



DEEP SPECIFICATION MINING

- TIEN-DUY B. LE, DAVID LO.
- PROCEEDINGS OF THE 27TH ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS [ISSTA'18] P106-117.

ABSTRACTION

- 背景: ソフトウェア仕様(specification)の手動生成は高コスト[23]
 - 低成本化のために仕様解析技術(specification mining)が登場
 - モチベ: しかし、既存の技術は品質が伴わない
- 提案手法: **Deep Specification Miner (DSM)**
 - 有限状態オートマトン(FSA)ベース with **RNNLM**
 - ターゲットクラスの仕様をFSAとして出力
- 実験結果: 精度をF値で評価
 - 実験においてF値で**平均71.97%**を達成
 - 比較対象技術と比較して**+28.22%**の精度向上
- 応用先:
 - Sand-box system for Android applications
 - 仕様と異なる動作を検知/報告

[23] Why Are Formal Methods Not Used More Widely? [Fourth NASA Formal Methods Workshop'97]

BACKGROUND

- ソフトウェア仕様書の記述は高コスト[23]
 - ソフトウェア: アプリケーション、ライブラリ
 - ex.) ソフトウェア更新頻度の上昇
 - ex.) 仕様書として文書化するには要スキル
 - にもかかわらず、正確な保守作業には仕様書は必須
 - →自動仕様解析技術(specification mining)の登場
 - k-tails[7] (本論文の比較対象)
 - CONTRACTOR++[24], SEKT[24], TEMI[24] (本論文の比較対象)
 - SpecForge[26] (本論文と同著者)
- FSA/FSM based approach:
 - FSA: finite-state-automaton, FSM: Finite-state-machine
 - specification miningの一形態
 - 提案手法の骨組み
 - メソッド呼び出しシーケンス(≒実行トレース)を解析
 - →仕様を表現するFSAを生成
 - [7][24]もFSA based (program invariantもふまえて導出)

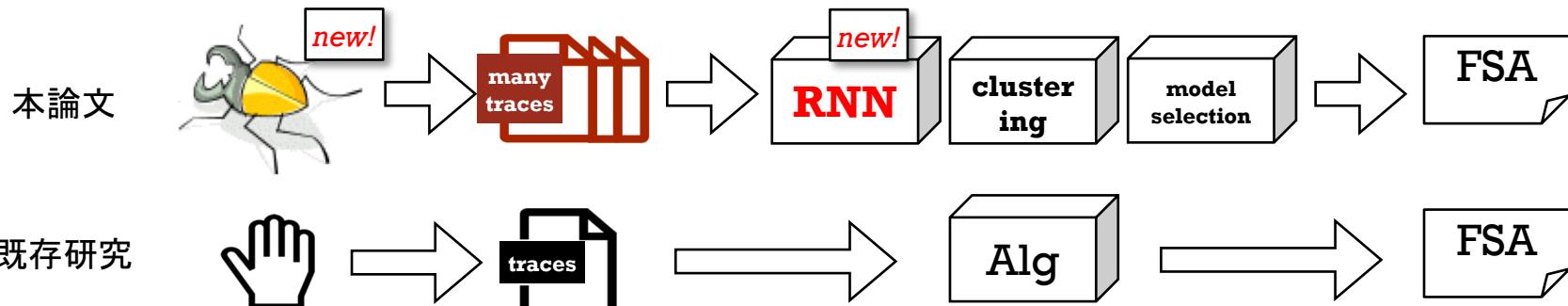
[7] On the synthesis of finite-state machines from samples of their behavior [IEEE Trans. Comput'72]

[24] Automatic mining of specifications from invocation traces and method invariants [ISSTA'14]

[26] Synergizing Specification Miners through Model Fissions and Fusions [ASE'15]

