モデル検査入門

北陸先端科学技術大学院大学 先端科学技術研究科（青木研究室）

長谷川 央

2023-06-27

はじめに

ソフトウェアの典型的な開発方式（ウォーターフォールモデル）では

[仕様決め（要求分析）] → [設計] → [実装]

の順に開発が進む

開発の早い段階（設計段階）でバグが含まれていることも多い

→ バグがないかを検証し，バグがあるなら早めに取り除きたい
（デザイン検証, design validation）

テスト手法との比較

従来はシミュレーションとテストでデザイン検証をしていた

この方法はバグを取り除く方法としては優れているがバグがないことは示せない

> *program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence* [1]

シミュレーションやテストとは異なり，
モデル検査は網羅的に起こり得る振る舞いを探索するため，バグがないことを示せる [2] ．

[1] (E. W. Dijkstra, **“The humble programmer,”** Commun. ACM, 1972)
[2] (E. M. Clarke Jr, O. Grumberg, D. Kroening, et al., Model Checking. MIT press, 2018)

モデル検査の概要

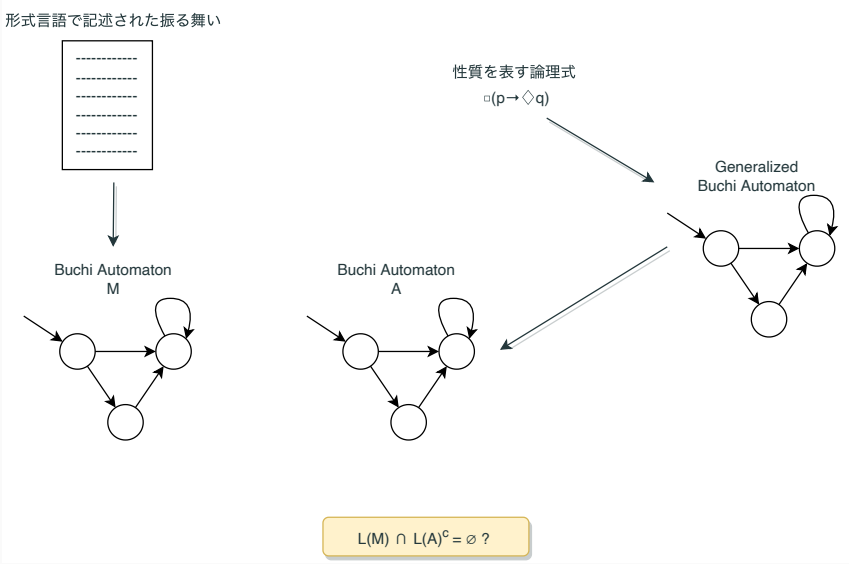
モデル検査をすると，
振る舞いを表すモデルが満たして欲しい性質・仕様を満たしているかを検証できる

モデル検査は以下の 3 ステップで行われる

1. modeling（モデリング）: 対象システムの振る舞いをモデルとして表す
2. specification（性質の記述）: 対象システムが満たすべき性質・仕様を論理式等で記述する
3. verification（検証）: モデル検査器を使ってモデルが性質を満たすかを検証する

モデル検査器は性質を満たさない振る舞い（反例，counter example）を報告する

モデル検査の概要図


形式言語で記述された振る舞い
性質を表す論理式
□(p→◇q)
Generalized
Buchi Automaton
Buchi Automaton
M
Buchi Automaton
A
L(M) ∩ L(A)^C = ∅ ?

Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) は状態遷移系上で意味論が定義されており，モデル検査では対象システムに対する要求（満たして欲しい性質）を記述するために使用される
日本語では線型時相論理と呼ばれる

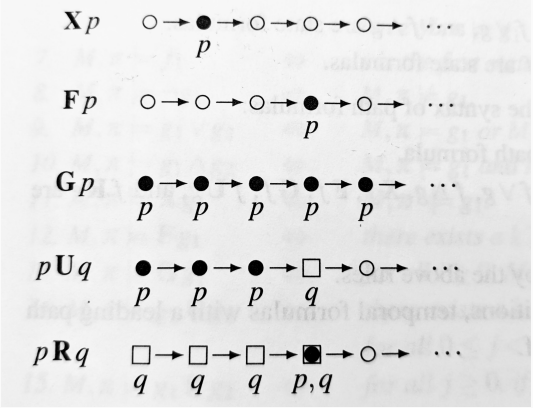
以下，定義は [3] からの引用

> Let $p \in \mathrm{AP}$ then, the syntax is defined by
> $\varphi ::= p \mid \neg\varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi\ \mathbf{U}\ \varphi \mid \varphi\ \mathbf{R}\ \varphi$, or sometimes
> $\varphi ::= p \mid \neg\varphi \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi\ \mathcal{U}\ \varphi \mid \varphi\ \mathcal{R}\ \varphi$.

