GSLetterNeo vol.134

2019 年 9 月

形式手法コトハジメ

–TLA$^+$ Toolbox を使って (3)-

熊澤 努 kumazawa @ sra.co.jp

はじめに

GSLetterNeo Vol.132 では、TLA$^+$ Toolbox の仕様記述言語 PlusCal を使って、簡単な信号機の仕様を書くところまで解説しました。今回は、仕様の構文チェックを行い、TLA$^+$ Toolbox がサポートするもう一つの仕様記述言語 TLA$^+$に変換する機能を使ってみましょう[1]。

---

[1] GSLetterNeo Vol.132 で使用した TLA$^+$ Toolbox のバージョンは 1.5.7 でしたが、本稿では、執筆時点（2019 年 8 月）で最新の 1.6.0 を使用します。

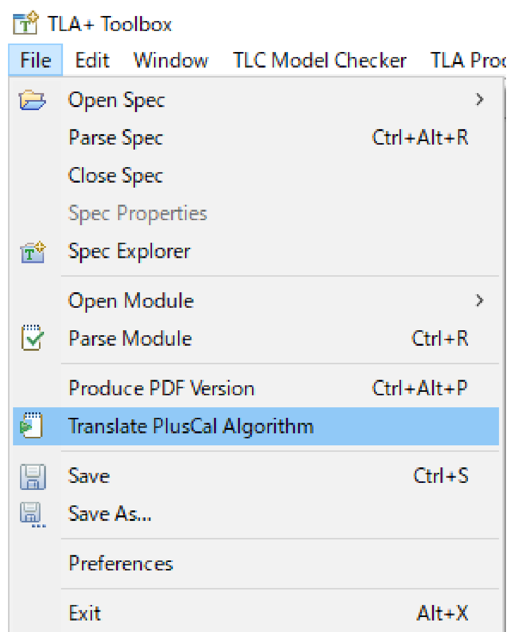
TLA+に変換する

Vol.132 で書いた信号機の仕様を以下に再掲します。ファイル名は spec.tla です。この仕様は PlusCal の文法に正しく従って書けているでしょうか。

```
-------------------------------- MODULE spec --------------------------------
(* --algorithm TrafficLight
variables
    light = "red";
begin
  Green:
    light := "green";
  Yellow:
    light := "yellow";
  Red:
    light := "red";
end algorithm;*)
=============================================================================
```

Vol.132 で述べたように、PlusCal で書いた仕様は TLA^+ のコメントとして扱われるため、TLA^+ Toolbox は読み飛ばしてしまいます。そのため、仕様の検証のみならず構文のチェックも PlusCal の仕様に対しては機能しません。

そこで、TLA^+ Toolbox が提供する PlusCal から TLA^+への自動変換機能を使い、PlusCal の仕様を TLA^+で書かれた仕様に変換します。その仕様が構文上正しければ変換に成功して、Toolbox は TLA^+で書かれた仕様を生成します。Toolbox には TLA^+で書かれた仕様を対象とした自動検証機能が実装されていますので、この手順を実施することで、PlusCal の仕様を検証できるようになります。



PlusCal からTLA^+への自動変換を行うには、仕様を開いた状態で**File**メニューから**Translate PlusCal Algorithm** を選択してください（右図）。

spec.tlaの `end algorithm;*)` の次の行に新たな記述が自動的に追加されたはずです。この追加された記述が、仕様記述言語TLA^+で書かれた信号機 spec の仕様です。少々長いですが、Toolboxが追加したTLA^+の仕様を以下に載せます。

```
\* BEGIN TRANSLATION
VARIABLES light, pc

vars == << light, pc >>

Init == (* Global variables *)
        /\ light = "red"
        /\ pc = "Green"

Green == /\ pc = "Green"
         /\ light' = "green"
         /\ pc' = "Yellow"

Yellow == /\ pc = "Yellow"
          /\ light' = "yellow"
          /\ pc' = "Red"

Red == /\ pc = "Red"
       /\ light' = "red"
       /\ pc' = "Done"

(* Allow infinite stuttering to prevent deadlock on termination. *)
Terminating == pc = "Done" /\ UNCHANGED vars

Next == Green \/ Yellow \/ Red
           \/ Terminating

Spec == Init /\ [][Next]_vars

Termination == <>(pc = "Done")

\* END TRANSLATION
```

