A-002　文法推論に基づくシステム仕様からのモデル生成高速化手法の検討

金子　舟 †　　千代　英一郎 †

† 成蹊大学理工学部情報科学科

1　はじめに

モデル検査とは，システムの振る舞いを分析する手法のひとつである．これは様々な要素で構成されたシステムを対象とし，その振る舞いを反映したモデルを用いて，デッドロック等の不具合が含まれていないかを分析する．テストとは異なり，システムの取り得る状態やイベントすべてに対して網羅的に検査出来ることが特徴で，高信頼システムの実現手法のひとつとして注目されている．

モデル検査を行うには検査対象となるシステムの振る舞いを正しく表現したモデルが必要となるが，このモデルの作成は現在のところ人手によって行われている部分が多く，技巧的な難しさと相まって実利用の妨げになっている．

この問題に対し我々は，システムの正しい振る舞いを表す具体的な遷移例（以下，イベント列とする）からプロセス式によるモデルを学習により自動生成する手法について検討している．

本アプローチでは，学習の部分に文法推論 [1] を用い，与えられたイベント列を一般化してモデルの生成を行う．文法推論とは，ある文法に従う正しい文（正例）や，従わない文（負例）が与えられるとき，それらの文を基にして文が従っている文法を推定することである．ここでは，2 章に示す RPNI と呼ばれるアルゴリズムを使用する．

本アプローチによるモデル生成の流れについて説明する．予め，システムの仕様を満たす正しい振る舞いを持ったいくつかのイベント列を用意する．これらイベント列を例文として文法推論を行い，一般化することでモデルを生成する．続いて，生成されたモデルの表現する振る舞いが，システムの仕様に一致するかを照合する．仕様を満たさない場合や表現に不足する部分が存在する場合は，これらを補うイベント列を例文に加えることで，容易に修正が可能である．このようなサイクルによってシステムの振る舞いを正しく表現したモデルの生成が行われる．

我々は本アプローチにもとづき対話的にモデル生成を行うためのツールを試作し，信号機システムなどの基本的なシステム仕様を例としてその有用性を検証している．その結果，実用にあたっては，与える例文数が多くなったときの記述の手間，文法推論の処理時間，および出力表示の可読性が対話的なモデル生成を円滑に行う上で問題になることが判明した．これらの問題点のうち本研究では，文法推論の処理時間に関する問題に焦点を当て，実装時の結果から処理に時間を要している部分についての分析を行い，後述する提案手法を用いて高速化，処理時間の短縮を図ることを目的とする．

2　RPNI アルゴリズム

RPNI(Regular Positive Negative Inference)[3] とは，Oncina, Garcia によって考案された状態統合を用いて正則言語を極限学習する手法の 1 つである．ここでは以下の例をもとにアルゴリズムの処理の流れを簡単に説明する．

まず，与えられた例文のうち，正例から PTA(Prefix Tree Accepter) を生成する．これは，木構造上のエッジにイベントが対応した構造となっている．この PTA について，状態同士の統合（マージ）を行いながら一般化を行う．

ここで対象とするシステムの条件は次に挙げるものとする．

条件 1 イベント a から始まり b を経て c で終了する．
条件 2 イベント a, c は各 1 回しか発生しない．
条件 3 イベント b は 0 回以上発生する．

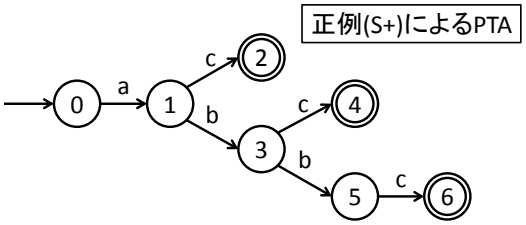

図 1　PTA の作成

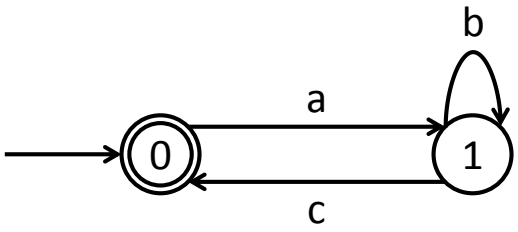

図 2　一般化後の PTA

条件より正例（S+）を設定するが条件 3. を満たすイベント列は無限に存在するため，この例では S+ = [a c, a b c, a b b c] のように考える．また，負例（S-）を S- = [a, c] とする．設定された正例から，図 1 に示すような PTA が作成される．このとき，初期状態:0，輪郭が 2 重線で表される状態が受理状態である．この PTA の根に近い方から順に 2 つの状態を選択してマージを行う．選択される 2 つの状態は，一方が他の状態とのマージが不可能な状態集合の要素，他方をその状態集合に隣接する状態の集合の要素とする．便宜のためにそれぞれの集合について前者を RED，後者を BLUE と呼ぶ．