[3] (E. M. Clarke Jr, O. Grumberg, D. Kroening, et al., Model Checking. MIT press, 2018)

Linear Temporal Logic (LTL) の直感的な意味

- $\mathbf{X}p$: “next time p“
- $\mathbf{F}p$: “eventually p“ or “in the future p“
- $\mathbf{G}p$: “always p“ or “globally p“
- $p\ \mathbf{U}\ q$: “p until q“
- $p\ \mathbf{R}\ q$: “p release q“



Büchi Automata (BA)

$\mathcal{B} = (\Sigma, Q, Q^0, \Delta, F)$, where

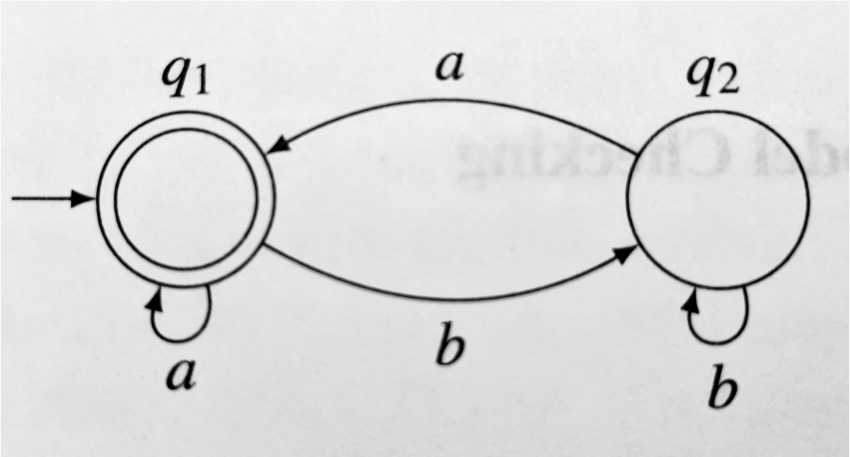
- Σ is the finite alphabet,
- Q is the finite set of states,
- $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation,
- $Q^0 \subseteq Q$ is the set of initial states, and
- $F \subseteq Q$ is the set of accepting states.

受理条件について

- the language of A is $\mathcal{L}(A) \subseteq \Sigma^\omega$（語は無限長）
- $\inf(\rho)$ を 無限長の語 ρ を読んだときに infinitly often（無限回）訪れる状態の集合とする (Eng: Let $\inf(\rho)$ be the set of states that appear infinitely often in the run ρ.)
- $\inf(\rho) \cap F \neq \emptyset$ であるとき，ρ を受理するという．

Büchi Automata の例

この Büchi automaton [4] は $(b^*a)^\omega$ を受理する



[4] (E. M. Clarke Jr, O. Grumberg, D. Kroening, et al., Model Checking. MIT press, 2018)

Generalized Büchi Automata (GBA)

直感的に言えば Büchi Automata の受理状態の集合を複数個持つオートマトン
ただし表現力は BA と等しい
LTL 式を BA に変換する際に使用する

$\mathcal{B}' = (\Sigma, Q, Q^0, \Delta, F)$, where

- $F \subseteq \mathcal{P}(Q)$; that is, $F = \{P_1, \cdots, P_k\}$, where for every $1 \leq i \leq k,\ P_i \subseteq Q$
- the other components are same with Büchi automata

受理条件について
無限長の語を ρ としたとき 全ての $P_i \in F$ で $\inf(\rho) \cap P_i \neq \emptyset$ ならば ρ を受理するという
(Eng: A run ρ of a GBA is accepting if for each $P_i \in F$, $\inf(\rho) \cap P_i \neq \emptyset$)

GBA から BA への変換

以下の方法で GBA は等価な BA に変換できる

$\mathcal{B}' = (\Sigma, Q \times \{0, \cdots, k\}, Q^0 \times \{0\}, \Delta', Q \times \{k\})$, where
$((q, x), a, (q', y)) \in \Delta'$ when $(q, a, q') \in \Delta$ and x and y are defined according to the following rules:

- If $q' \in P_i$ and $x = i$ then $y = i + 1$
- If $x = k$ then $y = 0$
- Otherwise $x = y$

$q_0 \in Q^0$ としたとき $(q_0, 0)$ からスタートし P_0 から P_k までを 1 周して
状態 $(q, k) \in Q \times \{k\} (= F')$ に到達し次の周回へ進む

