GSLetterNeo vol.140

2020 年 3 月

形式手法コトハジメ

–TLA$^+$ Toolbox を使って (6)-

熊澤 努 kumazawa @ sra.co.jp

はじめに

GSLetterNeo Vol.138 では、並行性を持つシステムの仕様を書くための方法として、プロセスを導入しました。プロセスによって、複雑な挙動をするシステムを簡潔に書くことができることがわかりました。

今回は、引き続き、プロセスを使った仕様を正しく書くための方法を解説します。まず、仕様の正しさをチェックする機能である検証機能を使って、これまで書いてきた信号機の仕様に誤りがないかチェックします。そして、その結果をもとに仕様をブラッシュアップしていきます。

今回は連載記事「形式手法コトハジメ–TLA+ Toolbox を使って-」の最終回です。最後までお付き合いいただけましたら幸いです。

信号機の仕様を検証する

Vol.138 では、以下の仮定の下で、信号機が 2 基の場合の仕様を考えました。

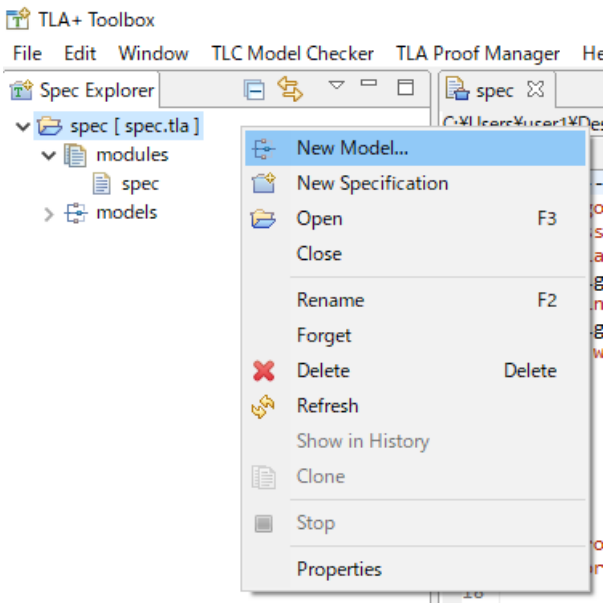
仮定1. 全ての信号機は「赤→青→黄」の順に点灯を繰り返す。故障や電源 OFF については考えない。

仮定2. 各信号機は、他の信号機の灯火や挙動に依存しない固有のタイミングで、灯火を変化させる。

プロセスを導入して記述した仕様を下に再掲します。

```
------------------------------- MODULE spec -------------------------------
(* --algorithm TrafficLight
  process tl \in {1, 2}
    variables
      light = "red";
    begin
      Signal:
        while TRUE do
          Green:
            light := "green";
          Yellow:
            light := "yellow";
          Red:
            light := "red";
         end while;
  end process;
end algorithm;*)
=============================================
```

さて、今回は、仮定を仕様がどの程度正しく反映しているのか詳しく検討しましょう。これは、仕様を検証することで実現できます。検証については Vol.130[1]で簡単に触れましたが、仕様の正しさを検査することです。TLA$^+$ Toolbox にはモデル検査という検証技術が実装されているので、使ってみましょう。TLA$^+$ Toolbox で検証をするには、テスト項目のように検証すべき項目をあらかじめ決めておく必要があります。ここでは、仮定 1 を簡単に述べた「信号機は 2 基とも点灯を繰り返す」こととしましょう。



