1F3-1

ハイブリッド制約処理系 HyLaGI における LTL モデル検査

Introduction of LTL model checking to a hybrid constraint system HyLaGI

若槻祐彰 *1　松本翔太 *1　上田和紀 *1
Yoshiaki WAKATSUKI　Shota MATSUMOTO　Kazunori UEDA

*1早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻
Gradute School of Fundamental Science and Engnnering, Waseda University

Hybrid Systems consist of discrete behaviors and continuous behaviors. Physical phenomena and embeded systems can be described as hybrid systems. HydLa is a hybrid system modeling language based on constrants. HyLaGI is an implementation of HydLa that uses symbolic formula manipulation, which means that HyLaGI is free from calculation errors. HyLaGI also allows us to deal with systems with symbolic parameters. HyLaGI featured bounded model checking but its function was very limited. The purpose of this study is the introduction of LTL (Linear Temporal Logic) model checking to HyLaGI. We propose a method to verify safety and liveness properties expressed by LTL formulas by checking an inclusion relation between states in a trajectory. With our new HyLaGI, we can obtain an LTL model checking result as a counterexample of an LTL proeprty or information of state space. A feature of our method is that it uses an inclusion relation between states to detect a loop structure. In our method, we perform LTL model checking to each trajectory in a nondeterministic model. For inclusion checking, we need to abstract a model properly. We made preliminary experiment for automatic abstraction of programs to support this. In this preliminary experiment, we abstracted the initial value of each variable in a hybrid system, and succeeded in abstracting simple examples by using this method.

1. はじめに

ハイブリッド (動的) システム [2] は，時間の経過に伴い状態が連続変化したり，状態や方程式系が離散変化したりするシステムである．

ハイブリッドシステムをモデリングする宣言型言語 HydLa[4] は，方程式や論理式による制約と，制約間の制約階層によってシステムを簡潔に表すことができる．HydLa の数式処理による実装として HyLaGI[3] が存在し，ハイブリッドシステムのシミュレーション，記号実行や有界モデル検査が可能である．

従来の HyLaGI には，有界モデル検査機能があるが，常に条件式が満たされているかどうかというグローバルな安全性の判定しかできなかった．そこで本研究では，LTL(線形時相論理) モデル検査を行い解軌道が満たすべき性質を検証した．HyLaGI が出力する解軌道が満たすべき安全性や活性 (liveness) といった性質を表現する際に，LTL 式を用いることができる．

提案手法では状態空間でのループの検出に状態同士の包含関係を用いている．この包含判定のために，HydLa プログラムを適切に抽象化する必要があるが，HydLa プログラムの抽象化においては，抽象化により軌道群が包含されるように，モデルの性質や HyLaGI の実行結果を考慮して適切に抽象化する必要がある．そこでプログラマを支援するための自動抽象化機能を考察する．

2. HyLaGI への LTL モデル検査の導入

2.1 HyLaGI の LTL モデル検査アルゴリズム

状態空間探索に基づくモデル検査では，システムオートマトンと性質オートマトンからなる同期積オートマトンの状態空間を作成し，受理列が存在するか探索する．ただし，実際の実装では，同期積オートマトンの状態空間を構築しながら，受理列が存在するか探索することになる．

実装アルゴリズムは図 1 のようになる．通常の HyLaGI 実行アルゴリズム [3] に加えて，LTL モデル検査のために，入力として検証したい性質を表す LTL 式が必要になり，モデル検査部分の初期化処理 (5・6 行目)，各 PP (Point Phase，離散変化する段階) と IP (Interval Phase，連続変化する段階) の計算終了時にモデル検査を行う処理 (15・30–31 行目)，次の離散変化時刻計算のための処理 (27 行目) が追加されている．