すると，無限回，$P_0, \cdots, P_k$ に含まれる状態を踏むことになる

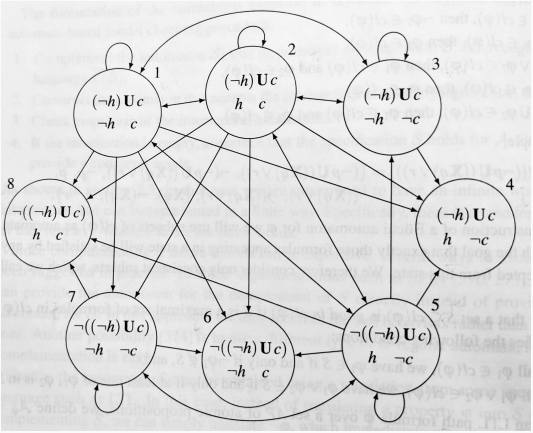
LTL 式から GBA への変換の例

この GBA は $(\neg h)\ \mathbf{U}\ c$ を満たすような語を全て受理できる
$A_\varphi = (\mathcal{P}(AP), Q, \Delta, Q^0, F)$, where

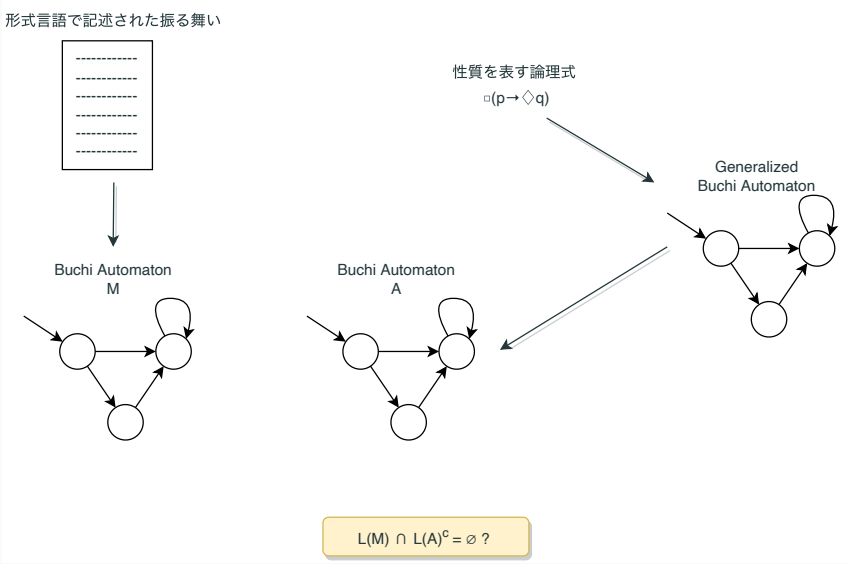
- $Q^0 = \{1, 2, 3, 4\}$
- $F = \{\{1, 2, 5, 6, 7, 8\}\}$
- $\Delta \ni (q, \sigma, q')$ is defined by following rules:
  1. $\sigma = q \cap AP$,
  2. for all $\mathbf{X}\varphi_1 \in cl(\varphi)$, we have $\mathbf{X}\varphi_1 \in q$ iff. $\varphi_1 \in q'$, and
  3. for all $\varphi_1\ \mathbf{U}\ \varphi_2 \in cl(\varphi)$, we have $\varphi_1\ \mathbf{U}\ \varphi_2 \in q$ iff. either $\varphi_2 \in q$ or both $\varphi_1 \in q$ and $\varphi_1\ \mathbf{U}\ \varphi_2 \in q'$

Note:

$cl((\neg h)\ \mathbf{U}\ c) = \{(\neg h)\ \mathbf{U}\ c, \neg((\neg h)\ \mathbf{U}\ c), \neg h, h, c, \neg c\}$



[再掲] モデル検査の概要図



Spin での例

次のソースコードは Spin model checker の記述言語 Promela で簡単な排他制御を書いたもの

```
byte mutex = 0;

active [2] proctype process() {
  do
  :: printf("%d: check mutex\n", _pid);
     mutex == 0 ->
      printf("%d: incr mutex\n", _pid);
      mutex++; /* lock */

      /* critical section */

      printf("%d: decrmutex\n", _pid);
      mutex-- /* unlock */
  od
}

#define locked (mutex == 1)
#define unlocked (mutex == 0)

ltl p1 { [](locked -> <>unlocked) }
ltl p2 { [](locked \/ unlocked) }
```

LTL からオートマトンへの変換

以下のコードは Spin model checker によって自動生成されたコードである
内容は単純な状態遷移系である
accept されたときにエラーを出力するため never claim という

```
never  {    /* !([](locked -> <>unlocked)) */
T0_init:
  do
  :: (! ((unlocked)) && (locked)) -> goto accept_S4
  :: (1) -> goto T0_init
  od;
accept_S4:
  do
  :: (! ((unlocked))) -> goto accept_S4
  od;
}
```

```
never  {    /* !([](locked \/ unlocked)) */
T0_init:
  do
  :: atomic { (! ((locked \/ unlocked))) -> assert(!(! ((
     locked \/ unlocked)))) }
  :: (1) -> goto T0_init
  od;
accept_all:
  skip
}
```