オートマトンとテスト手法

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

2022-5-7

自己紹介

名前
長谷川 央（ハセガワ アキラ）

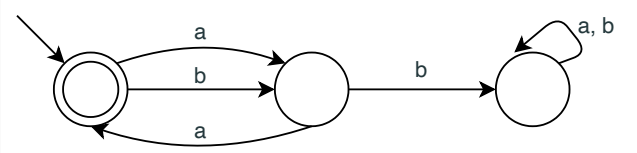
経歴

1997	愛知県豊田市で生まれる
2016	名古屋大学教育学部附属中・高卒
2016–2020	三重大学 総合情報処理センター主催 講習会「パソコン分解講習会」TA
2019	三重大学 総合情報処理センター主催 講習会「Linux 実践入門」講師
2020	北陸先端科学技術大学院大学 入学
2021–	JAIST 情報基盤センター ヘルプデスク

背景

計算機の理論を勉強するときに「オートマトンと形式言語」などの講義で
オートマトンの理論について学ぶ

しかしオートマトンの**実用的な**使用方法については授業では習わない



オートマトンをソフトウェア開発で活用する方法を知る

自作ゲーム（テスト対象）

オセロっぽいゲーム

1 次元 4 マスのボードでプレイし 相手の石を挟むと取り除ける

白手番スタートで最終的に石数が多いほうの勝ち

```
→ ./target/release/simple_game
[Light] Choose: [0|1|2|3]
1
[ |○| | ]
[Dark] Choose: [0| |2|3]
0
[●|○| | ]
[Light] Choose: [ | |2|3]
2
[●|○|○| ]
[Dark] Choose: [ | | |3]
3
[●| | |●]
[Light] Choose: [ |1|2| ]
1
[●|○| |●]
[Dark] Choose: [ | |2| ]
2
[●| |●|●]
[Light] Choose: [ |1| | ]
1
[●|○|●|●]
Dark: 3, Light: 1
DARK WIN!
```

テスト (1/2)

実装が正しく出来ているか確認したい

→ テスト手法の適用

用語

- テストケース: 入力と期待値の組
- テストスイート: テストケースの集合

テストケースの入力を与え 出力と期待値を比較する

テスト (2/2)

（例）階乗を計算するプログラムのテスト

テストケースの例

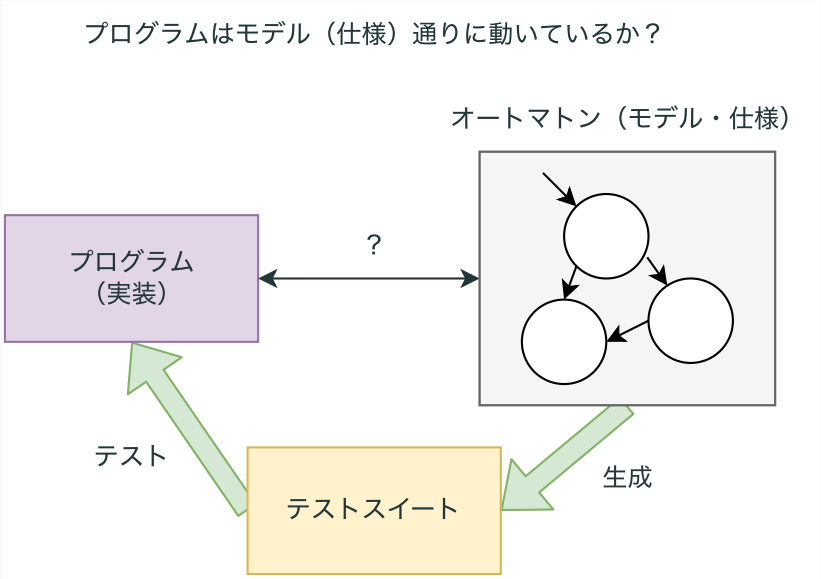
- (0, 1)
- (2, 2)
- (3, 6)
- ...
- (10, 3628800)

今回のテストの問題点

テスト対象が**状態**を持っているのでテストケースが作りづらい

→ オートマトンで動作のモデルを作成し**テストケースを自動生成する**

オートマトンを使ったテストの概要図



モデル (1/2)