MOTIVATION

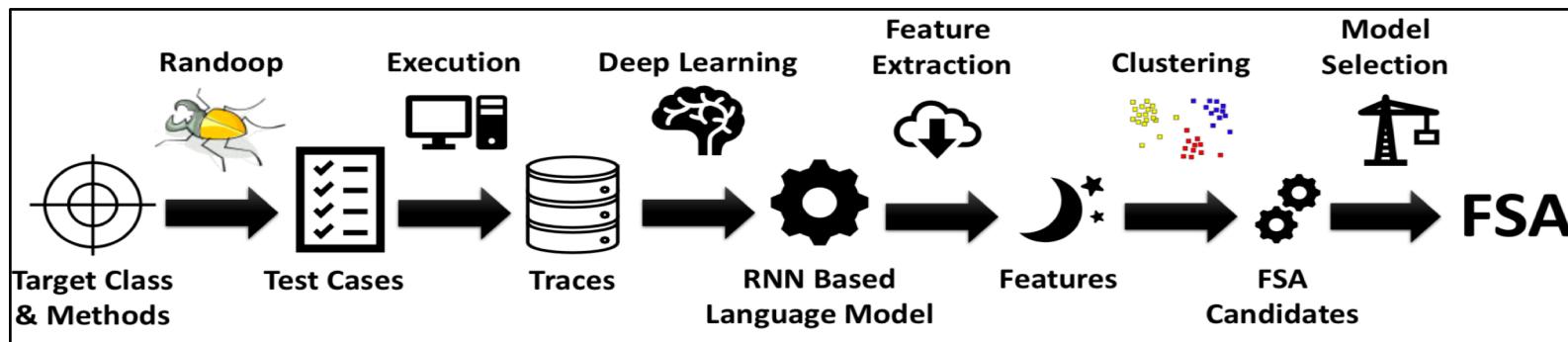
- 既存のFSAによる仕様自動生成では品質が伴わない
 - 未解決の問題有り
 - ex.) 高頻度あるいは低頻度のメソッドを正確に対処できない
 - →汎化に難有り
- **RNNを用いて汎用的なFSAモデルを作成**
 - 入力: メソッド呼び出しを1単語とみなした文のセット
 - →大量の文章(trace)を用意する必要性
 - Randoop[42]を用いた自動テストケースの大量生成



[42] Scaling up automated test generation: Automatically generating maintainable regression unit tests for programs
 [ASE'11] <https://randoop.github.io/randoop/>

APPROACH OVERVIEW

- 提案手法: **Deep Specification Miner (DSM)**
 - 新規性①: 仕様解析に自動テストケース生成とRNNを組み込んだ点
 - 新規性②: RNNで抽出する特徴の定義と、集約アルゴリズムの選択
- 大まかな流れ
 - Randoop[42]によるテストケースの大量生成
 - テストケースの実行結果をRNNに入力/学習 & PTAに変換(後述)
 - [学習済みモデルとPTA] → [特徴] → [候補FSAs] → [FSA]
 - [特徴] → [候補] → [FSA]は既存技術やヒューリスティクスを使用

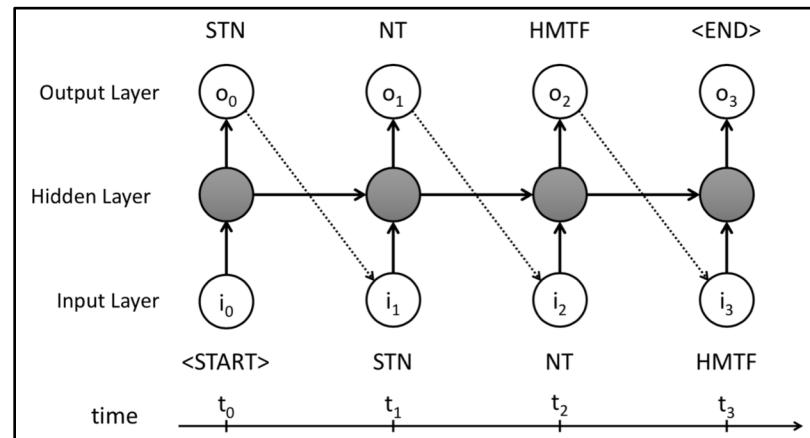


APPROACH

DEEP LEARNING ARCHITECTURE

- 今回採用された学習アーキテクチャはLSTM[20]
 - 1 sentence : テスト1回の実行(1 trace)
 - 1 word : 各メソッド呼び出し
- テスト実行 → <START>と<END>に挟まれた文(trace)を生成
 - ex.) $S_i = (<\text{START}>, \text{method_1}, \text{method_2}, \dots, <\text{END}>)$
 - この文を教材として使用
- 使用用途:
 - <START>からある状態Sまでのメソッド列(単語列)から次のメソッドを確率論的に予測するRNNLMを構築
 - → 予測値を特徴として抽出

