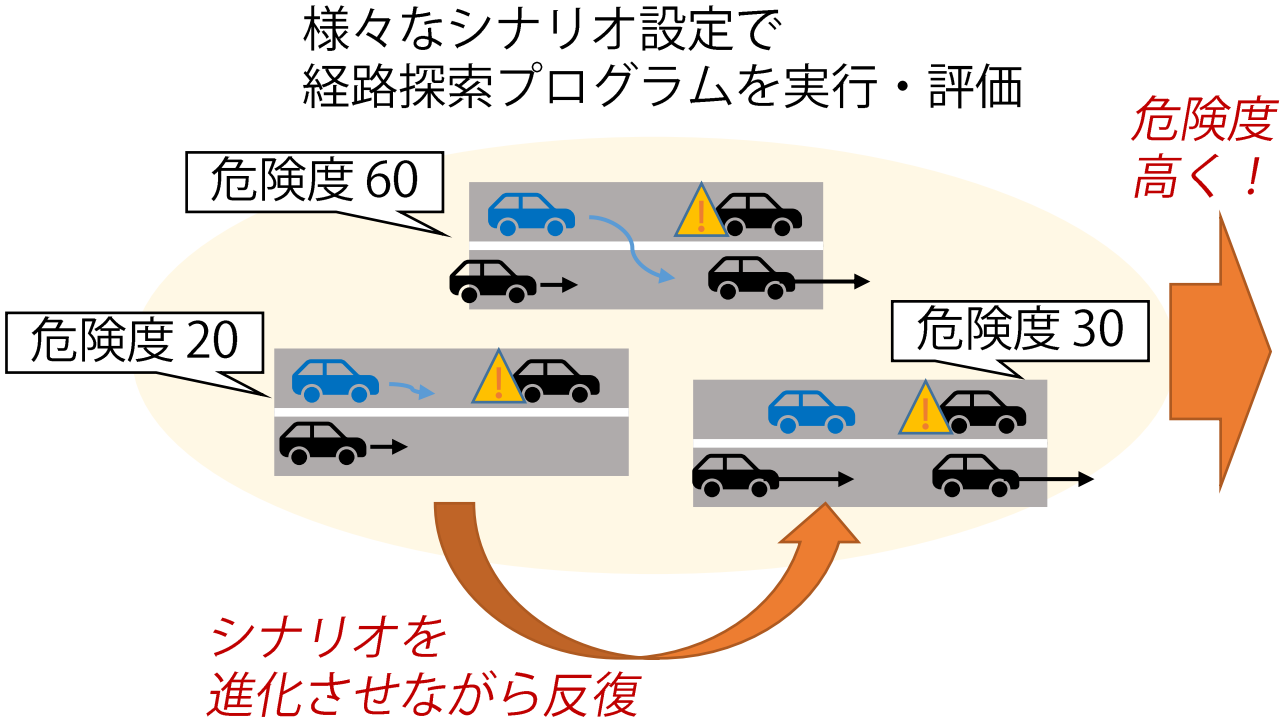
研究事例（１）：衝突発生シナリオの検出

- 経路計画プログラムのテスト
  - 他車や歩行者の位置や動き，道路形状，信号配置・変化などを設定し，「適切に」運転できるか検査したい
  - 安全性が一番重要
  - 他にも，快適さやレーン遵守なども評価



