数理解析研究所講究録
第1849巻 2013年 71-76
2012年度冬のLAシンポジウム [S5]

Java言語による確率時間CEGAR検証器の開発

金沢大学
小池 脩平　長谷川 尭志　清水 隆也　山根 智
Shuhei Koike, Takashi Hasegawa, Takaya Shimizu and Satoshi Yamane,
Kanazawa University

1 導入

1.1 背景

E.M.Clarkeらによって，リアクティブシステムの反例による抽象化精錬の枠組み（CEGAR）[6] のモデル検査 [4] が提案された．モデル検査ではシステムの状態数が大きくなる状態爆発の解決が課題になるが，述語抽象化 [7] を導入し，状態数を抑えながら検証可能な手法がCEGARである．このCEGARを時間確率システムに応用し，確率時間オートマトンの到達可能性解析を可能にした確率時間CEGAR[12] が当研究室の清水らによって提案された．

本稿では，確率時間CEGARを実装し，実験により得られた結果について考察する．開発言語はJavaである．

1.2 確率時間CEGAR

一般に，抽象化を行うとシステムは本来の性質を損なう．CEGAR[6] では，抽象モデル上での反例の候補の導出と，反例を用いた抽象モデルの精錬を，結論が得られるまで繰り返すことで正当性を保証するとともに，偽反例から得た情報により，検証に必要な部分のみを詳細化することで状態数を抑えることができる．確率時間オートマトンに対してCEGARによる枠組みを用いて検証を行うための手法が当研究室の清水らによって確立された [13].

- 検証したい性質を抽象モデルが持っていれば，具体モデルも持っている健全性を保つ抽象化手法
- 抽象モデルから反例の候補を導出でき，その反例の候補が実動作可能な反例であるか解析する反例解析手法
- 反例解析の結果，反例の候補に対応する反例が存在しないとき，同じ反例の候補を生じさせないように抽象モデルを精錬する手法

これにより，確率時間オートマトンを対象としてCEGARを導入することが可能になった．

なお，本稿では上記の詳しい説明は省略する．本稿中の記号はすべて文献 [13] と同様である．

1.3 関連研究

これまでCEGARを適用した検証手法として，リアルタイムシステムを対象とした時間CEGAR[10] や，ハイブリッドシステムを対象とした研究 [5]，確率システムを対象とした確率CEGAR[8]，などが研究されてきた．

本研究では確率システム及びリアルタイムシステムの両方を併せ持つ性質を対象とするため，検証を行う上での課題として，同時実行可能性の解析手法が必要である．

一方，確率リアルタイムシステムに対する検証の既存手法として，記号モデル検査 [9] やPRISM[1] がある．CEGARは，状態数を抑えながら検証可能な手法であり，導入することで効率的な検証を期待できる．本研究では確率時間CEGARを計算機上に実装して，上記手法との性能比較を行う．

2 検証器

本研究で開発した検証器はJavaにより実装を行った．Javaはプラットフォームに依存しないプログラミング言語であり，様々な実行環境で実験を行うことができ，対象となるモデルの検証結果が実行環境依存ではないという結果を得るために非常に有効であった．検証器のソースコードは2000行程度である．

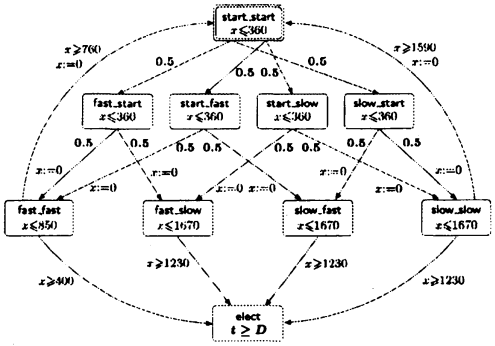

図 1: FireWire Root Contention Protocol

確率時間 CEGAR で取り扱うゾーンは全て凸ゾーンであるため，ゾーンの実装には DBM[2] を用いた．DBM のように，モデルに表れる最大定数を用いてゾーンを表現する場合，反例解析で用いられている前方解析が正しく行えない事が知られているが [3]，以下のいずれか 1 つ以上を満たすモデルであれば，前方解析を正しく行える事も知られている [3].

