証明支援系 Coq を用いた有界モデル検査

藤井 采人　石井 大輔

有界モデル検査は, システムの検証問題を述語論理式にエンコードし, 高性能な SMT ソルバーで解くことにより, システムの安全性を示したり, バグを見つけたりするのに有用な方法である. 様々なエンコード法が提案されているが, それぞれの方法は, 帰納法や状態の抽象表現, 実行パスの unrolling など煩雑な考察に基づいて設計されている. 有界モデル検査の適用範囲を広げるために, エンコード法を改変するような際には, 背景の考察を理解して行わねばならず, 困難を伴う. 本研究では, ユーザがエンコード法を簡潔に定義し, プラグインされた SMT ソルバーを使って検査を実施したり, エンコード法と検査法の正しさを証明したりすることを可能にするための環境を, 証明支援系 Coq 上に構築することを目指す. そこで, 状態遷移系, 性質, エンコード関数, 有界モデル検査法の健全性を Coq 上に記述した. 実際に, Sheeran らの方法について健全性を証明し, 複数の状態遷移系の例を検査する実験を行った.

1　はじめに

モデル検査によりシステムの安全性を保証することは非常に重要であるが, モデル検査はシステムの状態空間を網羅的に探索するため, 問題によっては検証に膨大な時間を要する. そこで, 探索範囲を一定の深さに限定してシステムの検証を行う**有界モデル検査**が提案されている [4]. 有界モデル検査は, システム (状態遷移系) と検証したい性質を述語論理式にエンコードし, その充足性や妥当性を判定することで検証を行う手法である. 様々なエンコード法が提案されているが, それぞれの方法は, 帰納法や状態の抽象表現, 実行パスの unrolling など煩雑な考察に基づいて設計されている [5][10][8]. 問題ごとに適したエンコード法が異なるが, 更に複数の方法を組み合わせたり, 問題個別の考察を取り入れたりして, 効率的な有界モデル検査を行うためには, 各エンコード法を理解する必要がある. そのため, 各エンコード法の背景となる考察の見通しを良く整理するとともに, 基本となるエンコード法を組み合わせて使用可能な形で提供することが望まれる.

本研究では, ユーザがエンコード法を簡潔に定義し, プラグインされた SMT ソルバーを使って検査を実施したり, エンコード方法と検査法の正しさを証明したりすることを可能にするための環境を, 証明支援系 Coq 上に構築することを目指す. 具体的には, 状態遷移系 (システム), 検証する性質 (安全性), エンコード方法を定義することで, 検査の実施や検査法の健全性の証明を行うことができる環境である. そして, 定義したエンコード方法から述語論理式を生成し, 述語論理式の演繹的な書き換え操作 (タクティック) や, 自動定理証明器 (SMT ソルバー) の呼び出しを行うことで, 機械的に検査を行うことができる. また, 定義したエンコード方法から検査法の健全性を証明するための定理を生成することもできる. そして, 生成された定理をユーザーが手動で証明することで, 検査法の健全性を示すことができる.

本論文の構成は以下のとおりである. 第 2 章では, 有界モデル検査, 状態遷移系, 検証する性質, エンコード法について説明する. 第 3 章では, 定理証明支援系 Coq について説明する. 第 4 章では, Coq 上に構築した有界モデル検査環境について説明する. 第 5 章では, Sheeran らの方法の健全性を実装した環境下で証明す

Saito Fujii, Daisuke Ishii, 福井大学大学院工学研究科情報・メディア工学専攻, Dept. of Information Science, University of Fukui.

る流れを述べる. 第 6 章では, 関連研究を, 第 7 章では, まとめを述べる.

2 有界モデル検査

2.1 有限状態遷移系と安全性

有限状態遷移系 (システム) は, 初期状態を定める条件 I と, ある状態から別の状態への遷移関係 T からなる. 本研究では, システムの各状態を実数で表すことができるものとする. また状態を変数 s や s_i で表し, 列 $s_i, \ldots, s_j$ を $s_{[i..j]}$ で表す.

