Week1 SWEN421 2020

Watch and discuss Video 1 (20-min) & 2 (15 min)

Watch the first 4 mins of video 8(a) and first 5mins of video 9(a) these two video fragments go over how to use the TLA+ tool to write functions.

Goal - to be able to read and understand TLA+ assertions. You should be able to (a) translate simple TLA+ assertions into plain English and explain them plus (b) translate English assertions into TLA+.

You will need to:

1. install TLA+
2. install GraphViz.
3. if you want to produce nice pdfs you will need to have installed latex.

In this weeks lectures we will (a) review :

1. Propositional logic truth tables
2. Predicate Logic, (variables free and bound - $\forall$, $\exists$)
3. Set theory,
4. lambda, map and reduce

and (b) explore how to write and validate assertions using TLA+ and its model checker.

To explain these idea using TLA+ syntax you might find TLA+ summary helpful. But it assumes that you are familiar with latex as most, but not all, of the commands are the same as used in latex, e.g. for $\in$ type `\in` but for $\mapsto$ type `|->` and not the latex command `\mapsto`.

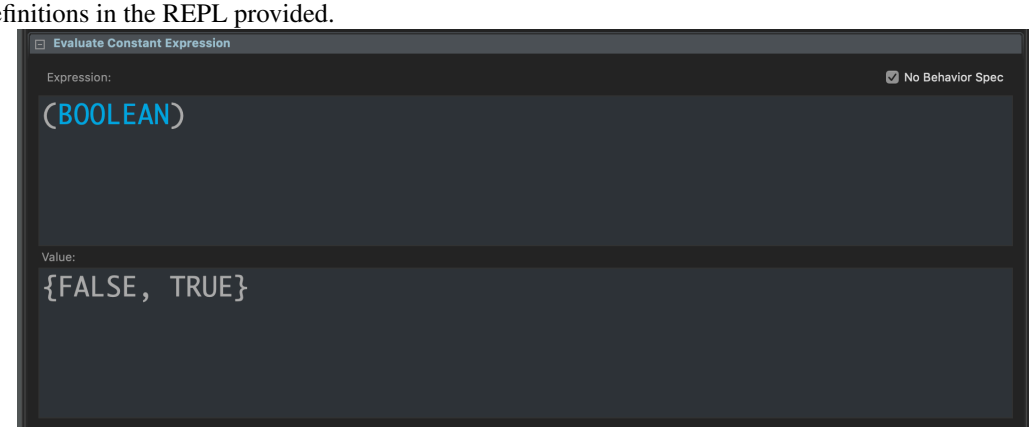
The best introduction to TLA+ terminology I have found is TLA+ in the learning PlusCal tutorial. Please read the section named **TLA+** that covers most of what you will need this week. This tutorial is also an excellent source of exercises with answers that are initially hidden. Please use it wisely as a tool for learning not as a speed reading exercise. The definition of the Standard Modules defining the TLA+ operators on Sets, Sequences, ... can be found in Chapter 18 of Specifying Systems and should be used as a reference while completing the Home Work.

Home Work

Read the web pages https://learntla.com/tla/ but ignore the few examples that start with `(* --algorithm`. These are the larger examples and contain PlusCal specific commands, the other examples only contain TLA+ code.

Build a new directory >Week1 and in the TLA+ tool open a new specification in that directory. Name the specification >`Week1.tla`.

1. Build a new Model and 2. in the "Model Checking results" tab use **evaluate constant Expressions** with the "No Behaviour Spec" box checked to validate your definitions in the REPL provided.



Some time the brackets and needed sometimes not.

You can also save you results and print them as below:

```
EXTENDS Naturals, Sequences, Integers, TLC
VARIABLES  q1, q2, q3
Init == /\ q1 = BOOLEAN
        /\ q2= 1 \* your def here
        /\ q3 = 2 \* your def here
        /\ PrintT("q1= "\o ToString(q1) \o " q2= "\o ToString(q2))
 Next ==  /\ q1' = q1
          /\ q2' = q2
          /\ q3' = q3
```

Read Section 14.5 "How to use TLC" of Specifying Systems, particularly Section 14.5.3 "how to use TLC as a TLA+ calculator".

Below are exercises in writing abstract specifications do not use literals `TRUE` and `FALSE`. **In these exercises Do not use constants as they are parameters to the model.**

q1 the set of all truth values.
Solution: `BOOLEAN` **check you get the output** `{TRUE,FALSE}`

q2 the set of all pairs of truth values.
output `{<<TRUE,TRUE>>,<<TRUE,FALSE>>,<<FALSE,TRUE>>,<<FALSE,FALSE>>}`

q3 a tuple of all truth values. *requires that you* **CHOOSE** *different approach* **output** `<<TRUE,FALSE>>` **or** `<<FALSE,TRUE>>`

q4 a truth table is a set of rows. For terms with two variables A and B each row is a sequences the three truth values A , B followed the truth value of the term. Build the truth table of: A IMPLIES B. **output**

```
<<TRUE,TRUE,TRUE>>,
<<TRUE,FALSE,FALSE>>,
<<FALSE,TRUE,TRUE>>,
<<FALSE,FALSE,TRUE>>
```

q5 truth table for AND.
q6 truth table for OR .
q7 truth table for NEGATION.
q8 write proposition: *3 is in the range 1 to 4*
q9 the set of all pairs from the range 1 to 4 to the set of truth values *hint: may need "(" and ")"*
output

```
<<1,TRUE>>,<<1,FALSE>>,
<<2,TRUE>>,<<2,FALSE>>,
<<3,TRUE>>,<<3,FALSE>>,
<<4,TRUE>>,<<4,FALSE>>
```

q10 a binary number is a sequence of "1"s and "0"s. And a 4 bit sequence is a function from `1..4` to "1"s and "0"s. To write the four bit binary number for 10.
Write a function from from `1..4` to "1"s and "0"s, so that `i |-> i %2`.
output `<<1, 0, 1, 0>>`

q11 The function from `2..5` to "1"s and "0"s, so that `i |-> i %2`
will return `(2 :> 0 @@ 3 :> 1 @@ 4 :> 0 @@ 5 :> 1)`

q12 Where `b` is a 4 bit binary number. Define an evaluation function `myBitVal(b) ==` that returns the value of . Use the `LET` command and may be best defined using recursion. (Not easy)
Another way to explain this is to write a function that takes a binary (a function from from `1..4` to "1"s and "0"s) and returns its value:
`myBitVal(<<1, 0, 1, 0>>)` evaluates to `10` (0*1 + 1*2+0*4+1*8)

q13 Amend the previous question. Write an evaluation function `myBitV(b) ==` where `b` is a binary number with any number of bits between 1 and 8. Use the `CHOOSE` and `LET` commands, both can be looked up on the referenced tutorial on TLA+

End of Home Work