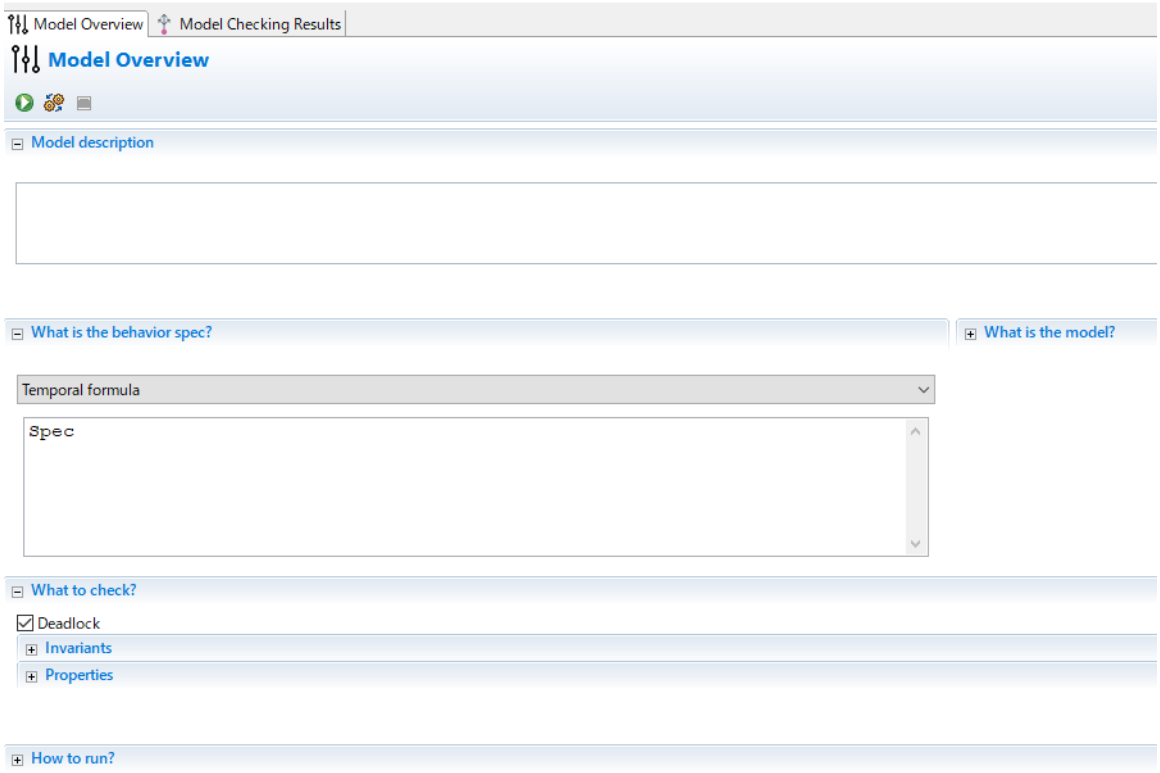
検証を実行するには少々準備が必要です。まず、上の PlusCal の仕様を TLA$^+$に変換してください。Vol.134[2]で説明した通り、**File** メニューの **Translate PlusCal Algorithm** を選択するだけですね。

次に、検証のための「モデル」を作成します。仕様を開いた状態で、Spec Explorer

[1] https://www2.sra.co.jp/Portals/0/files/gsletter/pdf/GSLetterNeoVol130.pdf
[2] https://www2.sra.co.jp/Portals/0/files/gsletter/pdf/GSLetterNeoVol134.pdf

内で右クリックしてください。前頁右下の図のようなメニューが表示されるので、一番上の**New Model...**を選択します。モデルの名前の入力が求められたら、適当な名称を入力します。今回は単純に Model_1 としておきます。作成したモデルは Spec Explorer の models という項目に追加されます。

モデルは自動的に開かれます(開かれない場合には、models から選択してください)。下のような Model Overview 画面が表示されると思います。この画面では、検証したい仕様や検証項目を設定します。