あるシステム (I,T) が (状態に関する) 性質 P について安全であるとは, I が定める初期状態から遷移関係 T によって到達可能なすべての状態が, 性質 P を満たすことを言う. 逆に, 到達可能な状態が $\neg P$ を満たすシステムは, 不具合 (バグ) を含んだシステムと言える.

2.2 有界モデル検査

有界モデル検査[4] は, システムの遷移を一定の回数 k に限定したうえで, 安全性の検査を行う手法である. 具体的には, システム (I,T), 性質 P, 正数 k を入力とし, 安全性の必要条件や十分条件を述語論理式にエンコードし, その述語論理式の成否を SAT ソルバーや SMT ソルバーによって判定し, 検査を行う手法である. 以下, 本節ではその概要を説明する.

まず, s, s' を状態としたとき, 述語論理式 $I(s)$, $T(s,s')$, $P(s)$ によりそれぞれ, s が初期状態であること, s と s' が遷移関係にあること, s が P を満たすことをエンコードできるものとする. つぎに, $s_{[0..k]}$ が状態 s_0 から k ステップ目の状態 s_k までの実行パスであることを式 $path(s_{[0..k]})$ としてエンコードできる:

$$path(s_{[0..k]}) := \bigwedge_{0 \le i < k} T(s_i, s_{i+1}) \tag{1}$$

さらに, システムの k ステップ以内の状態が P を満たさないことを以下のようにエンコードできる:

$$I(s_0) \wedge path(s_{[0..k]}) \wedge \bigvee_{0 \le i \le k} \neg P(s_i) \tag{2}$$

SAT/SMT ソルバーにより条件

$$\neg\exists s_{[0..k]}.\ (\text{式 (2)}) \tag{3}$$

が成り立つことを示せれば, システムが k ステップまでは安全であることが分かる. この式 (2) による検査では, k ステップより後の状態を含むすべての状態については安全であることは言えない.

帰納法に基づきすべての状態について安全性を示したり, 検査可能なシステムを増やしたり, エンコード式のサイズを小さくしたりするために, さまざまなエンコード方法が提案されている[10][8][5].

Sheeran ら[10] は, 帰納法に基づき簡潔な式にエンコードする 6 つの方法を提案している. ここでは, 方法 1 による安全性の検査について述べる. まず, システムが k ステップ以内で安全であることを

$$\bigwedge_{0 \le i \le k} \left(\neg\exists s_{[0..i]}.\ I(s_0) \wedge path(s_{[0..i]}) \wedge \neg P(s_i)\right) \tag{4}$$

とエンコードすることができ, SAT/SMT ソルバーで各 i について部分式が充足不能であることを確認できれば, 示すことができる. つぎに, 実行パス $s_{[0..k]}$ に状態の重複がない (閉路がない) ことを式 $loopFree(s_{[0..k]})$ としてエンコードする:

$$loopFree(s_{[0..k]}) := path(s_{[0..k]}) \wedge \bigwedge_{0 \le i < j < k} s_i \neq s_j \tag{5}$$

以下の条件を考える:

$$\Big((\neg\exists s_{[0..k]}.\ I(s_0) \wedge loopFree(s_{[0..k]})) \vee (\neg\exists s_{[0..k]}.\ loopFree(s_{[0..k]}) \vee \neg P(s_k)) \Big) \wedge (\text{式 (4)}) \tag{6}$$

上記が成り立つとき, すべての状態についてシステムが安全であることが言える (5 章で証明). 実際には, k を徐々に増やしながら式 (6) の各部分式を評価することにより, 効率よく検査を実施することができる.

3 定理証明支援系 Coq

Coq[†1] はフランスの INRIA で開発されている定理証明支援系である. Coq は, 高階述語論理式, 型, 関数などを記述するための言語を備え, 証明したい定理を記述することができる. 記述した定理について, 対話環境上でタクティックと呼ばれる専用コマンド群を用い, 簡潔に証明を記述することができる.