研究事例（１）：衝突発生シナリオの検出

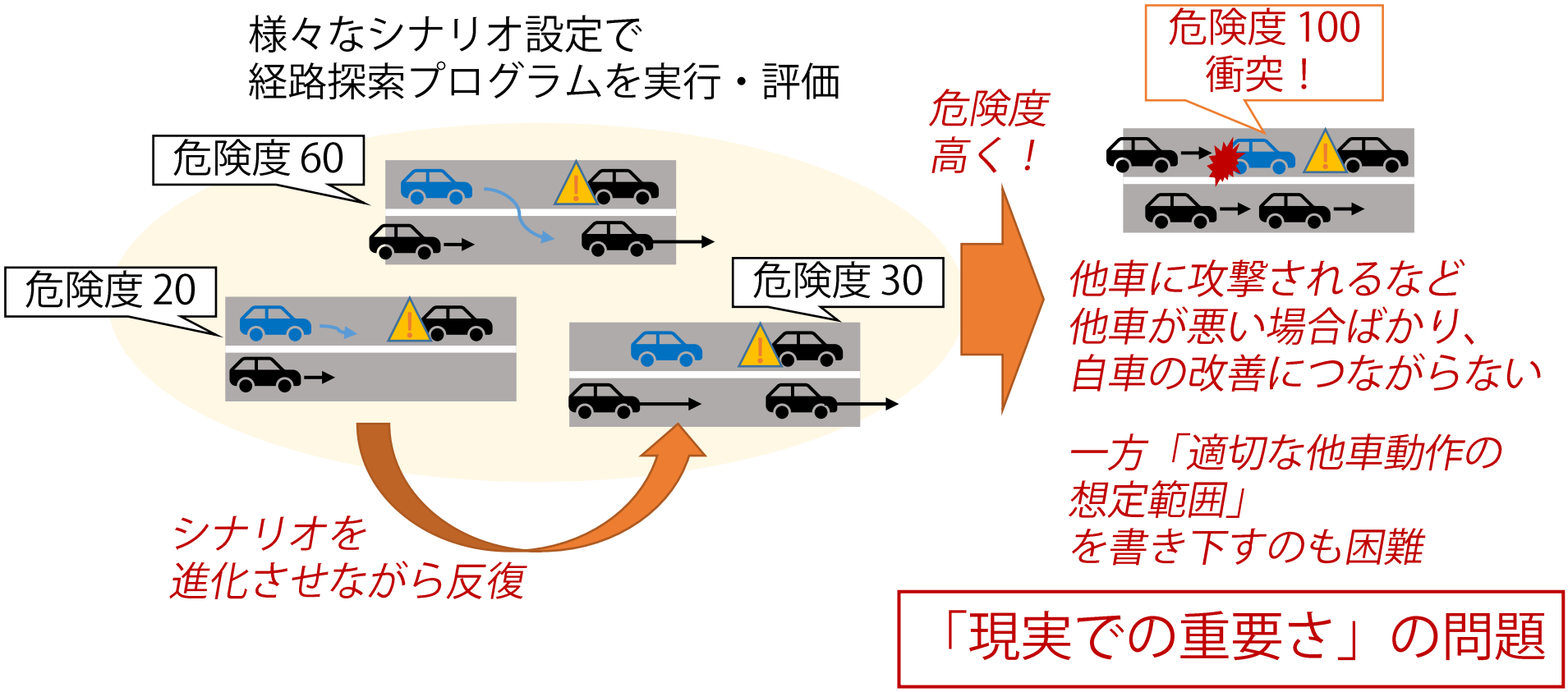
Search-based Testingしてみると…



「求める」結果になるシナリオを探索
基本：危険度がより高い動作が発生するシナリオがよい
応用：さらに今回の意図「停止車両追い抜き」に、より合致するとよい

研究事例（１）：衝突発生シナリオの検出

Search-based Testingしてみると…



「現実での重要さ」の問題

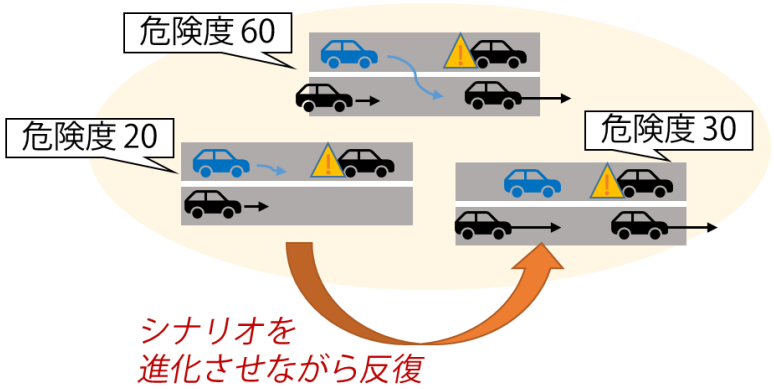
「求める」結果になるシナリオを探索
基本：危険度がより高い動作が発生するシナリオがよい
応用：さらに今回の意図「停止車両追い抜き」に、より合致するとよい

研究事例（１）：衝突発生シナリオの検出

■提案手法

■シンプルなアイディアで有用な結果に！

様々なシナリオ設定で
経路探索プログラムを実行・評価



※「停止車両追い抜き」といった
シナリオ設定ごとに

マツダ株式会社に
いただいた事例で評価

新たなスコアを追加：
危険を避ける自車動作修正が容易か？
（自動修正案を試しに作ってみる）

自車の責任での衝突,
つまり対処すべき問題点を示す
シナリオを検出可能に！

[Calo et al., Generating Avoidable Collision Scenarios for Testing Autonomous Driving Systems, 2020]