- $x_1 - x_2 \leq d$ や $x_1 - x_2 < d$ を不変条件や遷移条件に持たない，diagonal-free なモデルである
- クロック変数の数が 3 以下のモデルである

そこで本研究では，クロック変数が 3 以下のモデルのみを対象とする．

3 時間確率システムの検証実験

本章では，確率時間 CEGAR 検証器を Java によって実装し，実験を行って既存手法とその状態数について比較する（3.1 節，3.2 節）．

以下に実験環境を示す．

- Intel Core i3-2120T 2.60GHz
- MemTotal 3980MB
- Windows 7 Home Premium SP1
- Java SE 1.7.0.09

3.1 FireWire Root Contention Protocol

本実験で扱う IEEE 1394 FireWire Root Contention Protocol は，IEEE 1394 FireWire 規格において，2 つの競合するノードから，一方をリーダーとして選出するプロトコルである．本実験ではこのプロトコルに対して，リーダー選出にデッドライン以上の時間を要する場合があるかどうか，という性質について検証を行う．このプロトコルに対する同様の性質についての検証実験は Symbolic Model Checking[9] 及び PRISM[1] によって既に行われており，本実験では同手法との比較を行う．

本実験では，Symbolic Model Checking[9] や PRISM[1] による検証実験で使用されたモデルを一部を変更し，新たに作成したモデルに対して検証実験を行う．そのモデルを，図 1 に示す．新たに作成したモデルでは，リーダー選出が完了したことを示すロケーション *elect* に不変条件 $t \geq D$ を追加している．ここで，t はリーダー選出に要した時間を表現するために，新たに追加したクロック変数である．また，D はデッドラインを表す定数である．従って，*elect* へ到達可能であることは，デッドライン以上の時間を要してリーダー選出を行ってしまう場合があることを意味する．

このような変更を加えるのは，Symbolic Model Checking や PRISM では検証する性質としてデッドラインを指定できるのに対し，本手法はロケーションに対する到達可能性のみを対象としているためである．

なお，既存手法との比較を行うため，デッドラインが 2000 から 60000 までのモデルを用意し，それぞれについて実験を行う．

このように構成したモデルに対し，Symbolic Model Checking と PRISM で検証された性質と等価な到達可能性問題を次のように定める．

$$(\lambda, L_e) = (0, \{elect\})$$

すなわち，

$$\forall A. Prob^A(S_e) \leq 0. S_e = \{(elect, \nu) \in S\}$$

を満たすかどうかについて検証を行う．この検証結果が “No” である場合，すなわち

$$\exists A. Prob^A(S_e) > 0. S_e = \{(elect, \nu) \in S\}$$

表 1: 実験結果: FireWire Root Contention Protocol

D	Loop	Dist	Edge	State	Time[s]
2000	1	12	18	10	0.4
4000	8	24	40	17	2.3
6000	15	38	64	24	4.5
8000	22	52	88	31	8.0
10000	29	66	112	38	13.0
20000	64	136	232	73	98.9
30000	99	206	352	108	399.0
40000	134	276	472	143	1148.3
50000	169	346	592	178	2584.3
60000	204	416	712	213	5093.5

表 2: 状態数の比較: FireWire Root Contention Protocol

D	PTCEGAR	SMC	PRISM
2000	10	15	15
4000	17	25	25
6000	24	47	47
8000	31	81	81
10000	38	126	126
20000	73	528	528
30000	108	1206	1206
40000	143	2168	2168
50000	178	3426	3426
60000	213	4964	4964

が真である場合，リーダー選出にデッドライン以上の時間を要する場合があることが分かる．

表 1 に実験結果を，表 2 及び図 2 に既存手法との比較結果を示す．

全てのデッドラインについて，既存手法よりも少ない状態数で検証を終えている．また，状態数の推移から，デッドラインの小さいうちは効果が少ないものの，デッドラインの増加に伴い，状態数がより削減されていることがわかる．

3.2 CSMA/CD

本実験で扱う CSMA/CD(Carrier Sense Multiple Access/Collision Detection) は，Ethernet における

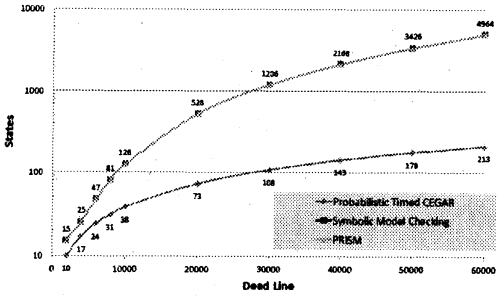

図 2: 状態数の比較: FireWire Root Contention Protocol

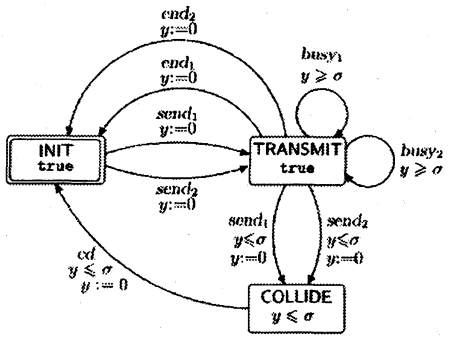

図 3: CSMA/CD: The Medium

基本的な通信プロトコルとして用いられており，次の手順に従って複数のクライアント間で通信を行う．

1. Carrier Sense: 通信を開始する前に，一度受信を試みることで現在通信をしているクライアントが他にあるかどうか確認する
2. Multiple Access: 複数のクライアントは同じ回線を共用し，他者が通信をしていなければ自分の通信を開始する
3. Collision Detection: 複数の通信が同時に行われた場合（コリジョン）はそれを検知し，ランダムな時間待ってから再び送信手順を行う