実際にモデル検査を行う処理の *LTLModelChecking* 関数のアルゴリズムは，図 2 のように状態空間を構築・探索している [6]．この LTLModelChecking 関数は，現在検査しているノード (状態) に関する情報 CN，現在のモデル検査状況 RA，検査する性質オートマトン PA，シミュレーションしたフェーズに関する情報 (S, MS, E, CP, T) を入力とする．出力には，次の検査のためのノードの情報 CN，更新されたモデル検査状況 RA が出力される．

1 行目は次の検査のためのノードの情報を格納する変数 *NextCheckingNodes* を初期化している．2 行目の for 文では現在検査するノードの情報 CN (*CheckingNodes*) から一つずつ *CheckingNode* を取り出してループを回している．次の *MakeNextNodes* では取り出したノードから遷移可能なノードの集合 (*NextNodeCandidate*) を作成する．4 行目の for 文では *NextNodeCandidates* から一つずつ *NextNode* を取り出してループを回す．ループ内では *NextNode* を用いて，受理サイクル (活性の検査) が発見できるかどうか (5 行目)，包含関係によって通常のループとなるかどうか (6 行目)，到達可能性問題 (安全性の検査) の受理状態かどうか (8 行目)，受理サイクル問題 (活性の検査) の受理状態かどうか (9 行目)，を検査している．また，検査して受理列が発見された場合や状態空間構築が終了した場合，10 行目の *ProcessNodeInResult* 関数で現在の状態空間を RA に保存し，*CheckingNode* を状態空間から削

連絡先: 若槻祐彰，早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻，
〒 169-8555 新宿区大久保 3-4-1 63 号館 5 階 02 号，
03-5286-3340，yoshiaki(at)ueda.info.waseda.ac.jp

Input: HydLa プログラム HydLaProgram, シミュレーション終了時刻 MaxT, 検査したい性質 LTL
Output: HydLa プログラムの実行結果, LTL モデル検査の結果 RA

```
1: MS := TopologicalSort(SolveCH(HydLaProgram))
2: M_all := MaxModuleSet(MS)
3: V := GetVariables(HydLaProgram)
4: T := 0, S := ture, CP := ture
5: PA := LTL2BA(LTL)
6: (CN, RA) := InitializeModelChecking(PA, S, CP, V)
7: while T <_CP MaxT do
8:    S := SubstituteMinTime(S, T)
9:    (S, CP, E, _, _) :=
10:   CalculateMCS(S, MS, E, CP, T, CheckConsistencyPP)
11:   if S = false then
12:      break
13:   end if
14:   (S, CP) := AddParameters(S, CP, V)
15:   (CN, RA) := LTLModelChecking(CN, RA, PA, S, MS, E, CP, T)
16:   (S, CP, E, A_+, A_-) :=
17:   CalculateMCS(S, MS, E, CP, T, CheckConsistencyIP)
18:   S := SolveDifferentialEquation(S)
19:   if S = false ∨ ¬IsUnique(S, V) then
20:      break
21:   end if
22:   (MinT, CP) := GetElement(CompareMinTime(
23:   {FindMinTime(S ∧ CP ⇒ g)|(g ⇒ c) ∈ A_-}
24:   ∪{FindMinTime(S ∧ CP ⇒ ¬g)|(g ⇒ c) ∈ A_+}
25:   ∪{FindMinTime(S ∧ CP ∧ M_-)|M_- ∈ (M_all \ M)}
26:   ∪{FindMinTime(S ∧ CP ∧ ¬M_+)|M_+ ∈ M}
27:   ∪{FindMinTime(S ∧ CP ⇒ g)|g ∈ GetEdgeGuard(PA)}
28:   ∪{(MaxT − T, true)}))
29:   T := MinT + T
30:   (CN, RA) :=
31:   LTLModelChecking(CN, RA, PA, S, MS, E, CP, T)
32: end while
```