本稿ではTLA+ を詳しくは解説しませんが、内容を簡単に見てイメージをつかみましょう。TLA+で書かれた仕様は、PlusCalで書かれた仕様とは様子がかなり異なりますね。手続き型プログラミング言語に近いPlusCalとは異なり、TLA+は論理式をベースとした仕様記述言語です。変数の宣言やコメント、一部の定義を除けば、仕様のほぼ全てを論理式で記述します。

まず、最初の`* BEGIN TRANSLATION` は一行コメントです(`*` がコメントの識別子です)。つまり、その次の行から最後の `* END TRANSLATION` の直前の行までがTLA+ で書かれた仕様となります。

次の行の `VARIABLES light, pc` はPlusCalと同じように変数宣言です。ただ、PlusCalでも宣言した変数 `light` の他に、見慣れない変数 `pc` があります。これは変換の際に自動的に追加される変数で、信号機の挙動の制御状態を管理するためのものです。ここでは、信号機の灯火の順序を制御する変数と考えればよいと思います。

信号機の状態遷移は、変数 light と pc の値の変化を論理式で記述することで表現します。状態遷移を記述している箇所は Init で始まる行から Termination で始まる行までですが、主要な部分は Init == ...、 Green == ...、 Yellow == ...、 Red == ... の4つの式です。これらは、信号機の初期状態（つまり灯色が赤）と、灯色が緑、黄、赤のそれぞれの状態における各変数の値の変化を論理式で定義しています。記号 /\ はいわゆるAND演算子です[2]。例えば、初期状態 Init は変数 light と pc の値がそれぞれ "red" と "Green" である、と定義されています。一方、状態 Green は、pc の値が "Green" であり、その次の時点での light と pc の値がそれぞれ "green" と "Yellow" であると定義されています。引用符 ' が付された変数に関する式は「次の時点での値」を意味します。信号機の状態変化は pc の値で制御され、初期状態における値は "Green" です。また、pc の値が "Green" であるような状態は Green であるので、次の時点での信号機の状態は Green になります。それにより、変数 light の値は初期値 "red" から "green" に変化します。この値の変化はPlusCalで書いた仕様に合致していることに注意しましょう。他の状態についても同様に考えることができるので、PlusCalで書いた元の仕様の意味が変わることなくTLA$^+$に変換できていることがわかります。終了状態や各状態への遷移の仕方は Terminating 以下に記述されていますが、その説明には記号論理学についての踏み込んだ解説が必要ですので、本稿では割愛します。

TLA$^+$に変換した後にPlusCalの仕様を書き直した場合には、改めてTLA$^+$に変換する必要があるので注意してください。その際、すでに生成されている古いTLA$^+$の仕様を手作業で消す必要はありません。もう一度、**File**メニューから**Translate PlusCal Algorithm** を選択すれば、TLA$^+$の仕様が新しいものに置き換わります。このように、PlusCalを使って仕様を書いているユーザはTLA$^+$を使わずに済みます。TLA$^+$で書いた仕様を誤って編集してしまった場合(TLA$^+$ ToolboxがTLA$^+$の構文エラーを表示することがありますが無視してください)も、同じ様に変換をやり直してください。