研究事例（２）：危険の要因分析

- たくさんテストは実行することになるが…
  - シミュレーションでも実機でも

膨大な
設計の選択肢

膨大な
動作環境の可能性

やや危険なケースも，
とても安全なケースも
いろいろあるけど…

シミュレーション
データ

知識になっていない！

安全性を向上したいが
何が発生した危険の原因
だったのだろう？

避けるべき設計や，綿密に
調べるべき動作環境は？

研究事例（２）：危険の要因分析

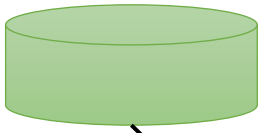
■提案手法

従来プログラムを対象とした
バグ箇所予測の技術

発展

離散のための技術
→ 連続世界を
「おおまかにとらえ」適合

シミュレーション
データ



x	y	z	…	危険度
0.3	0.8	0.2	…	0.7
0.5	0.1	0.3	…	0.2
0.2	0.3	0.6	…	0.4
0.1	0.2	0.1	…	0.3
…	…	…	…	…

分析１

単体要因の
危険への影響度把握

「Ｘが高めの場合に
危険度が高くなる」

分析２

危険につながる
複合要因の検出

「Ｙが高めでＺが低めの
場合に危険度が高くなる」

マツダ株式会社に
いただいた事例で評価

- 避けるべき設計の発見
- 綿密に検査すべき動作環境の同定
- 発生した危険の原因追及
- 期待と実態のずれの発見

[Zhang et al., Assessing the Relation Between Hazards and Variability in Automotive Systems, 2019]

他の研究事例

- Search-based 安定性分析・敏感性分析
  - 設計や環境がごくわずかに変わるだけで，
    安全性が大きく変わるような状況を検出 [Lee et al., GECCO'19]
- 経路計画プログラムに対する「テストのテスト」
  - 「快適さを極端に下げる」など「不具合」を意図的に
    入れて「気づくようなテストが揃っているか」を評価
- AI技術の活用 [Laurent et al., APSEC'19]
  - 「AIゲームプレーヤー」の技術（強化学習など）
- 様々なシステムでの「問題検出」 [Kato et al., PRDC'19]
  - 社会シミュレーション，ゲーム，…（各企業と）
- …

おまけ：理論側の仕事

- ヒューリスティックの塊…
- 理論側は同じ問題を受け取って…
  - 安全性そのものの証明は困難
  - 「この部分は調べなくてよい」につながる
    性質（例えば単調増加性）の証明ならイケる？

➡ Relational Differential Dynamic Logic

単純な例：
「等加速度運動では，加速度が
大きいほど速度は大きくなる」

$$0 = x = x^\sharp \land 0 < v = v^\sharp \land 0 < a < a^\sharp$$
$$\Longrightarrow \left[\left\langle\!\!\left\langle \frac{\dot{x} = v, \dot{v} = a}{\dot{x^\sharp} = v^\sharp, \dot{v^\sharp} = a^\sharp} \right\rangle\!\!\right\rangle x = x^\sharp \right] v \leq v^\sharp$$

➡ 探索によるテストを加速！

[Kolčák, et al., Relational Differential Dynamic Logic, TACAS' 19]

ここまでのまとめ

「プログラムを作るのがプログラミング研究」で止まらず閉じずに

「問題解決」の研究を一緒にしませんか？

- 「計算機に指示を直接出すこと」にとらわれない
- 「世界のあらゆるモノ・コトを動かすこと」
  （「やりたいこと」「世界に仮定すること」も扱うはず）
- ヒューリスティックも使っていっていいじゃない！
  （演繹的アプローチと両輪になるのが最強）

