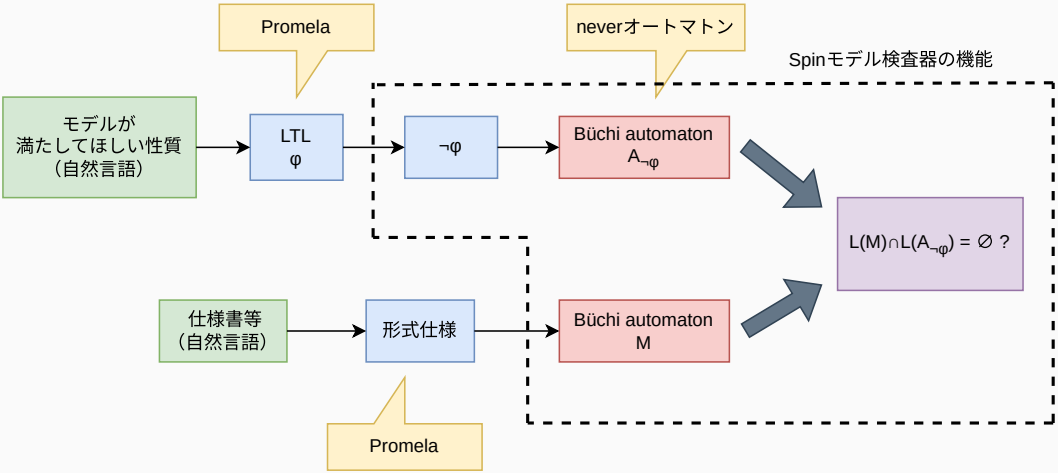
Spin モデルチェッカーと背後の理論

長谷川 央

2023-03-02

理論とツールの対応


Promela
neverオートマトン
Spinモデル検査器の機能
モデルが
満たしてほしい性質
（自然言語）
LTL
φ
¬φ
Büchi automaton
A¬φ
L(M)∩L(A¬φ) = ∅ ?
仕様書等
（自然言語）
形式仕様
Büchi automaton
M
Promela

Büchi automata と Intersection

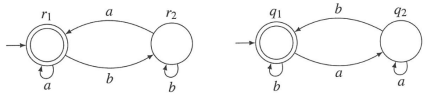


Figure 7.3
An automaton for an infinite number of a's (left) and an automaton for an infinite number of b's (right).

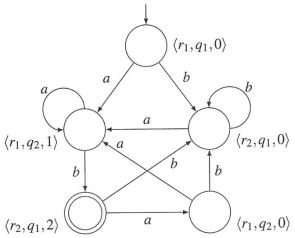


Figure 7.4
An automaton for words with an infinite number of a's and b's, obtained as intersection of the automata in figure 7.3.

Clarke Jr, Edmund M., et al. Model checking. MIT press, 2018, p. 90

Kripke structure から Büchi automaton への変換

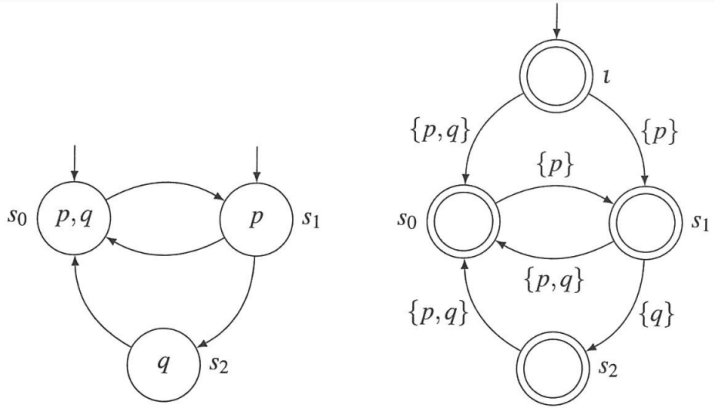


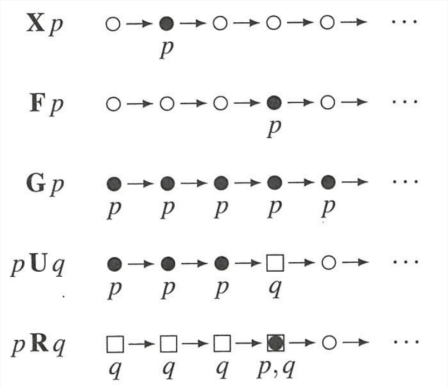
Figure 7.7
Transforming a Kripke structure (left) into an automaton (right).

Clarke Jr, Edmund M., et al. Model checking. MIT press, 2018, p. 96

構文

$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi \mathcal{U} \varphi$

直感的なイメージ



シミュレーション

以下のコマンドを実行するとランダムにパスが実行され、出力される

```
$ spin some_file.pml
```

オプションをつけると詳細な情報が見られる

```
$ spin -plgrc some_file.pml
```

モデル検査

モデル検査は次の手順で行う

```
$ spin -a some_file.pml
$ gcc pan.c -o pan
$ ./pan
```

deadlock や反例が見つかった場合は、以下のコマンドで表示できる

```
$ spin -t some_file.pml
```

最短の反例を表示したい場合は幅優先探索で探索し、反例を生成・表示する

```
$ gcc pan.c -o pan -DBFS
$ ./pan
```

Promela のプロセスについて

- プロセスは `proctype` で記述する
- 関数のような書き方をしたいときは `inline` を使用する
- `run <proc_name>`とすると、実行が開始される
- `active proctype <proc_name>`と、`active` をつけると、初期状態から実行開始される
- `active [2] proctype <proc_name>`と、`active` の後ろに個数を指定すると、
  複数個を実行できる
- `proctype A() provided(a > b)` のように書くと、プロセス A は `a > b` のときだけ
  動作する
- `init { ... }`と書くと、初期状態から active である特別なプロセスが作れる
  （主にデータの初期化やプロセスの初回起動用に使われる）