図 1: HyLaGI の LTL モデル検査アルゴリズム

Input: 現在検査するノードの情報 CN, 現在のモデル検査結果 RA, 性質オートマトン PA, 計算したフェーズの各情報 (S,MS,E,CP,T)
Output: 次に検査するノードの情報 CN, 更新されたモデル検査結果 RA

```
1: NextCheckingNodes := {}
2: for CheckingNode ∈ CN do
3:    NextNodeCandidates :=
4:    MakeNextNodes(PA, S, CP, V, CheckingNode)
5:    for NextNode ∈ NextNodeCandidates do
6:       (RA, NextNode) :=
7:       DetectingAcceptanceCycle(NextNode, CheckingNode, RA)
8:       (RA, NextNode) :=
9:       DetectingLoop(NextNode, CheckingNode, RA)
10:      NextCheckingNodes :=
11:      AddCheckingNode(NextCheckingNodes, NextNode)
12:      (RA, NextNode) :=
13:      CheckSafety(NextNode, CheckingNode, RA)
14:      NextNode := CheckLiveness(NextNode, CheckingNode, RA)
15:      RA := ProcessNodeInResult(RA, NextNode, CheckingNode)
16:   end for
17: end for
18: CN := NextCheckingNodes
```

図 2: LTLModelChecking 関数アルゴリズム

除して，次の非決定実行によるモデル検査の準備を行う．最後に 13 行目では次の検査のために，CN を *NextCheckingNodes* で更新する．

出力は，HyLaGI の実行結果に加えて，LTL モデル検査を行った結果として作成した複数の状態空間が出力される．ここで状態空間が複数になるのは，与えられる初期値が区間値のとき，定性的に異なる軌道ごとに場合分けがおこり，それぞれの軌道の性質によって構築される状態空間が異なるからである．

HyLaGI へのモデル検査の実装では，次のような理由で幅優先探索での実装を行った．一般的に深さ優先探索は幅優先探索に比べ空間計算量が小さくなるが，幅優先探索は最短の深さの目標状態を発見できる．HyLaGI のシミュレーションでは，シミュレーションを進めていくにつれて計算が複雑化し，時間がかかる場合がある．そのため「HyLaGI のシミュレーションにかかる時間計算量の増大」と「深さ優先探索に比べて幅優先探索の空間計算量の増大」を比較すると，前者の影響の方が大きいと考えた．またハイブリッドシステムは一般に無限状態なので，深さ優先だと深さが無限になり探索が終了しない場合も想定し，幅優先探索での実装を行った．このアルゴリズムはモデル検査が停止しない場合もあるため，健全だが完全ではない．

2.2 ループ検出

ハイブリッドシステムは一般に無限状態なので，モデル検査を行うと，ループを検出できない場合判定が停止しない．状態空間探索においてループを検出する必要がある．状態同士の包含関係を用いて，図 3 の 4 つの項目でループを検出する．状態 A と状態 B に関して，これらが満たされた場合 (A⊇B)，状態 B 以降の解軌道は，状態 A 以降の解軌道に包含されていることになる．これによって，状態 B 以降の遷移は，状態 A 以降の遷移と等しくなることを利用して，ループを判定している．

1: 性質オートマトンの状態が等しい
2: 採用されたモジュール集合が等しい
3: 展開されたガード条件が等しい
4: 変数値の組の取りうる領域が包含されている

図 3: 状態同士の比較手法

また，本手法では，変数間に相関がある場合を考慮して，図 3 の項目 4 では取りうる値の領域の包含を用いてループを判定している．領域を比較する二つの状態を L (large)，S (small) とする．L と S は，パラメータを引数として各変数の値を返す関数とみなすことができる．システムに現れる変数の数を n とし，それぞれの状態のパラメータに関する条件を P_S, P_L とすると，式 (1) によって領域の包含を判定してループを検出する．

$$\forall p_S \in P_S, \exists p_L \in P_L (\forall i \in 1 \ldots n, S(p_S)_i = L(p_L)_i) \quad (1)$$

式 (1) の $S(p_S)_i$ は，状態 S において i 番目の変数がパラメータ p_s によって取りうるベクトル (領域) を表している．したがって，状態 S が状態 L に包含されている場合，S 中の変数が取りうる領域の全ての値には，L 中の変数が取りうる領域の対応するある値が存在することを表す数式となる．