世界にはPPLの皆さんなら「もっとうまくできる」ことがたくさん転がっている！！

AI・機械学習システム

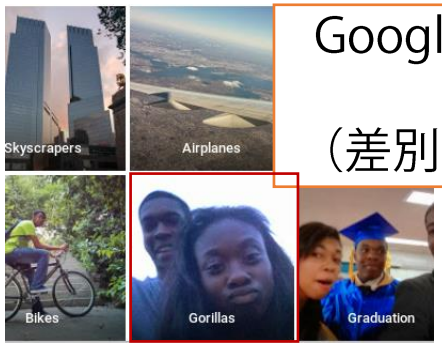
（簡易版）

AI・機械学習システムの課題（一部）

- 産業応用に向かった以上，「動けばよい」
  「当たる確率が高ければよい」では済まない

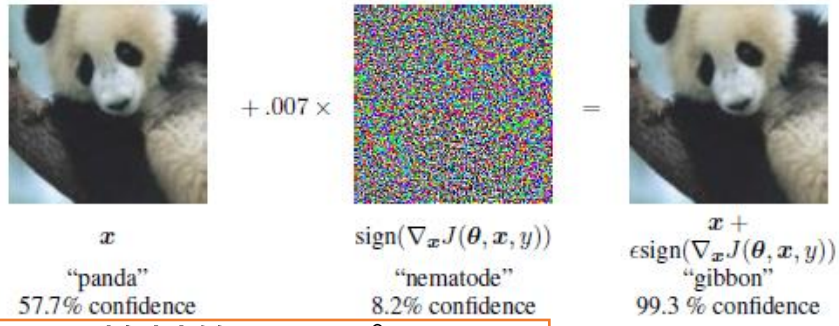
自動運転（認知部分）
というSafety-Criticalな応用

[http://www.dailymail.co.uk/news/article-3677101/Tesla-told-regulators-fatal-Autopilot-crash-nine-days-happened.html]



Googleが直せなかった
技術限界
（差別呼ばわりされる）

[https://www.theguardian.com/technology/2018/jan/12/google-racism-ban-gorilla-black-people]



敵対的サンプル
（ノイズでの結果変化）

[Goodfellow et al., Explaining and Harnessing Adversarial Examples, 2015]

機械学習*と公平性に関する声明

2019年12月10日
人工知能学会 倫理委員会
日本ソフトウェア科学会 機械学習工学研究会
電子情報通信学会 情報論的学習理論と機械学習研究会

倫理的要請
その裏の技術的課題

[http://ai-elsi.org/archives/898]

MLSE研究会 メルシー

https://sites.google.com/view/sig-mlse/

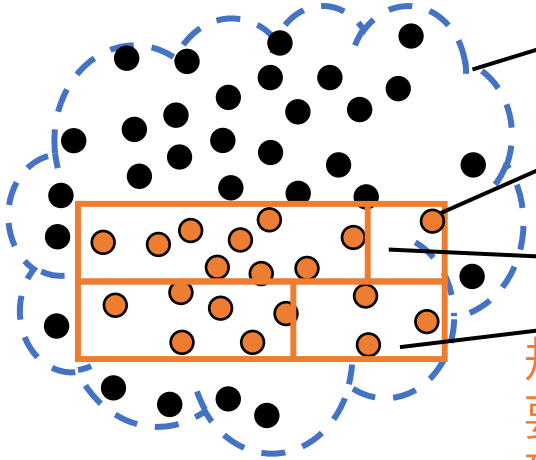
- 日本ソフトウェア科学会 機械学習工学研究会
  - 2018年4月 正式立ち上げ
  - 研究会なので，基本は「場」の提供に専念
- この2年で30回近くのイベント
  - 他イベント内のセッションも多い（分野横断的）
  - 参加者は企業の方々が中心
    （学界からの講演会や国際会議の輪講でも）
  - 焦点を絞ったいくつかのWGが進行中
- まだまだこれから
  - 学界からの貢献，異なる分野からの技術の連携・融合，現場経験に基づいた取り組み強化が必要

「機械学習の技術」→「機械学習工学の技術」

- ソフトウェア工学領域から考えると（ごく一部）
  - 要求工学： ユーザや発注者の「ゴール」を引き出し（ともに創り），今回作るシステムの要求（範囲や制約）を定義，合意，記述し，その妥当性確認をする
  - 設計： 保守性など内部品質も含め様々な品質を実現するための適切なアーキテクチャや実現手段を定める
  - テスティング・検証・品質保証： 不具合（バグ）に代表される様々な品質の問題を検出・修正し，必要な品質が実現されていることに一定の確信を持てるようにする
  - 運用，保守（適応・進化），管理，プロセス，チーム構成，日々のコミュニケーション，・・・