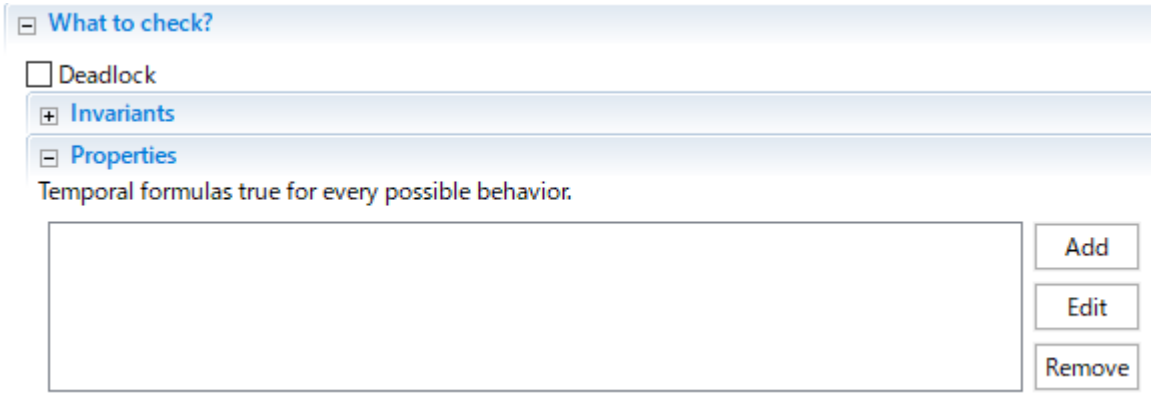
今回は必要最低限度の設定のみにとどめ、細々とした設定は飛ばすことにします。What is the behavior spec? という項目を見てください。ここでは検証する仕様を設定します。上の図のように Temporal formula が選択されていて、Spec と書かれていることを確認してください。 Vol.130 で説明したように、Spec は TLA$^+$に変換した際に自動的に作られる、信号機 2 基の動きを記述した仕様のことです (変換後の仕様の最終行にある `Spec == Init /\ [][Next]_vars` という記述です) 。

続けて、検証したい内容を設定する What to check? という項目を編集します。まず、Deadlock のチェックを外して、デッドロックの検証を実行しないようにしてください。それから、Properties の左脇の+をクリックして、メニューを開いてください。下の図のよ

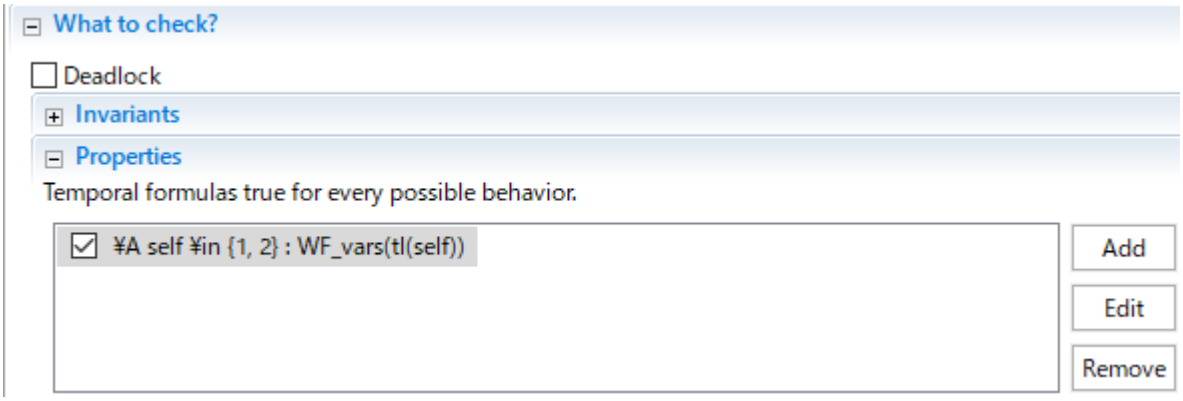
うな空のメニューが表示されます。空の四角の右端の Add ボタンを押して、以下のように記述して、Finish ボタンを押してください。

```
\A self \in {1, 2} : WF_vars(tl(self))
```