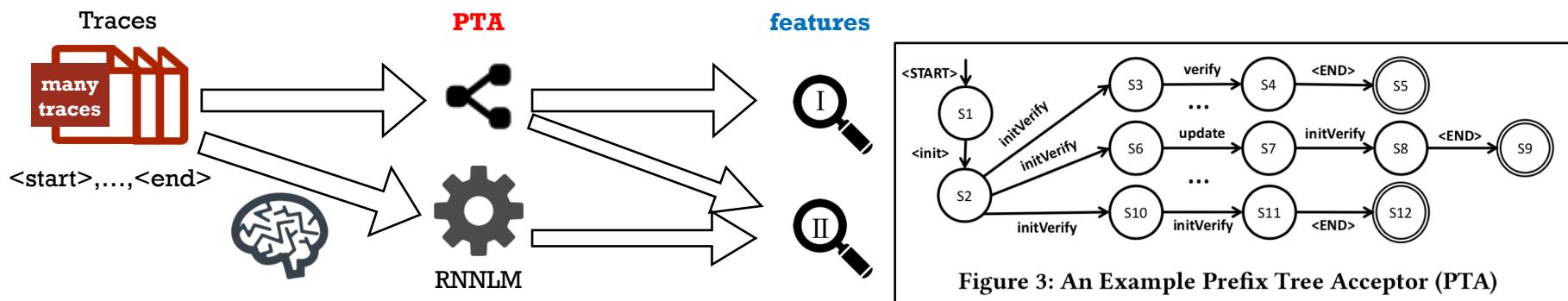
[20] Long short-term memory [Neural computation'97]



APPROACH

PREFIX TREE ACCEPTOR (PTA) & FEATURES

- 実行記録から(RNNLMとは別に)PTAを生成
 - PTA: Prefix Tree Acceptor**
 - <start>を根、<end>を葉とした実行記録を集約した木構造
 - DFAの性質を持ち、PTAから区別可能な1 traceを抽出可能 (= 全traceを受理可能)

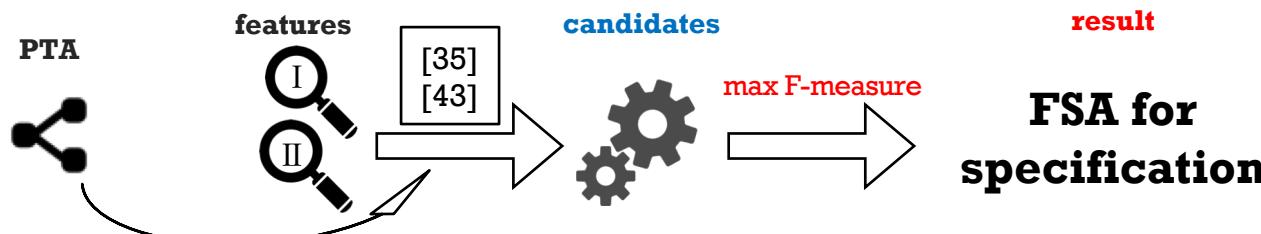


- PTAとRNNLMから特徴を抽出
 - Feature I**: PTA内の状態 S_i までのメソッドを1、 S_i からのメソッドを0とする特徴量
 - ex.) PTA図のS3: $F_1(S3) = (<\text{start}> == \text{init} == \text{initVerify} = 1, \text{Verify} == \text{end} == 0)$
 - Feature II**: 状態 S_i が与えられたとき、RNNLMが output する次のメソッドの予測値
 - ex.) PTA図のS3: $F_2(S3) = (P_{\text{Vrfy}}=P_{\text{e}}=0.4, P_{\text{s}}=P_{\text{i}}=P_{\text{iVrfy}}=P_{\text{update}}=0.15)$