従来ソフトウェアと機械学習（演繹と帰納）

■従来ソフトウェア



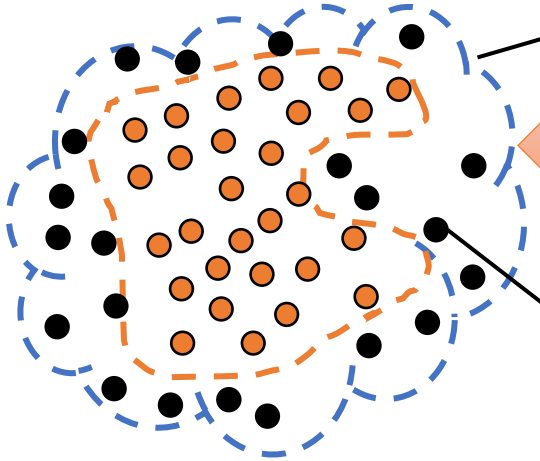
ゴール：名医のように健康状況を適切に判定する

個別事例：Aさんの健康診断データは・・・
→「糖尿病疑い，他問題なし」と判定すべき

状況に応じルール化された要求仕様：
- HbA1c値がｘ以上の場合…
- 面積を基にして…というアルゴリズムで
  レントゲンの影の有無を判定し…

規定された
要求範囲だけ
確実に充足

■機械学習型AI



ゴール：名医のように健康状況を適切に判定する

正解率 90%

ゴール全体を
大まかに達成

データ

過去の
判定結果

データを基に訓練して実現した機能

個別事例：Aさん（自明な例でも誤った判定が起きうる）

高度な認識や判断が可能になる一方で思わぬ誤りも

別の言葉： Software 2.0

- 成果物は，大量のパラメータ値
  - 「ソフトウェアの書き方（作り方）」が変わっている

Medium

Andrej Karpathy Follow

Director of AI at Tesla. Previously Research Scientist at OpenAI and PhD student at Stanford. I like to train deep neural nets on large datasets.