本実験ではこのプロトコルに対して，各クライアントが通信を完了するのにデッドライン以上の時間を要する場合があるかどうか，という性質について検証を行う．このプロトコルに対する検証実験は 3.1 節と同様に Symbolic Model Checking[9] 及び，PRISM[1] によって既に行われており，本実験では同手法との比較を行う．

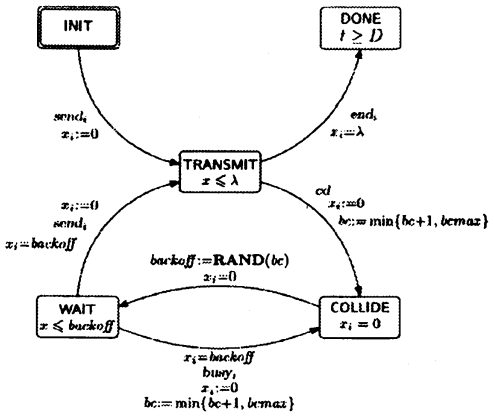

図 4: CSMA/CD: The Stations

本実験で使用するオートマトンを図 3 及び図 4 に示す．図 3 はクライアント間の通信の伝送路の挙動を表現したものであり，図 4 は通信を行うクライアントの挙動を表現したものである．今回の実験に用いる実際のモデルとしては 2 つのクライアント間を伝送路で繋いで通信を行うことを想定しており，これらの 3 つのオートマトンを並列合成したモデルとなる．ここで用いられる $bcmax$ とはコリジョンが発生した場合に行われる再送処理を行う回数の上限である．

このように構成したモデルに対し，検証性質となる到達可能性問題を次のように定める．

$$(\lambda, L_e) = (0, \{DONE\})$$

すなわち，

$$\forall A. Prob^A(S_e) \leq 0. S_e = \{(DONE, \nu) \in S\}$$

を満たすかどうかについて検証を行う．この検証結果が “No” である場合，すなわち

$$\exists A. Prob^A(S_e) > 0. S_e = \{(DONE, \nu) \in S\}$$

が真である場合，各クライアントが通信を完了するのにデッドライン以上の時間を要する場合があることがわかる．

表 3 及び表 4 に実験結果を，表 5，表 6，図 5 及び図 6 に既存手法との比較結果を示す．

$bcmax = 2$ のときについては，デッドラインが 1000 のときを除き、すべてのデッドラインについて既存手法よりも少ない状態数で検証を終えているが，$bcmax = 1$ かつデッドラインが 1800 及び 2000 のときには既存手法よりも状態数が大きくなってしまっている．この実験の際にログを追ってみると，ループを含んだロケーションにより多くの述語が追加されることによって，状態が細分化されていることが判明した．FireWire Root Contention Protocol のモデルではループが 2 カ所であったが，CSMA/CD のモデルではループが 16 カ所にもなり，これにより CEGAR ループの回数が増え，状態数が増えてしまっているのではないかと考えられる．

表 3: 実験結果: CSMA/CD ($bcmax = 1$)

D	Loop	Dist	Edge	State	Time[s]
1000	1	82	109	46	0.8
1200	1	82	109	46	0.8
1400	1	82	109	46	0.8
1600	1	82	109	46	0.8
1800	167	8068	82816	583	4651.1
2000	502	71661	325344	8393	31984.1

表 4: 実験結果: CSMA/CD ($bcmax = 2$)

D	Loop	Dist	Edge	State	Time[s]
1000	1	282	400	148	1.3
1200	1	282	400	148	1.3
1400	1	282	400	148	1.3
1600	1	282	400	148	1.3
1800	63	839	3233	185	93.5
2000	63	839	3233	185	93.9

4 まとめ

本研究では，確率時間オートマトンの到達可能性解析において，CEGAR による枠組みを適用した検証手法を実装し，実験によって一定の効果が得られることを示した．本論文の主な貢献は以下である．

- 本手法を実装して実験を行い，既存手法と比較し，特定のモデルにおいて，より少ない状態数での検証が可能であることを示した

今後の課題として，以下のことが挙げられる．

- 本手法をモデル検査に拡張し，PTCTL による到達可能性以外の性質の検証

表 5: 状態数の比較: CSMA/CD ($bcmax = 1$)