†1 https://coq.inria.fr/

3.1 Coq-smt-check

Coq-smt-check[†2] は, SMT ソルバー [†3] による述語論理式の妥当性判定を Coq 上でタクティック `smt solve` として用いるためのプラグインである. `smt solve` は, 証明コンテキストのゴールが, 実数変数と命題変数を含み, 量化子やユーザ定義の述語を含まない簡単な形をした述語論理式であるとき, SMT ソルバーによる求解を行い, その結果を Coq 側に反映する.

4 有界モデル検査環境の構築

本章では, 2.2 節で述べたエンコード式から有界モデル検査を行うための環境を構築する.

まず, 3 ビットのシフトレジスタの例[4] を基に, 検証するシステムと性質を記述する. シフトレジスタの i ステップ目の j ビット目を, 状態 $s_i[j]$ として扱うための型 `state` を以下のように定義する.

```
Definition state := nat → nat → R.
```

つぎに, シフトレジスタの初期状態の条件, ある状態から次状態への遷移関係, 性質を `SR_I`, `SR_T`, `SR_P` として以下のように記述する.

```
Definition SR_I (s : state) (i : nat)
 : Prop :=
  s i 2 = 1 ∧ s i 1 = 1 ∧ s i 0 = 0.

Definition SR_T (s : state) (i : nat)
 : Prop :=
 s (i+1) 2 = 1 ∧
 s (i+1) 1 = s i 2 ∧ s (i+1) 0 = s i 1.

Definition SR_P (s : state) (i : nat)
 : Prop :=
 ¬ (s i 2 = 1 ∧ s i 1 = 1 ∧ s i 0 = 1).
```

ここでは, エンコード式 (2) に基づく有界モデル検査を行うための関数 `naive_method` を記述する. まず, $path(s_{[o..(o+len)]})$ を以下のように記述する.

```
Fixpoint path (T : state → nat → Prop)
    (s : state) (o len: nat) : Prop :=
 match len with
 | 0 ⇒ True
 | 1 ⇒ T s (len + o - 1)
 | S len' ⇒ path T s o len' ∧
   T s (len' + o)
 end.
```

つぎに, 式 (3) を以下のように記述する. ただし, エンコードのために記述した関数 `violate_P` の定義は省略する.

```
Definition naive_method
 (I T P : state → nat → Prop)
 (k : nat) : Prop :=
  ∀ s : state,
  ¬ (I s 0 ∧ path T s 0 k ∧
     violate_P P s k).
```

`naive_method` によって有界モデル検査を行うためには, 以下のような定理 `ex1` を記述する. `ex1` は, 関数 `naive_method` の引数を `SR_I`, `SR_T`, `SR_P` と検査を行うステップ数 ($k = 1$) として, シフトレジスタが k ステップまで安全であることを表す述語論理式を生成している. そして, 生成した式の関数呼び出しをすべて展開, 簡約し, SMT ソルバーによる証明を行っている. ただし, 使用する SMT ソルバーは Z3 ソルバーとする.

```
Example ex1: naive_method SR_I SR_T
    SR_P 1.
Proof.
 unfold naive_method.
 unfold I; unfold T; unfold P.
 unfold path.
 simpl.
 intros.
 smt solve calling "z3"; apply by_smt.
Qed.
```

`smt solve` を実行後のメッセージウィンドウを図 1 に示す. メッセージウィンドウには, `ex1` の反例となる実行パスが表示されており, 今回の例題は 1 ステップ以内にバグが存在していたことが分かる. また, この反例では, シフトレジスタの 1 ステップ目のビットの値 (4, 7, 8 行目) がすべて 1 になっていることから, 性

†2 https://github.com/gmalecha/coq-smt-check
†3 Z3 と CVC4 をサポート.

```
Error:
Tactic failure: solver failed
    to solve the goal.

  (s 1 2) = 1.0
  (s 0 0) = 0.0
  (s 0 1) = 1.0
  (s 1 1) = 1.0
  (s 1 0) = 1.0
  (s 0 2) = 1.0
.
```

図 1　ex1 の smt solve 実行後のメッセージウィンドウ

質 SR_P が満たされていないことが確認できる.