Nov 11, 2017 · 8 min read

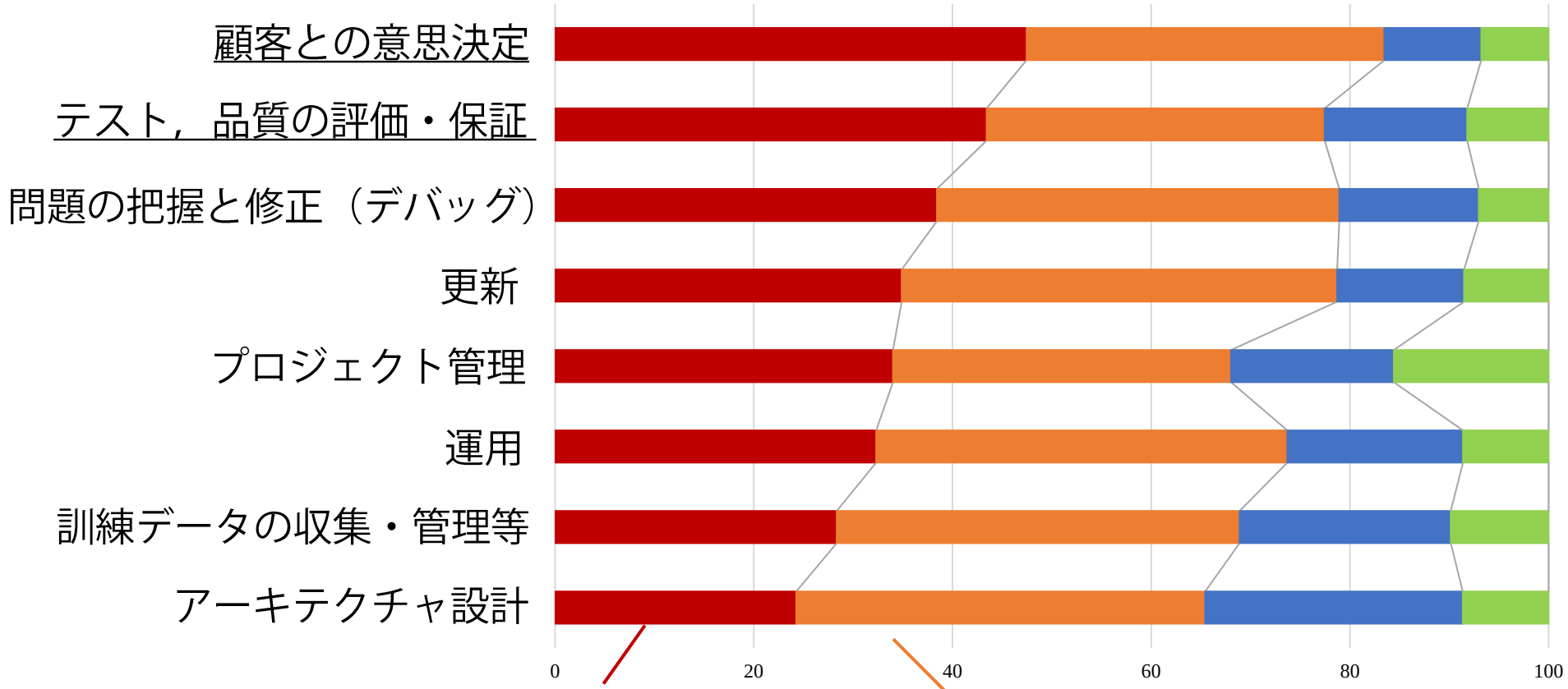
Software 2.0

I sometimes see people refer to neural networks as just “another tool in your machine learning toolbox”. They have some pros and cons, they work here or there, and sometimes you can use them to win Kaggle competitions. Unfortunately, this interpretation completely misses the forest for the trees. Neural networks are not just another classifier, they represent the beginning of a fundamental shift in how we write software. They are Software 2.0.

[https://medium.com/@karpathy/software-2-0-a64152b37c35]

2018年のアンケートより（MLSE）

- 280名弱のアンケート対象者，大半は開発者
  （ソフトウェア開発経験豊富，機械学習には新規参入）



根本的に異なる新たな考え方が必要

考え方は同じだが
手法／ツールが未成熟

[https://sites.google.com/view/sig-mlse/] → 「発行文献」

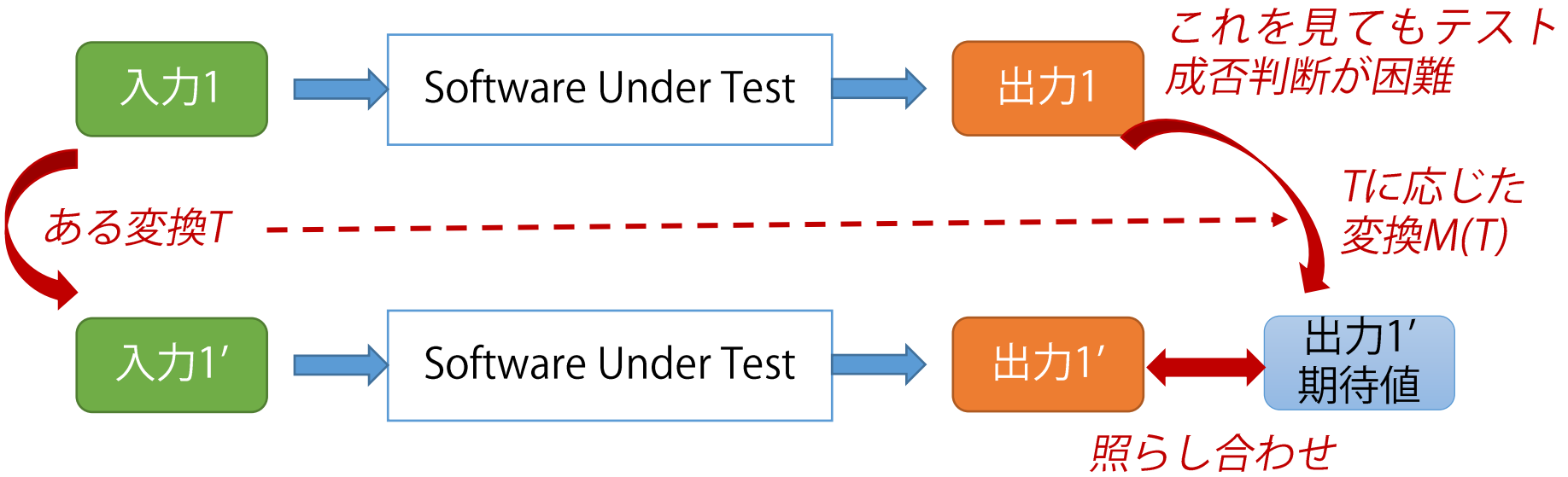
例：テストの課題

- そもそも「テスト不可能」[Weyuker, On Testing Non-Testable Programs, 1982]
  - 画像分類など，正解を与えることのコストが大きい
  - 給与判断など，唯一の正解を決めがたい場合もある
  - 推薦やデータマイニングなどの場合，未知の正解を求めることが目的で，出力の期待値は存在しない
- 「テストでバグを見つける」？
  - 分類や予測が100%成功することは本来ないため，「Failならバグの存在を示唆」ということにはならない
- 単体テスト？
  - 実装は大きなブラックボックスであり，「一部分だけテストしてデバッグを容易にする」ことができない