2.3 跳ねる質点のモデル検査

例として，跳ねる質点の例題を用いて，LTL モデル検査を行い考察する．質点の初期位置 y が (0,10)，初速度 y' が 0 で

```
1    INIT <=> 0<y<10 & y'=0.
2    FALL <=> [](y''=-10).
3    BOUNCE <=> [](y-=0 => (y'=-4/5*y'-)).
4    INIT,FALL<<BOUNCE.
```

図 4: 跳ねる質点のプログラム

運動を始める．初期位置 y が (0,10) の区間値なのは，包含されていることを判定しやすくするためである．無限回跳ねるという性質 (式 (2)，図 5) を検証する．

$$\Box\Diamond(y = 0) \wedge \Box\Diamond(y \neq 0) \tag{2}$$

このモデルの実行結果，モデル検査結果はそれぞれ，図 6，図 7 のようになる．受理列が発見できないままループを検出して状態空間の構築が終了するため，質点が無限回跳ねることがわかる．

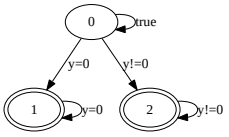


図 5: 式 (2) の性質オートマトン

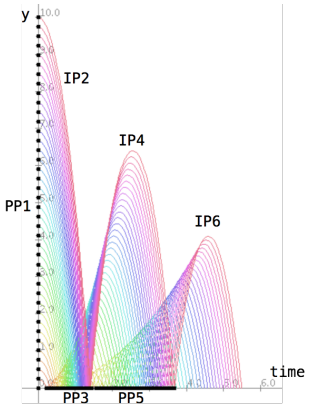


図 6: 跳ねる質点の解軌道

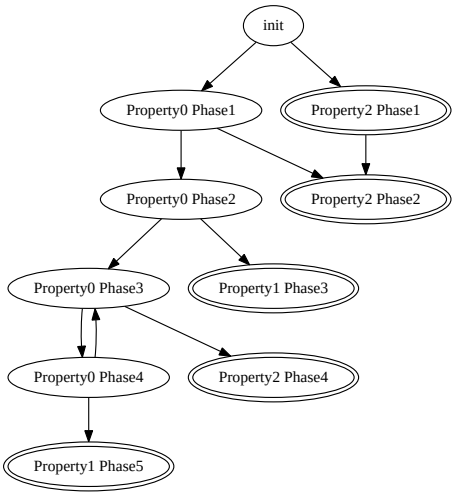


図 7: 無限回跳ねる性質のモデル検査結果

3. 非決定実行

HyLaGI はシミュレーションの結果，定性的に異なる軌道を場合分けを行って全て出力する．そのため，解軌道が場合分けされたとき，それぞれの軌道ごとに探索を行う必要が有る．

例えば図 9 のようなモデルを想定する．このモデルでは，床に穴があいていて，穴の内部にある $y = -1$ の位置のセンサーで質点が通過したかどうかわかるようになっている．このモデルで，「質点が常にセンサーに反応しない $(y \neq -1)$」性質を検証する．つまり式 (3) を検査するとする．

$$\Box(y \neq -1) \tag{3}$$

式 (3) から作成される性質オートマトンは図 8 となる．このモデルの実行結果，モデル検査結果はそれぞれ，図 9，図 10 のようになり (深さ 4 で打ち切り)，モデルの軌道が場合分けされるため，得られるモデル検査結果もそれぞれ異なる結果が得られる．床の穴に落ちた Case3 ではセンサーが反応するため，赤色で反例の経路を出力している．