4.1　Sheeran らのエンコード方法 1

本節では, Sheeran らの方法 1 (式 (6)) の記述を行い, システムの安全性を検査する方法を説明する.

まず, 式 (6) を Sheeran_method1 として, 以下のように記述する. ただし, エンコードのために記述した関数 lasso, violate_loop_free, kth_P_safe の定義は省略する.

```
Definition Sheeran_method1
  (I T P : state → nat → Prop)
  (k : nat): Prop :=
  ∀ s1 : state, ∀ s2 : state,
    (lasso I T s1 0 k  ∨
     violate_loop_free T P s1 0 k ) ∧
    kth_P_safe I T P k.
```

Sheeran_method1 によって有界モデル検査を行う場合の定理 ex2 を以下のように記述する.

```
Example ex2 :
Sheeran_method1 I T P k.
Proof.
 unfold Sheeran_method1.
  (* コンテキスト中の関数をすべて unfold *) ...
 simpl.
 split.
 smt solve; apply by_smt.
  (* k 回 split *) ...
 intros.
  (* k+1 回 SMT ソルバーを呼び出す *) ...
Qed.
```

ex2 では, システム (I,T), 性質 P, 自然数の値 k を Sheeran_method1 の引数として渡し, 述語論理式を生成している. そして, 生成した式の関数呼び出しをすべて展開, 簡約を行う. その後, 式の分割と SMT ソルバーの呼び出しを 1 回行う. これは, 式 (6) から式 (4) を取り除いた部分の証明である. さらに, 式 (4) を証明するため式を k 回分割して各部分式について SMT ソルバーにより証明を行う. ただし, 分割後の intros は, $k+1$ 番目の式に対して使用している. 分割したすべての式が証明されるとシステムの安全性が示されたことになる. 証明できなかった場合は, 値 k では, 安全性が示せなかったことが分かる.

5　有界モデル検査の健全性の証明

4 章において構築した環境下では, エンコード法と検査法の健全性を示すことができる. 本章では, 有界モデル検査の健全性を証明する例として, Sheeran らの方法 1 (式 (6)) の健全性の証明 [10] を形式化する.

エンコード法の健全性を示すための証明式を Coq 上に記述する. システムが安全であるということは, 初期状態から到達可能な状態がすべて性質 P を満たすことを示せば良いので,

$$\forall i.\ \forall s_0 ... s_i.\ (I(s_0) \wedge path(s_{[0..i]}) \rightarrow P(s_i)) \quad (7)$$

が成り立つことを示せば良い. つまり, エンコード法の事後条件が成り立つとき, 式 (7) が成り立つことを示せば, エンコード法の健全性を示したことになる. よって, Coq 上で健全性を証明する場合の雛形は, 図 2 のようになる.

ここでは, 例として Sheeran らの方法 1 の健全性を証明する. まず, 雛形 (図 2) 中の “a_method” の箇所に “Sheeran_method1” (式 (6)) を記述する. つぎに, 式 (7) を以下のように変形する.

$$\forall i.\ \forall s_0 ... s_i.\ \neg(I(s_0) \wedge path(s_{[0..i]}) \wedge \neg P(s_i)) \quad (8)$$

さらに, 小さい i から順に検査していく場合, 閉路を含む実行パスの到達する先はすでに検証済みであることを考慮に入れると式 (8) は以下の式と等価と言える.

$$\forall i.\ \forall s_0 ... s_i.\ \neg(I(s_0) \wedge loopFree(s_{[0..k]}) \wedge \neg P(s_i)) \quad (9)$$