このとき BLUE のある要素が RED の各要素とそれぞれマージできるかを考える．マージ後の PTA が負例を受理しない場合，マージは成功であり PTA をマージ後のものに置き換える．マージ後の PTA が負例を受理する場合，マージは失敗であり選択した BLUE の要素を取り除いて RED の要素に加える．また，どちらの場合も処理後には RED に含まれる状態に隣接する状態を新たに BLUE へ加え，次の 2 つの状態を選択してマージを行う．こうした処理を BLUE の要素がなくなるまで繰り返すことで一般化された PTA（図 2）を得ることが出来る．

ここで例文数が増加した場合について考える．正例は PTA によって表現されるため，共通する接頭辞の数に応じて少ないデータで表現出来るほか，マージが成功すれば状態数は減少していく．一方，負例のデータ構造に関してはアルゴリズム上での指定が存在せず，重複部分の多い例文が設定されたとき冗長度が上昇し，処理回数が大きく増加することが懸念される．

3　問題点の分析

2 章に示した RPNI アルゴリズムを Java によって実装し，4 章で後述する環境下においてテストデータを与えたときの CPU 使用状況をプロファイラ（hprof）によって分析した．プロファイラ出力のうち上位 5 メソッドを表 1 に示す．また，プロファイラを起動しないときの実行時間は 5 回の中央値をとって約 20.16[s] であった．表 1 ではメソッドに重複が発生しているが，これは呼出し元が異なる trace を区別してカウントするためである．

プロファイラ出力とその trace の詳細から分析したところ，表 1 に示した上位 4 メソッドが関数 transit 自体，もしくはその関数内で呼出される関数であることが分かった．

Efficient model generation from system specifications based on grammatical inference

†Shu Kaneko　†Eiichiro Chishiro

†Science and Technology, Seikei University

表 1　RPNI 実行時の CPU 使用時間

rank	self(%)	accum(%)	count	trace	method
1	37.89	37.89	770	300067	RPNItest1.transit
2	21.26	59.15	432	300065	RPNItest1.transit
3	19.29	78.44	392	300070	Tuple.TFdefined
4	10.09	88.53	205	300066	Tuple.delta
5	7.68	96.21	156	300060	java.lang.Object.hashCode

この関数 transit は PTA が，あるイベント列を受理するか否かの判定を行うもので，ここでは負例受理判定に用いている．これらの処理が CPU 使用時間中の約 89% を占めていた．これは，分析のために与えたテストデータのうち負例の例文数が正例に対して比較的多く，また冗長度が高いことに起因するものと考えられる．

4　提案手法

3 章での結果から，正例数に対して負例数が多いテストデータを用いた場合の，負例の受理判定に関するメソッドを中心に処理時間の短縮を検討する．

4.1　負例に対する PTA の適用

負例を保持するデータ構造に木構造を用いることで，負例の各イベント列間に存在する共通なイベント列を集約する．本アプローチでは正例を保持するために既に実装済みである PTA を活用する．この手法では，判定処理は集約されたイベント列とその差分のイベント列のみとなるため，初期状態から遷移し判定する場合と比較して関係する関数の呼出し回数を減らすことができる．この手法は負例におけるイベント列の冗長度が高い場合，特に有効な手法であると考えられる．

4.2　並列処理の利用

負例受理判定中に対象の PTA が変更されることはなく，負例の選択に順序は関係しない．このことを利用して判定処理の並列化を行う．予め設定された任意のスレッド数に対し，負例の集合をこの数に応じて均等に分割しておき，スレッドにそれぞれを割り当てて初期状態から処理を行う．4.1 の手法では負例の冗長度によって有効性が変化する可能性があるが，この手法では負例の冗長度によらず処理速度の向上を図ることが期待される．