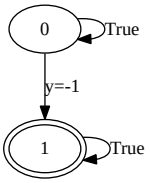


図 8: 式 (3) の性質オートマトン

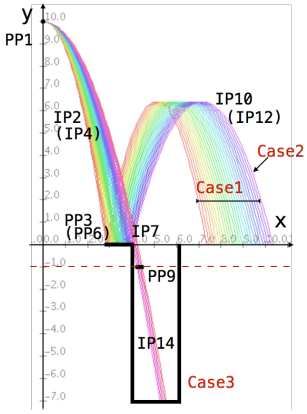
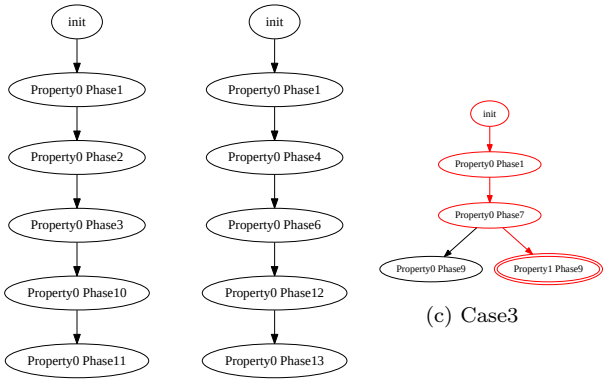


図 9: 穴のあいた床を跳ねる質点の解軌道



(a) Case1　(b) Case2　(c) Case3

図 10: 床に穴があいた跳ねる質点の LTL モデル検査の状態空間

4. 例題による評価

本研究では，表 1 の例題の LTL モデル検査に成功した．跳ねる質点のモデルの場合，床との反発係数が 1 より小さいた

モデル	LTL 式	検証結果
跳ねる質点	$\Box\Diamond(y=0)$ $\wedge\Box\Diamond(y\neq 0)$	満たしている
	$\Box(\neg(y<0))$	満たしている
	$\Box\Diamond(y>7)$	場合分けが起きる (反例がみつかる+無限状態)
	$\Diamond(y\neq 0)$	満たしている
	$\Box(y<10)$	満たしている
	$\Box(y'<10)$	満たしている
穴のあいた床を跳ねる質点	$\Box(y\neq -1)$	場合分けが起きる (反例がみつかる+無限状態)
センサーによる水量制御	$\Diamond(y=12)$	反例がみつかる
	$\Box\Diamond(y=6)$	満たしている
周期的な運動をする気球	$\Box(y>0)$	満たしている
のこぎり波	$\Box(f\geq 0\wedge f\leq 10)$	満たしている
	$\Diamond\Box(\Diamond f=11\vee\Box f=-1)$	反例がみつかる
三角波	$\Box(f\geq 0\wedge f\leq 10)$	満たしている
	$\Diamond\Box(\Diamond f=11\vee\Box f=-1)$	反例がみつかる

表 1: 評価に用いた例題

め，解軌道自体は収束し，正しく包含判定を行いつつ LTL モデル検査ができるかどうかで検証した．穴のあいた床を跳ねる質点の場合は第 3 節のように，定性的に同一の軌道ごとの状態空間は数状態だが，場合分けによって実行する LTL モデル検査自体が増えるものを検証した．結果は第 3 節で述べた通りだが，フェーズ数を伸ばすにつれて，場合分けの数が増えていった．センサーによる水量制御，周期的な運動をする気球，のこぎり波，三角波は，解軌道が周期的なモデルであるため，ループ検出のための包含判定や HyLaGI の実行の処理が軽く，LTL モデル検査の結果状態空間が 10 状態以上になるようなモデルの検証ができた．

5. 自動抽象化

ループの判定には状態の包含関係を用いているため，適切なシステムの抽象化が必要になる．例えば，図 4 のプログラムにおいて初期位置 y の値を $y=10$ にしてモデル検査を行うと，ループを検出できず状態空間は無限状態となりモデル検査は停止しない．そこで，HydLa プログラムを自動的に抽象化してモデル検査を支援する自動抽象化機能に向けて，初期値を抽象化する手法の基礎検討を行った．