この式は, 初期状態 s_0 から性質 P を満たさない状態 s_i に到達するような閉路を含まない実行パスは, 存在し

```
1 Theorem soundness : ∀ (I T P : state → nat → Prop) (k : nat),
2     (* 有界モデル検査法の事後条件 *)
3       a_method I T P k →
4     ∀ (i : nat) (s : state), I s 0 ∧ loop_free T s 0 i → P s i.
```

図 2　有界モデル検査 a_method の健全性の雛形 soundness

ないことを表す. 従って, Sheeran らの方法 1 の健全性は, 図 2 の形から「式 (6) ならば, 式 (9) が成り立つ」の形に変形できる.

以下, 証明の流れを述べる. まず, $i < k$ と $i \geq k$ に場合分けを行い, それぞれ

$$\forall s_0 ... s_i.\ \neg(I(s_0) \wedge loopFree(s_{[0..i]}) \wedge P(s_i)) \qquad (10)$$

が成り立つことを証明する.

$i < k$ の場合, 式 (6) の部分式 (4) より, k ステップ目まで安全であることが分かっているので, 式 (10) が成り立つ.

つぎに, $i \geq k$ の場合について証明する. 式 (6) より, (a) $\forall s_0 ... s_k.\ \neg(I(s_0) \wedge loopFree(s_{[0..k]}))$ (b) $\forall s_0 ... s_k.\ \neg(loopFree(s_{[0..k]}) \wedge \neg P(s_k))$ のどちらかが成り立つことがわかっている. ここで, 閉路を含まない実行パス $loopFree(s_{[0..i]})$ は分割できることを利用すると, 式 (9) の部分式 $I(s_0) \wedge loopFree(s_{[0..i]}) \wedge P(s_i)$ は以下のように書くことができる.

$$I(s_0) \wedge loopFree(s_{[0..k]}) \wedge loopFree(s_{[k..i]}) \wedge P(s_i) \qquad (11)$$

式 (11) は, (a) が成り立つとき否定されるので, 式 (10) が成り立つ. また, $I(s_0) \wedge loopFree(s_{[0..i]}) \wedge P(s_i)$ は, 以下のように分割することもできる.

$$I(s_0) \wedge loopFree(s_{[0..(i-k)]}) \wedge loopFree(s_{[(i-k)..i]}) \wedge P(s_i) \qquad (12)$$

そして, (b) は以下のように考えて良い.

$$(\forall s_0 ... s_k.\ \neg(loopFree(s_{[0..k]}) \wedge \neg P(s_k))) \Leftrightarrow (\forall s_{(i-k)} ... s_i.\ \neg(loopFree(s_{[(i-k)..i]}) \wedge \neg P(s_i))) \qquad (13)$$

(b) が成り立つときは, 式 (12) が否定されるので, 式 (10) が成り立つ. よって, $i \leq k$ の場合も, 式 (10) が成り立つ.

以上により, すべての i について式 (10) が成り立つので, Sheeran らのエンコード方法 1 の健全性が示された.

実際に Coq 上で証明する際は, 式 (7) と 式 (9) が等価であるという補題 `t1`, $i < k$ のとき, 式 (4) ならば 式 (10) が成り立つという補題 `t2`, 閉路を含まない実行パス $loopFree(s_{[0..i]})$ は分割できるという補題 `t3`, 式 (13) を補題 `t4` を用意して利用した. 現状, これらの補題の証明が課題として残っている. 上記の補題を利用して, Coq 上で本節に述べた証明を記述し, 型チェックに成功した.

6　関連研究

さまざまなクラスのシステムや性質を対象としたさまざまな有界モデル検査ツールが開発されている. たとえば NuSMV [6] は有限状態遷移系と LTL で記述した性質を対象とした有界モデル検査機能を備える. こうしたツールの問題点として, ツール自体の正しさが保証されていない点や, 検査結果の確認が容易でない点が指摘されている [1] [7] [9]. また, ツールにユーザが新たなエンコード方法を追加しようとする場合, ツールの実装を理解し, 適切に追加実装を行う必要がある. こうした実装作業はユーザに多大な負担を強いるうえ, 追加実装が正しく行われたかどうかを確認するのも容易ではない.