APPROACH

CLUSTERING & MODEL SELECTION

- 得られた特徴(features)を元にPTAを[35][43]でクラスタリング
 - k-means[35](k-平均法)
 - hierarchical clustering[43](階層型クラスタリング)
- 出力は複数のFSA候補 (FSA Candidates)
- 各候補のF値を計算し、最大のF値を持つFSAを仕様として選出
 - ここでF値は実験での精度とは異なる
 - $F - measure_{FSA} = 2 * \frac{Precision_{FSA} * Recall_{FSA}}{Precision_{FSA} + Recall_{FSA}}$
 - $Precision_{FSA} = \frac{|P_{FSA}|}{|P_{FSA} \cup P_{Traces}|}$ P: 順序関係のあるメソッドペア(m1,m2)の集合
 - $Recall_{FSA} = \frac{|Accept_{Traces}|}{|All_{Traces}|}$ → 候補FSAが受理できたテストケースの割合



[35] Some methods for classification and analysis of multivariate observations [Fifth Berkeley Symp. '67]

[43] Clustering Methods. In The Data Mining and Knowledge Discovery Handbook [2005. p321-352]

EXPERIMENT OVERVIEW

- 実験評価項目:
 - DSM単体の精度評価
 - 9 library classes in JDK
 - 1 library class from Daikon[14]
 - 1 library class from Apache Xalan
 - 既存研究との精度比較
 - k-tails[7]
 - CONSTRUCTOR++[24]
 - SEKT[24], TEMI[24]
 - RNNアーキテクチャの精度比較
 - LSTM[20] (default)
 - GRU[9]
 - Standard RNN
- 正解セット:
 - ground truth model
 - [24]で生成されたものを使用
 - ドキュメントから微調整済み
- 評価指標:
 - Precision
 - Recall
 - F値
 - アプローチ内のF値とは別

[9] Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [CoRR'14]

[14] The Daikon system for dynamic detection of likely invariants [Sci. Comput. Program. '07]

EXPERIMENT

RESULT (DSM VS PREVIOUS WORKS)

T1: 1-tails[7]
 T2: 2-tails[7]
 C+: CONTRACTOR++[24]
 S1: SECT 1-tails[24]
 S2: SECT 2-tails[24]
 OT: Optimized TEMI[24]

DSM

Class	F (%)	Class	F (%)
ArrayList	22.21	Signature	100.00
HashMap	86.71	Socket	54.24
HashSet	76.84	StackAr	74.38
Hashtable	79.92	StringTokenizer	100.00
LinkedList	30.98	ZipOutputStream	88.82
NFST	77.52%	Average	71.97

previous works

Target Library Class	Scheme II					
	T1	T2	C+	S1	S2	OT
ArrayList	5.61	3.08	36.03	4.34	3.08	35.82
HashMap	37.35	21.93	68.94	37.35	21.93	68.94
HashSet	22.96	15.35	52.22	22.96	15.35	55.62
Hashtable	78.42	73.37	92.78	81.69	65.47	92.78
LinkedList	26.03	8.07	86.02	18.45	6.59	86.02
NFST	68.72	51.04	30.40	66.58	49.51	33.40
Signature	100	95.41	66.88	95.41	89.78	66.88
Socket	37.30	21.98	55.15	35.32	20.87	55.62
StackAr	42.57	42.57	34.91	42.57	42.57	-
StringTokenizer	100	89.69	21.30	92.21	84.77	0.00
ZipOutputStream	82.51	78.08	62.08	86.47	67.84	66.20
Average	54.70	45.50	55.22	53.03	42.52	56.13

- DSM: F値平均71.97%を達成
 - Signature, StringTokenizerでは100%を達成
- vs Prev works: 平均値最大はOptimized TEMI(56.13%)
 - scheme II : Randoop[42]によるテストケース生成 (I はDaikon)
 - Daikon[14]: invariantの動的検出ツール
 - →平均値比較で+28.22%の精度改善