5.1 初期値の自動抽象化

この方法では HydLa プログラムの初期値をすべて抽象化し，モデルの軌道を計算する．

初期値 (時刻 0 での各変数の値) をすべて抽象化するとは，実数値が取りうる値を $(-\infty,\infty)$ に変更することである．これによって，HydLa プログラムで表現されたシステムが取りうるすべての解軌道 (各変数のふるまい) を計算することになる．今回は予備実験として，初期値の自動抽象化機能を実装して，跳ねる質点のプログラムの初期値を自動抽象化したシミュレーションを行った．

跳ねる質点の例題 (図 4) の自動抽象化を行うと図 11 のようにシステムの取りうる軌道を抽象化することができた．これにより，モデル検査時のループの包含関係を判定のために自動抽象化が利用可能であると考えられる．ただし，今回の跳ねる質点の例題の場合 4 つのケースに分岐したが，床の下を運動する 3 つのケースは必要でない．これら 3 つのケースは初期値との関係で削減可能であると考えられる．

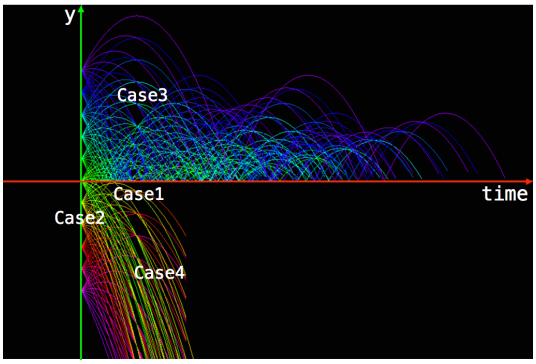


図 11: 跳ねる質点のモデルの自動抽象化シミュレーション結果のグラフ

今回の跳ねる質点のような簡単なプログラムでは可能だったが，はじめに最大限の抽象化を行っているため，抽象化したプログラムをシミュレーションすることは一般には大変難しいと考えられる．システム中の変数をすべて抽象化するため，数式処理で微分方程式が解けなくなってしまうからである．

6. まとめと今後の課題

ハイブリッドシステムの LTL モデル検査においては，ループの検出に状態の包含関係を用いている．本研究ではその実装を行い，いくつかの例題に関して LTL モデル検査ができることを確認した．また，自動抽象化の予備実験として，跳ねる質点の初期値を用いた自動抽象化を行った．

本論文では性質検証の手法として LTL モデル検査を導入した．今後は，自動抽象化したモデルの微分方程式を解けるように，最小限の抽象化を行いつつ，状態空間の包含判定が行えるように段階的に抽象度をあげていくような手法を考えたい．さらに，他の解析 ([5][1]) と連携して性質を検証することも考えていきたい．

参考文献

[1] 別納健市, 松本翔太, 若槻祐彰, 上田和紀 : 多数の離散変化をともなうハイブリッドシステムに対する不変条件を用いた解析, 情報処理学会第 78 回全国大会, 2016.

[2] Jan Lunze, Francoise Lamnabhi-Lagarrigue: Handbook of Hybrid Systems Control, Theory, Tools, Applications: Cambridge University Press, 2009.

[3] 松本翔太, 上田和紀: ハイブリッド制約言語 HydLa の記号実行シミュレータ Hyrose, コンピュータソフトウェア, Vol.30, No.4 (2013), pp.18–35.

[4] 上田和紀, 石井大輔, 細部博史: ハイブリッド制約言語 HydLa の宣言的意味論, コンピュータソフトウェア, Vol. 28 No. 1, (2011), pp. 306-311.

[5] 若槻祐彰, 松本翔太, 伊藤剛史, 和田努, 上田和紀: ハイブリッド制約処理系 HyLaGI による微小誤差を用いたモデル解析, 日本ソフトウェア科学会第 32 回大会 (2015).

[6] 若槻祐彰: ハイブリッド制約処理系 HyLaGI への LTL モデル検査の導入, 早稲田大学大学院基幹理工学研究科, 修士論文, 2016.