D	PTCEGAR	SMC	PRISM
1000	46	362	71
1200	46	362	191
1400	46	362	311
1600	46	362	431
1800	583	440	617
2000	8393	562	725

表 6: 状態数の比較: CSMA/CD ($bcmax = 2$)

D	PTCEGAR	SMC	PRISM
1000	148	737	91
1200	148	737	423
1400	148	737	759
1600	148	737	1095
1800	185	1208	1834
2000	185	1751	2615

- 既存手法よりも状態数が多くなったモデル等で有効な状態数削減手法の考案

確率時間オートマトンに対する検証手法である Symbolic Model Checking [9] では，PTCTL (Probabilistic Timed Computation Tree Logic) によって性質を記述したモデル検査が可能であるが，確率時間 CEGAR では PTCTL で表現可能な性質のうち到達可能性のみ検証可能である．一方，CEGAR を用いた PTCTL のサブクラスによる性質について検証可能な手法として，確率時間 WiGAR[11] が提案されている．この手法では PTCTL のサブクラスではあるが，到達可能性のみならず，いくつかの性質について CEGAR を用いた検証が可能であることが報告されている．

FireWire Root Contention Protocol においては成果が得られたが，CSMA/CD においては本手法が有効であるとは言い難いため，様々なモデルで状態数の削減が望める手法を考案する必要がある．そのためには，より多くのモデルにおいて確率時間 CEGAR 検証器を適用させ，原因を特定した上で，解決策を模索していくことが今後の課題といえる．

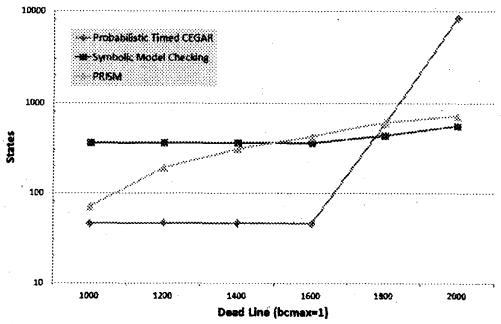

図 5: 状態数の比較: CSMA/CD(bcmax=1)

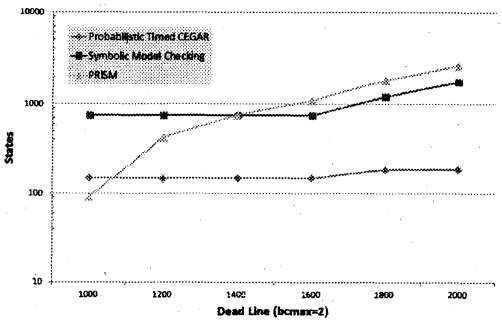

図 6: 状態数の比較: CSMA/CD(bcmax=2)

参考文献

[1] Prism - probabilistic symbolic model checker. `http://www.prismmodelchecker.org`, アクセス：2013 年 1 月 17 日.

[2] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *LNCS*, Vol. 3098, pp. 87–124, 2004.

[3] P Bouyer. Untamable timed automata! *LNCS, Vol. 2607, pp. 620-631*, 2003.

[4] E. M. Clarke, Orna Grumberg, and Doron Peled. Model checking. MIT, 2000.

[5] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joel Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. In *IJFCS*, Vol. 14, pp. 583–604, 2003.

[6] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith.

Counterexample-guided abstraction refinement. In *LNCS*, Vol. 1855, pp. 154–169, 2000.

[7] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In *LNCS*, Vol. 1254, pp. 72–83, 1997.

[8] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic cegar. In *LNCS*, Vol. 5123, pp. 162–175, 2008.

[9] Marta Kwiatkowska, Gethin Norman, Jeremy Sproston, and Fuzhi Wang. Symbolic model checking for probabilistic timed automata. In *Information And Computation*, Vol. 205, pp. 1027–1077, 2007.

[10] M. Oliver Moller, Harald Rues, and Maria Sorea. Predicate abstraction for dense real-time systems. In *Electronic Notes in Theoretical Computer Science*, Vol. 65, pp. 218–237, 2002.

[11] 高橋正樹, 清水隆也, 山根智. 確率時間 WiGAR による PTCTL サブクラスのモデル検査. 数理解析研究所講究録, 第 1744 巻, pp. 25–34. 京都大学, 2011.

[12] 清水隆也, 森下篤, 山根智. 確率時間 CEGAR の開発とその実証実験. 情報処理学会論文誌プログラミング (PRO) , Vol. 5, No. 2, pp. 43–66, mar 2012.

[13] 清水隆也, 森下篤, 山根智. 抽象化洗練を用いた時間確率システムに対する形式的検証手法 (アルゴリズムと計算理論の新展開). 数理解析研究所講究録, Vol. 1799, pp. 29–36, jun 2012.