背景技術：メタモルフィックテスティング

- メタモルフィックテスティング
  - 多数の入力を用いてテストをしたくても，各出力の正しさ（テスト成否）を判定するのが困難または高コスト

➡ 「入力を変えると出力はこう変わるはず」という関係を検証，既存テストケースから多数のテストケースを生成



例： $\sin(x) = \sin(\pi - x)$

[Segura et al., A Survey on Metamorphic Testing, 2016]

テスト技術の代表例（初期）

- 機械学習モデルの「あら探し」テスティング
  - 既存画像に雨や画像回転などのノイズを加えた様々な入力を探り，「よい」テストスイートを見つける（サーチベースドテスティング）
  - 目的１：「ニューロンカバレッジ」を最大化することで，多様な振る舞いを促すようにする
  - 目的２：「悪いケース」を検出する
    - 他バージョンとの比較
    - 入力へのノイズ追加でハンドル角が大きく変わるケース（メタモルフィックテスティング）

1.1 original

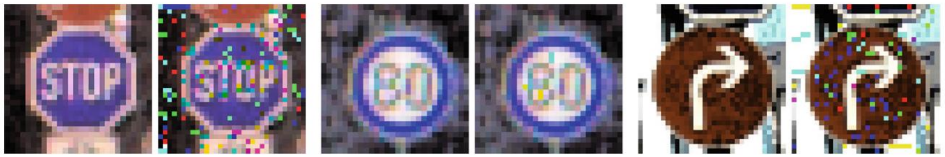
1.2 with added rain

[Pei et al., DeepXplore: Automated Whitebox Testing of Deep Learning Systems, 2017]

[Tian et al., DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars, 2018]

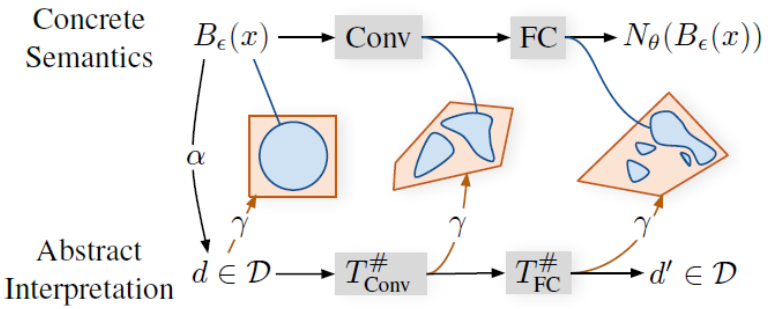
形式検証技術の適用

- 形式検証技術による頑健性検証
  - 画像認識において，「一定範囲の画像操作」を行っても認識結果が変わらないかどうかを網羅的に検証
  - SMTソルバーや抽象解釈を応用



"stop" to "30m speed limit"　"80m speed limit" to "30m speed limit"　"go right" to "go straight"

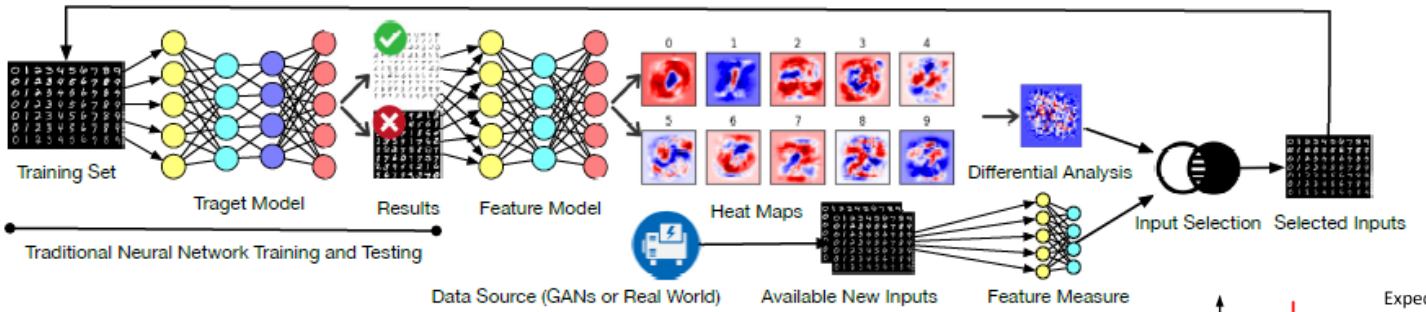
[Huang et al., Safety Verification of Deep Neural Networks, 2017]