EXPERIMENT

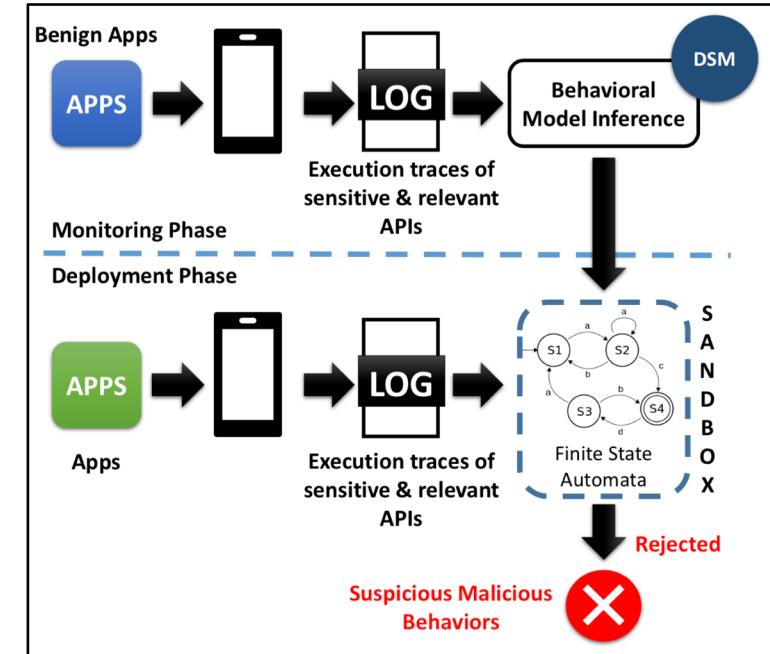
RESULT (LSTM VS OTHER ARCHITECTURE)

Target Library Class	DSM ^{LSTM}	DSM ^{RNN}	DSM ^{GRU}
ArrayList	22.21	4.95	9.39
HashMap	86.71	97.76	100.00
HashSet	76.84	68.86	73.88
Hashtable	79.92	87.94	87.94
LinkedList	30.98	32.29	34.86
NFST	77.52	70.09	73.31
Signature	100.00	65.31	95.41
Socket	54.24	51.31	58.34
StackAr	74.38	71.79	77.67
StringTokenizer	100.00	95.88	100.00
ZipOutputStream	88.82	87.36	73.29
Average	71.97	66.69	71.28

- 結論:
 - LSTM[20]とGRU[9]はほぼ精度差なし
 - Standard RNNは若干精度が落ちる
 - (原因については論文内で特に言及はなし)
 - 単純に考えると勾配消失/爆発が起きて極高/低頻度メソッドの学習が不正確になったか?

APPLICATION SAND-BOX FOR ANDROID

- 機密APIメソッドを対象に仕様を定義
 - 機密APIメソッド:
 - 他資源(写真など)へアクセスするメソッド
 - Monitoring phase
- 仕様外の挙動をした場合ユーザへ警告
 - 誤検出もあるためプロセス停止はなし
 - Deployment phase
- 既存研究との比較:
 - 誤検出(FP)や見逃し(FN)を抑制
 - vs k-tails[7]
 - 同程度のFNでFPを抑制
 - vs Boxmate[21]
 - 若干のFPを犠牲にFNを抑制



APR	TP	FN	TN	FP	TPR	FPR
DSM	93	9	398	112	91.18%	21.97%
k-tails	94	8	350	160	92.16%	31.37%
Boxmate	77	25	421	89	75.49%	17.45%

感想

- 自身の研究でもメソッドシーケンスを文と学習させるアイデア有
 - 実際に実験してくれているのはありがたし
 - 目標としてはどうにかオンライン解析にも適用したい
 - 本論文はオフライン解析
- 論文自体は図も多く読みやすかった
 - 比較対象も複数あるのは相対的でGJ
 - 強いて言うと学習やテストケース生成にかかった時間も知りたかった
- 結構な数の既存技術を使って組み立てた印象
 - Randoop, LSTM, k-means, ...
 - オリジナルのアルゴリズムは多くないが多工程
- スライド用素材:
 - 本論文(Deep Specification Mining [ISSTA'18])
 - <http://icooon-mono.com>
 - <https://www.silhouette-illust.com>