---

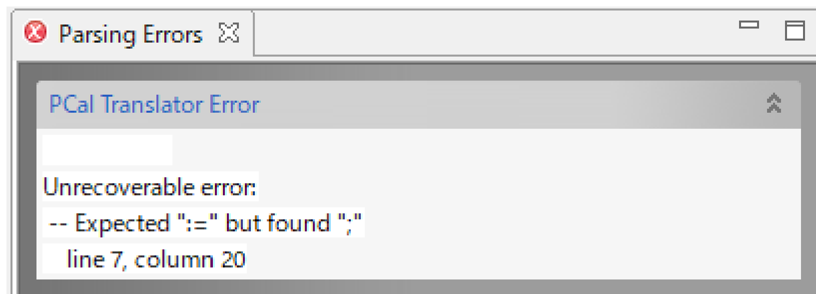
[2]連言や論理積とも呼ばれる論理演算子で、記号論理学ではしばしば ∧ という記号が使われます。Java や C/C++における && 演算子に相当します。論理積 $p \wedge q$ はpとqがともに真のときのみ真と評価されます。TLA+で使われる数学の詳細については、書籍 L. Lamport. (2002). Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc. (https://lamport.azurewebsites.net/tla/book.html?back-link=learning.html#book) の第 1 章を参照してください。

構文エラーを検出する

自動変換機能によって、PlusCal の構文エラーを検出することもできます。試みに、PlusCal で書いた信号機の仕様の一部を書き換えてエラーを出力させてみましょう。下のように、代入演算子 := からコロンを取り除いて = にしたうえで、TLA$^+$に変換してみます。

```
-------------------------------- MODULE spec --------------------------------
(* --algorithm TrafficLight
…略
  Green:
    light = "green"; \* 正しくは light := "green";
…略
end algorithm;*)
=============================================================================
```

すると、変換に失敗するだけでなく、構文エラーを示すエラーメッセージが、以下のように表示されます。このメッセージは、本来あるべき代入演算子 := がないことを述べているので、書き換えによって生じたエラーを適切に検出していることがわかります。



さらに、下の図のように、書き換えた行が赤くハイライト表示されて、エディタ上でもエラーの発生個所を容易に特定できます。

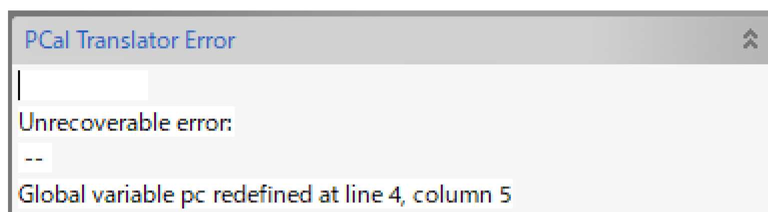
TLA Module

```
1  -------------------------------- MODULE spec --------------------------------
2  (* --algorithm TrafficLight
3  variables
4      light = "red";
5  begin
6    Green:
7      light = "green";
8    Yellow:
9      light := "yellow";
10   Red:
11     light := "red";
12 end algorithm;*)
```

もう一つ TLA$^+$への変換エラーの例を見ましょう。PlusCal で仕様を書いた場合、変数名に pc を使用することはできません。上で述べた変換の際に追加される変数 pc と名称が重複するためです。これも変換時のエラーとして検出することができます。例えば、信号機の仕様で下のように変数 light を pc に変更してみます。

```
-------------------------------- MODULE spec --------------------------------
(* --algorithm TrafficLight
variables
    pc = "red";
begin
  Green:
    pc := "green";
  Yellow:
    pc := "yellow";
  Red:
    pc := "red";
end algorithm;*)
=============================================================================
```

この仕様も変換に失敗します。エラーメッセージには、変数 pc が再定義されている旨が述べられており、変数名に pc を使用できないことがわかります。



おわりに

本稿では、PlusCal で書いた仕様を TLA$^+$言語に変換する機能について説明しました。また、仕様記述言語 TLA$^+$を簡単に解説しました。TLA$^+$は論理式により仕様を記述する言語のため、PlusCal と比べると敷居が高いと思います。また、PlusCal を使用する限りは、Toolbox が自動的に TLA$^+$に変換してくれるため、ユーザが TLA$^+$に触れることはあまりないかもしれません。しかしながら、TLA$^+$ Toolbox の検証機能は TLA$^+$で記述された仕様が対象であるため、いずれは習得しなければならなくなります。信号機のような簡単な仕様から始めて、少しずつ見て慣れていきましょう。

次回は PlusCal でもう少し複雑な仕様を書いてみたいと思います。

GSLetterNeo Vol.134
2019 年 9 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