証明支援系により実装の正しさが検証されたモデル検査ツールが提案されている [7]. また, 多くのモデル検査ツールが依拠する BDD を定理証明支援系上に実装し, その性質を証明したり, モデル検査と演繹的検証を連携したりする手法が多数提案されている (サーベイ [1], 例 [2]). 本研究では, 検査対象を BDD の形ではなく, 直接的に述語論理式の形で定理証明支援系上で記述している.

述語論理式に基づく検証ツールに関しては, SMT ソ

ルバーの求解における推論過程や, モデル検査ツールの検査結果の保証 (proof certificate) を証明支援系で確認する手法[3][9] が提案されている. 本研究はエンコード手法の記述と正しさの証明に証明支援系を用いているため目的が異なるが, これらの手法と組み合わせることでより積極的に証明支援系を利用できると考えられる.

7 まとめと今後の課題

本研究では, 証明支援系 Coq 上にユーザーがエンコード方法を記述して有界モデル検査を実施したり, その有界モデル検査法の健全性を証明したりできる環境を構築し, 文献[10] で提案されている方法 1, 方法 2 ついて, 検査を行う関数を記述し健全性の証明を行った.

構築した環境下では, 新たなエンコード方法を簡潔に記述して検証を行ったり, そのエンコード方法の健全性をそのまま証明することができる. また, エンコード方法の柔軟な切り替えや組み合わせ, エンコード関数の部分的使用, 証明済みの定理の再利用などを行うことで, より広範囲なシステムや性質に対して, 有界モデル検査を適用可能にすることが期待される.

今後の課題は, 5 章で残された補題の証明を行い, 方法 1, 方法 2 の健全性の証明をより完全なものにする. そして, 別のエンコード方法についても, 記述と健全性の証明を行いその例を蓄積する.

謝辞 本研究の一部は科研費 15K15968 の補助を得て行った.

参考文献

[1] S. Abed, O. A. Mohamed, and G. Al. Sammane: On The Integration of Decision Diagrams in High Order Logic Based Theorem Provers: a Survey. Journal of Computer Science, 3(10):810–817 (2007).

[2] H. Amjad: Programming a Symbolic Model Checker in a Fully Expansive Theorem Prover. TPHOLs, LNCS 2758, pp. 171–187 (2003).

[3] M. Armand, G. Faure, B. Grégoire, C. Keller, and B. Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. CPP, LNCS 7086, pp. 135–150 (2011).

[4] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu: Symbolic Model Checking without BDDs. TACAS, LNCS 1579, pp. 193–207 (1999).

[5] A. R. Bradley : SAT-based model checking without unrolling. VMCAI, LNCS 6538, pp. 1–14 (2011).

[6] R. Cavada, A. Cimatti, C. A. Jochim, C. Keighren, E. Olivetti, M. Pistore, M. Roveri and A. Tchaltsev: NuSMV 2.6 User Manual. `http://nusmv.fbk.eu/NuSMV/userman/index-v2.html` (2015).

[7] J. Esparza, P. Lammich, R. Neumann, T. Nipkow, A. Schimpf, and J.-G. Smaus: A Fully Verified Executable LTL Model Checker. CAV, LNCS 8044, pp. 463–478 (2013).

[8] K. L. McMillan: Interpolation and SAT-based model checking. CAV, LNCS 2725, pp. 1–13 (2003).

[9] A. Mebsout and C. Tinelli: Proof Certificates for SMT-based Model Checkers for Infinite-state Systems. FMCAD, pp. 117–124 (2016)

[10] M. Sheeran, S. Singh, and G. Stålmarck: Checking safety properties using induction and a SAT-solver. FMCAD, LNCS 1954, pp. 108–125 (2000).