今回使用するモデルの定義

$\mathcal{M} = (S, S_0, \Sigma, \delta)$, where

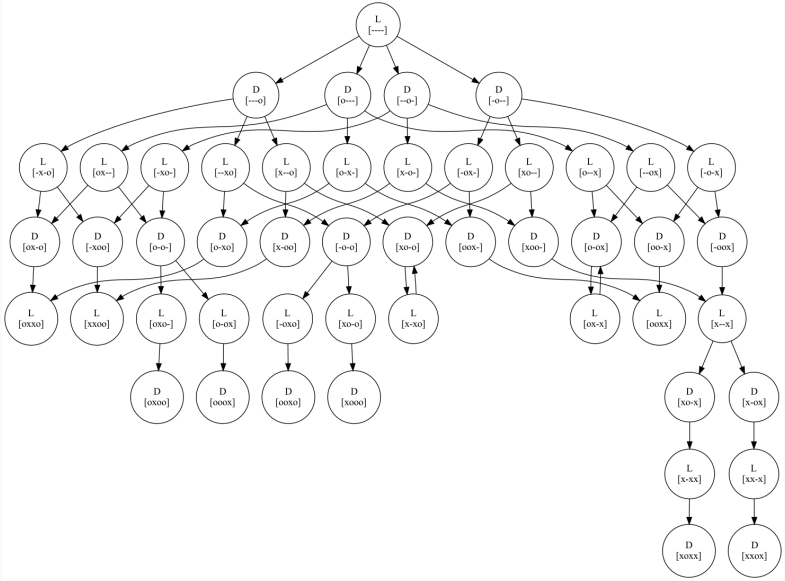
- S is a finite set of states
- $S_0 \in S$ is a initial state
- Σ is the alphabet, which is a finite set of input symbols
- $\delta : S \times \Sigma \to S$ is a transition function

状態は Turn × Board とする
e.g.) 状態 (L,[oxx-]) は現在の手番が Light（白）で盤面が [○|●|●|] であることを示す
また，(D,[- -o-]) は現在の手番が Dark（黒）で盤面が [| |○|] である

初期状態は (L,[- - - -]) とする

アルファベットは左から「n 番目に石を置く」という意味で $\Sigma = \{0, 1, 2, 3\}$ としている
紙面の都合で遷移のアルファベットに関しては省略して記載する

モデル (2/1)


L [----]
D [---o] D [o---] D [--o-] D [-o--]
L [-x-o] L [ox--] L [-xo-] L [--xo] L [x--o] L [o-x-] L [x-o-] L [-ox-] L [xo--] L [o--x] L [--ox] L [-o-x]
D [ox-o] D [-xoo] D [o-o-] D [o-xo] D [x-oo] D [-o-o] D [xo-o] D [oox-] D [xoo-] D [o-ox] D [oo-x] D [-oox]
L [oxxo] L [xxoo] L [oxo-] L [o-ox] L [-oxo] L [xo-o] L [x-xo] L [ox-x] L [ooxx] L [x--x]
D [oxoo] D [ooox] D [ooxo] D [xooo] D [xo-x] D [x-ox]
L [x-xx] L [xx-x]
D [xoxx] D [xxox]

モデルからテストスイートへの変換

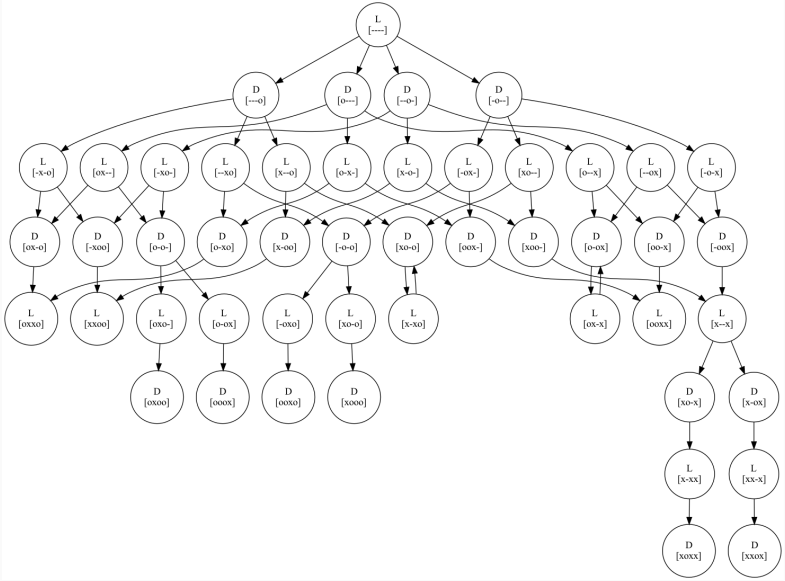
様々な変換方法があるが以下の 2 つが代表的

- All-states coverage
  全ての状態を網羅するようなテストスイート
- All-transitions coverage
  全ての遷移を網羅するようなテストスイート