[Mirman et al., Differentiable Abstract Interpretation for Provably Robust Neural Networks, 2018]

デバッグ技術の例

- 識別成功データと失敗データの特徴量を比較し再訓練に用いるデータを選択
  - その他，DNNのどのレイヤで過学習・未学習の影響が現れるかを調べそこを重点的に修正



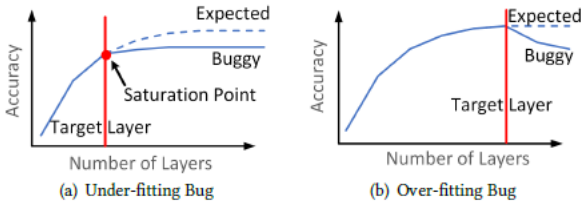


Figure 7: Test Accuracy v.s. Number of Layers

[Ma et al., MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection, 2018]

最近の論文例（１）

- 形式検証や安全性保証
  - SMTソルバを用いた形式検証 [CAV'17]
  - 抽象解釈を用いた検証 [ICML'17, NIPS'18]
  - 安全性制約を満たす強化学習 [AAAI'18]
- 「問題」があるケースの探索を行うテスティング
  - ニューロンカバレッジを用いたサーチベースドテスティング [SOSP'17] [ICSE'18]
  - 確率的ゲームによる検証 [TACAS'18]
  - 細粒度のニューロンカバレッジとそれを用いたサーチベースドテスティング [ASE'18]
  - 公平性のテスティング [ASE'18]
  - 「驚き」に基づくテスティング [ICSE'19]
  - ミューテーション（人工バグ）を用いた敵対的サンプル検出 [ICSE'19]
  - ニューロンカバレッジはミスリーディング [ICSE'19]

最近の論文例（２）

- 「バグ」（コーディングミスなど）の発見や調査
  - メタモルフィックテスティングの実証 [ISSTA'18]
  - バグの種類に関する調査 [ISSTA'18]
  - バグの種類に関する調査 [FSE'19]
- テストの評価・修正（再訓練）
  - ミューテーション分析（テストの強さの評価） [ISSRE'18]
  - 過学習・未学習に対するデバッギング [FSE'18]
  - 深層学習ライブラリにおけるバグ位置推定 [ICSE'19]
  - 自動プログラム修正の適合によるDNN修正 [JSSST'19]
- 上記のための道具
  - RNNの状態遷移の近似とその活用 [FSE'19], [AAAI'20]
- 品質全般
  - ソフトウェア品質特性の見直し [ISSRE'19]

JST MIRAI QAML 改め eAI プロジェクト

- “Engineerable AI” のビジョンを掲げ取り組み

  おおよそには，

  - 演繹的アプローチ（要は従来プログラム）との融合により，レアケースを扱えたり，信頼性を保証できたりする新たな機械学習モデルの確立
  - プログラムに対するテスト・検証・自動修正の技術を展開することによる安全性・信頼性技術の確立
  - 自動運転における安全性論証や希少がんの診断へ

（2020年度，強力な新体制にて探索研究実施予定）

今回のまとめ

「プログラムを作るのがプログラミング研究」で止まらず閉じずに

「問題解決」の研究を一緒にしませんか？

- 「計算機に指示を直接出すこと」にとらわれない
- 「世界のあらゆるモノ・コトを動かすこと」
  （「やりたいこと」「世界に仮定すること」も扱うはず）
- ヒューリスティックも使っていっていいじゃない！
  （演繹的アプローチと両輪になるのが最強）
- 機械学習（AI）の安全性・信頼性については，
  演繹・論理の力との融合が必要・有効なはず

世界にはPPLの皆さんなら「もっとうまくできる」ことがたくさん転がっている！！

ありがとうございました！

再：Shameless Advertisement

2つのプロジェクトで連携教員，研究員，
エンジニア，インターン，諸々募集中です！
（JST ERATO-MMSD, JST-MIRAI QAML改め eAI）
お気軽にお問い合わせください！