実行可能性

Promela は全ての文に実行可能性 (executability) が定義されている
実行可能な文は実行され、そうでなければブロックされる

実行不能な例

- chan 型のバッファーがフルのときに、チャネルにデータを送ろうとする（また、その逆）
- x=1 のときの x > 3 など、偽となる比較演算

if 文

Promela の if 文と do 文は条件分岐というよりも、非決定性を記述するための記法である
if の選択肢のうち、実行可能な文が非決定的に選ばれる
以下のように書けば、x が 0-2 のうちのランダムな値を取ることを表現できる

```
int x;
if
:: x = 0;
:: x = 1;
:: x = 2;
fi
```

else は特殊な文で、他の選択肢が全て実行不能である場合に実行可能になる
(条件分岐では->を使用することが多いが、; の syntax sugar であるためどちらでも良い)

```
if
:: x % 2 == 0  ->
    x = 3;
    printf("x␣was␣even,␣but␣now␣3␣is␣assigned\n")
:: else ->
    x =2 ->
    printf("x␣was␣odd,␣but␣now␣2␣is␣assigned\n")
fi
```

do 文

do 文は if 文の繰り返し版であり、以下の 2 つは同じ動作となる

```
int i = 0;
do
:: i++;
   printf("i␣was␣incremented\n");
od
```

```
int i = 0;
loop:
  i++;
  printf("i␣was␣incremented\n");
goto loop
```

chan, mtype

chan はチャネル（チャンネル）、mtype はメッセージタイプの意味であり、どちらも Promela では型として提供されている。`TypeA(x,y)` は `TypeA,x,y` の糖衣構文。

```
#define MAX 3
int i; byte j;
mtype = {TypeA, TypeB};
chan buf = [MAX] of {mtype, int, byte};

inline get_random_number(rnd) {
  for (i: 0..5) {
    rnd = i;
    if
    :: break
    :: skip
    fi
  }
}
```

```
init {
  int x; byte y;

  loop:

  get_random_number(x); get_random_number(y);
  // 送信
  if
  :: buf!TypeA,x,y;
  :: buf!TypeB(x,y);
  fi

  // 受信
  if
  :: buf?TypeA(x,y) -> printf("TypeA(%d, %d)\n", x, y);
  :: buf?TypeB,x,y -> printf("TypeB(%d, %d)\n", x, y);
  fi

  goto loop
}
```

マニュアル

- for: `https://spinroot.com/spin/Man/for.html`
- Send: `https://spinroot.com/spin/Man/send.html`
- Receive: `https://spinroot.com/spin/Man/receive.html`

atomic と d_step

次のような処理を素直に実装してしまうと、クリティカルセクションに複数のプロセスが同時に動作してしまう

```
MUTEX = 0
if (MUTEX == 0) {
  MUTEX = 1; // lock
  {
   // critical section
  }
  MUTEX = 0; // unlock
}
```

これを防ぐために、`if` 文と `MUTEX = 1` (lock) を続けて実行したいという場合がある

promela でこの処理を表す方法は 2 通り用意されている

- d_step : 複数の文を 1 つの不可分な文としてみなし、実行する
  （処理中にブロックされるような動作や決定性を持たない動作は書けないなど制限が強い）
- atomic : 複数の文を連続的に実行する（検証の複雑さを削減するためにも使用される）

線形時相論理 (Linear Temporal Logic, LTL)

SPIN モデル検査器は LTL を使用する
LTL は、モデルを記述したファイルの下側に書けば良い

構文は次のページを参照：`https://spinroot.com/spin/Man/ltl.html`

LTL 記述時、原子命題は Promela 中のラベルや、変数と比較演算子を使用する

例えば、プロセス A のラベル `Label` を通ったときに真となるラベルは、`A@label` と書く

`ltl p1 { []<>(A@label) }`と書けば、頻繁にこのラベルが踏まれることを検証でき、
`ltl p2 { [](x % 2 == 0) }`と書けば、x が常に偶数であることが検証できる

以下のように書けば、sender が送信したら、そのうち、receive されることが検証できる

```
1  #define send (A@send || B@send)
2  #define receive (C@receive)
3
4  ltl p3 { [](send -> <> receive) }
```

マニュアル

- トップページ：
  `https://spinroot.com/spin/Man/index.html`
- Promela の説明：
  `https://spinroot.com/spin/Man/promela.html`
- spin コマンドのオプション：
  `https://spinroot.com/spin/Man/Spin.html`
- コンパイル時/コンパイル後に使うオプション：
  `https://spinroot.com/spin/Man/Pan.html`

その他

- assert 文
  https://spinroot.com/spin/Man/assert.html
- progress ラベル
  進行性検査に使う（実行時は別途オプションの指定が必要なので要注意）
  https://spinroot.com/spin/Man/progress.html
- end ラベル
  意図しない位置でブロックされて止まっているプロセスがないかを確かめる機能が spin にはある。その機能に、「ここは止まっても良い場所」と伝えるために使う。
  (e.g. main などの無限ループ）https://spinroot.com/spin/Man/end.html