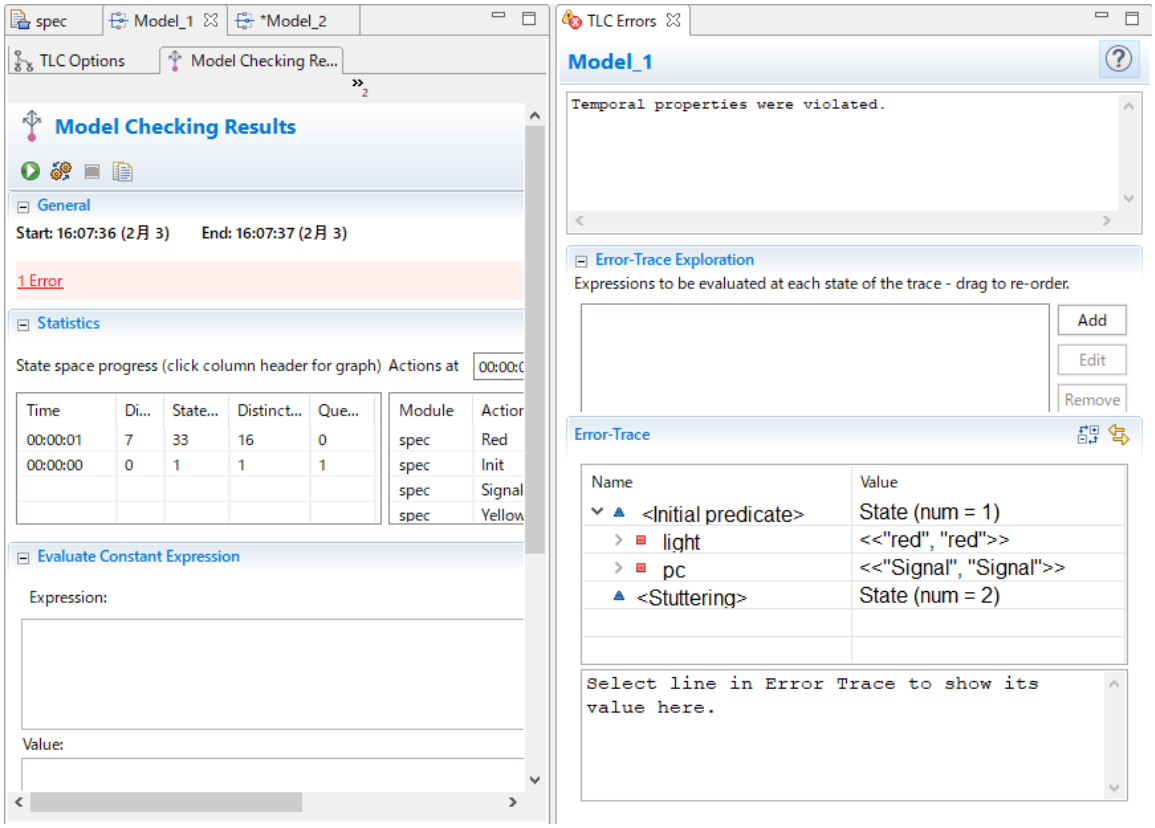
これが今回正しいかどうか検証する内容を形式的に記述した式です（詳細は省略します）。



下の図のように追加した項目にチェックが入っていれば準備完了です。チェックが入っていない場合には、手作業でチェックを入れてください。



それでは、検証を実行してみましょう。Model Overview の左上のボタン  をクリックすると検証が始まります。結果は、以下のように Model Checking Results 画面に自動的に表示されます。



画面左側に判定結果が 1 Error と表示されています。これは、検証の結果、仕様は「信号機は 2 基とも点灯を繰り返す」という検証項目に違反していることを意味します。つまり、仕様は仮定 1 を満たしていないのです。右側には、検出された違反の詳細な説明が書かれています。特に重要なのは、Error-Trace という仕様に違反した実行例を表示する項目です。TLA$^+$ Toolbox はここに、初期値から違反までの変数の値の変遷を表示します。青の三角は状態(変数の値の組合せ) で、右隣の<>は状態名(コメントのようなもの) です。例えば、<Initial predicate>は初期状態を表しています。赤の四角は変数で、その値は<<>>で囲って表します。初期状態では変数 light の値は、 2 基の信号機ともに赤となっています。そして、次の状態<Stuttering>に遷移して、その結果、検証項目に違反してしまったことをこの違反例は説明しています。