どちらも深さ優先探索の応用で簡単に実装可能（今回は詳細は省略）

今回は All-transitions coverage が適当

（再掲）モデル


L [----]
D [---o] D [o---] D [--o-] D [-o--]
L [-x-o] L [ox--] L [-xo-] L [--xo] L [x--o] L [o-x-] L [x-o-] L [-ox-] L [xo--] L [o--x] L [--ox] L [-o-x]
D [ox-o] D [-xoo] D [o-o-] D [o-xo] D [x-oo] D [-o-o] D [xo-o] D [oox-] D [xoo-] D [o-ox] D [oo-x] D [-oox]
L [oxxo] L [xxoo] L [oxo-] L [o-ox] L [-oxo] L [xo-o] L [x-xo] L [ox-x] L [ooxx] L [x--x]
D [oxoo] D [ooox] D [ooxo] D [xooo] D [xo-x] D [x-ox]
L [x-xx] L [xx-x]
D [xoxx] D [xxox]

自動生成されたテストの例

```
3,(Dark, [None, None, None, Some(Light)])
2,(Light, [None, None, Some(Dark), Some(Light)])
0,(Dark, [Some(Light), None, Some(Dark), Some(Light)])
1,(Light, [Some(Light), Some(Dark), Some(Dark), Some(Light)])
====
3,(Dark, [None, None, None, Some(Light)])
2,(Light, [None, None, Some(Dark), Some(Light)])
1,(Dark, [None, Some(Light), None, Some(Light)])
0,(Light, [Some(Dark), Some(Light), None, Some(Light)])
2,(Dark, [Some(Dark), Some(Light), Some(Light), Some(Light)])
====
3,(Dark, [None, None, None, Some(Light)])
2,(Light, [None, None, Some(Dark), Some(Light)])
1,(Dark, [None, Some(Light), None, Some(Light)])
2,(Light, [None, Some(Light), Some(Dark), Some(Light)])
0,(Dark, [Some(Light), Some(Light), Some(Dark), Some(Light)])
...
```

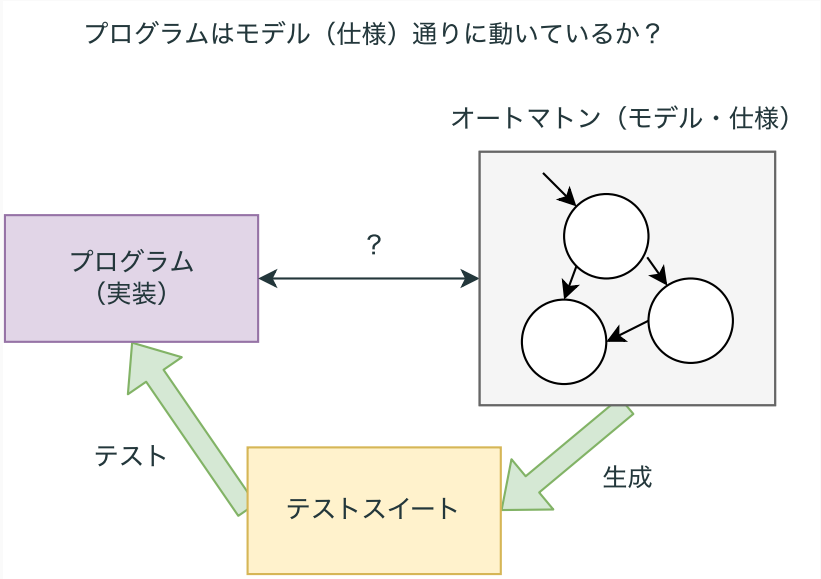
モデル作成途中でのテスト結果

モデル作成中にテストを実行しミスを探した結果
実装とモデルの動作に差異があることが分かった（修正済み）

test case 以下は失敗したテストケースを表示しており
その上に書かれているものは原因となった期待値と実際の実行結果である

```
[Failed] expected:
[●|○|●|○]
the output was:
[●|  |●|○]
test case:
Some([(3, (Dark, [None, None, None, Some(Light)])), (2, (Light, [None, None, Som
e(Dark), Some(Light)])), (0, (Dark, [Some(Light), None, Some(Dark), Some(Light)]
)), (1, (Light, [Some(Light), Some(Dark), Some(Dark), Some(Light)]))])
[Failed] expected:
[○|●|○|●]
the output was:
[○|●|  |●]
test case:
Some([(3, (Dark, [None, None, None, Some(Light)])), (2, (Light, [None, None, Som
e(Dark), Some(Light)])), (0, (Dark, [Some(Light), None, Some(Dark), Some(Light)]
)), (1, (Light, [Some(Light), Some(Dark), Some(Dark), Some(Light)]))])
```

（再掲）オートマトンを使ったテストの概要図



おまけ

Q. 状態数が大きくてモデルを手で作るのが大変なときはどうする？

A. 抽象化を行い状態数を減らす
e.g.) - ボード全体ではなく (白の数, 黒の数) を状態として扱う

→ 確かめたい性質を保ちながら抽象化を行う必要があるためテクニックが必要

→ モデル検査などの検証分野で多く研究されている

まとめ

- オートマトンは理論だけでなく実用面でも有用である
- オートマトンでモデル・仕様を表すことができる
- オートマトンからテストを自動生成することができる