5　評価

提案手法による処理時間の変化を確認するため，同一のテストデータを用いた場合の実行時間を計測した．テストデータは，正例にイベント列のインターリーブを，負例に正例の接頭列へ禁止対を付加したものから作成したイベント数 5，正例数 42，負例数 966，PTA 状態数 196 となるデータを用いた．これをもとに正例数を変化させず，負例数を 100 から 900 の間で変化させることによって，実行時間への影響を確認した．なお，マージが成功すると状態数が減り各データ間での比較が困難になるため，正例に含まれるイベント列を 1 つ負例に追加して対応した．この処理を加えることで，最後に判定される負例は正例の PTA に必ず受理されるため，各マージにおける負例受理の判定回数は常に負例数だけ行われる．並列処理によって負例が分割される場合についても，追加されたイベント列は割り当てられたスレッド内で最後に判定されるよう設定した．

提案手法実装前と，高速化に関する提案手法を施した場合の実行時間を表 2 に示す．なお，計測環境は CPU:AMD Opteron 6386SE 2.8GHz (32 cores)，OS:CentOS 6.4 (2.6.32-358.el6.x86_64)，Memory:256GB である．

実行結果より，提案手法実装前は負例の増加に伴う処理時間の増加がみられたが，提案手法 4.1 ならびに 4.2 の場合は，それぞれほぼ一定の処理時間であった．4.1 の手法の場合，イベント数に対する負例数が大きいテストデータであったため，共通するイベント列を集約化する効果が大きく表れたものと考えられる．4.2 の手法の場合，概ね 2 秒未満であるものの 4.1 の手法より実行に時間を要している．この原因を分析するため，4.2 の手法において 64 スレッド，負例数 900 実行時のプロファイラ出力を検証したところ，当初のプロファイラ出力（表 1）に含まれる関数 transit や，

表 2　実行時間の比較

負例数	提案手法実装前(s)	提案手法4.1(s)	提案手法4.2(s)			
			8スレッド	16スレッド	32スレッド	64スレッド
100	4.26	2.01	2.55	2.99	2.68	3.49
200	6.53	1.98	2.46	2.95	2.80	3.53
300	8.78	2.02	2.44	2.77	2.81	3.39
400	10.91	2.02	2.31	2.88	2.85	3.50
500	12.86	2.04	2.43	3.73	2.77	3.60
600	14.96	2.04	2.23	2.87	2.80	3.56
700	16.79	2.06	2.27	2.93	2.78	3.61
800	18.51	2.06	2.36	2.78	2.80	3.73
900	20.16	2.61	2.41	2.67	2.80	3.70

それらに関連するメソッドが依然約 50% を占めていた．逆説的ではあるが，4.2 の手法の実行時間短縮の大部分が並列化の効果によるものであると言える．

これらの結果より，与えられる例文のうち負例の冗長度が高いテストデータを用いた場合に関してではあるが，負例を保持するデータ構造の木構造化，並びに負例が受理されるか判定する処理の並列化によって RPNI アルゴリズムの高速化を図ることが可能であることが確認された．各手法に共通する課題として，負例の冗長度の変化に対する処理速度の変化，並列化ではさらにオーバーヘッドに関する分析が今後の課題である．

6　関連研究

文法推論の分野では RPNI 以外にも GOLD[2] などのアルゴリズムが発表されている．また，文法推論を利用した研究としてノイズを含む信号列から確率論を用いて状態遷移系を推測する BLUE*[4]，が挙げられるが，本研究と同様のテーマは他に見られない．

7　結論

システムの振る舞いの具体例から文法推論により検査に必要なモデルを得るアプローチにおいて，その実用化に向けて鍵となる RPNI アルゴリズムの高速化手法を検討した．実行時間の分析にもとづきボトルネックとなる処理を特定し，対話的なモデル生成作業を円滑に進める上で望まれるモデル生成時間を達成するための改善方法を設計した．予備評価の結果は有望であり，負例の冗長度が高い条件下ではあるが，処理対象の木構造化，処理の並列化を用いることで処理時間の短縮に寄与できるということがわかった．

今後の課題としては，負例の冗長度の変化に対する処理速度の変化，並列処理におけるオーバーヘッドの分析などがある．

参考文献

[1] de la Higuera, C.: *Grammatical Inference: Learning Automata and Grammars*, Cambridge University Press (2010).

[2] Gold, E. M.: Language identification in the limit, *Information and Control*, Vol. 10, No. 5, pp. 447–474 (1967).

[3] Oncina, J. and Garcia, P.: Identifying regular languages in polynomial time, *Series in Machine Perception and Artificial Intelligence*, Vol. 5, pp. 99–108 (1992).

[4] Sebban, M., Janodet, J.-C. and Tantini, F.: BLUE*: a Blue-Fringe Procedure for Learning DFA with Noisy Data, *Proceedings of Genetic and Evolutionary Computation Conference* (2004).