この違反は何を意味しているのでしょうか。実は、TLA$^+$ Toolbox で書かれた仕様には、stuttering(「つっかえること」の意)という特殊な挙動が許されています。stuttering とは、各信号機が動いている間に変数の値が変化しない「何もしない」期間や状態を指します。要するに、信号機が 2 基ともに「赤」のまま灯火色が遷移せず、したがって、「信号機は 2 基とも点灯を繰り返す」ことがない、という場合を TLA$^+$ Toolbox の検証機能が検出したというわけです。仕様に書いた、信号機の灯火は常時遷移するものだ、という私たちの

信号機に対する暗黙の想定が原因で生じた違反例と考えられます。実際には、振動機を挙動を構成する各ステップの進行の仕方について何も仕様に記述されていないため、灯火が永遠に変化しない場合が含まれてしまったのです。

公平なプロセスを書く

検証によって仕様が誤っていることがわかったので、仕様をもう一度修正しましょう。stuttering のない仕様を書けばよいのですが、それを実現する仕組みは PlusCal に備わっています。書き方は非常に簡単です。下の図のように、`process` の前に `fair` と書くだけです。

```
------------------------------ MODULE spec ------------------------------
(* --algorithm TrafficLight
  fair process tl \in {1, 2}
    variables
      light = "red";
    begin
      Signal:
        while TRUE do
          Green:
            light := "green";
          Yellow:
            light := "yellow";
          Red:
            light := "red";
         end while;
  end process;
end algorithm;*)
=========================================
```

このようなプロセスを「公平なプロセス[3]」と呼びます。公平なプロセスは「システムの実行中に、(変化できるのであれば)いつか変数の値が変化する」ようなプロセスのことです。信号機の仕様では、点灯を繰り返すような信号機を記述していることになります。この仕様をもう一度 TLA⁺に変換して検証すると、エラーが消えることがわかります。試してみてください。

公平なプロセスは、信号機のように常に進行し続けたり、待機のすることのないシステムを記述したりする場合に威力を発揮します。一方、計算機のプロセスやスレッドをモデル化し

[3] 本稿の stuttering や公平なプロセスの説明はきちんとしたものではありません。正確に理解したい読者は、本稿最後の文献案内を参照してください。

た仕様では、特定のプロセスを待機させる場合など、プロセスの用途によっては公平なプロセスを使わない場合もあるでしょう。うまく使い分けてください。

おわりに

本稿では、TLA$^+$ Toolbox の検証機能を使って、信号機の仕様の検証を行いました。その結果、公平なプロセスを使うことで、灯火の色が遷移する信号機の仕様を書くことができることがわかりました。

これまで、TLA$^+$ Toolbox と仕様記述言語 PlusCal を使った形式仕様の書き方について、基本事項を解説してきました。PlusCal について説明しなければならないことはまだあるものの、一通りの機能を見ましたので、後はいろいろと使っていただくのが良いと思います。

最後に、もう少し詳しく知りたい読者のための参考資料を案内して本連載を終えたいと思います。残念ながら、本稿の執筆時点では、筆者の知る限り、これらの書籍の日本語訳はありません。

PlusCal を使って様々な仕様を書くには、以下が参考になるでしょう。

Wayne, H. (2018). *Practical TLA+: Planning Driven Development*. Apress.

TLA$^+$や本稿に出てきた stuttering、公平なプロセスなどの概念については、Lamport の本を読むのが良いと思います。

L. Lamport. (2002). *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Longman Publishing Co., Inc.

また、TLA$^+$ Toolbox のヘルプからも、この本を含め、いろいろな情報にアクセスできます。

TLA$^+$ Toolbox が採用している検証技術はモデル検査と呼ばれています。モデル検査の詳細については、下記の書籍がよく知られていますが、どちらも読み通すのは大変です。

Clarke Jr., E. M., Grumberg, O., Kroening, D., Peled, D., & Veith, H. (2018). *Model Checking* (2nd ed.). MIT Press.

Baier, C., & Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.

GSLetterNeo Vol.140
2020 年 3 月 20 日発行
発行者　株式会社 SRA 先端技術研究所

編集者　土屋正人
バックナンバー http://www.sra.co.jp/gsletter
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8
夢を。Yawaraka Innovation
やわらかいのべーしょん