**Home Work 2**

A TLA+ Module defines a State Machine using both and only Classical Logic and Temporal Logic. We will look at Temporal Logic later but this week you need to understand how you can use First order Classical Logic to define both state and steps that change the state. In the very simple example we consider the state machine is used to simply compute a value and as we can define such a computation as a function the effect of the State machine can be verified by comparing the value it returns with the result of the function.

**Goal - to be able to read and write very simple TLA+ specifications.**

1. understand how actions are defined in first order logic.
2. understand what actions and behaviours are.

The components of the State Machine are the Initial state `Init` and a rule to build the next State `Next`. These components can be put together as a TLA+ specification

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

The exact meaning of this definition will be explained in later lectures but its operational behaviour can easily be explored using the TLC model checker.

**Debugging tips:**

1. use `EXTENDS Naturals, Integers, TLC, Sequences` then you can use print statements like `/\ PrintT(" Rec END" \o ToString(value))`
2. read any error trace
3. set the Model >Additional TLC Options >Features checking the box "Visualise State Graph after completion of model checking"
4. visually check the State Graph.

## 1 Home Work

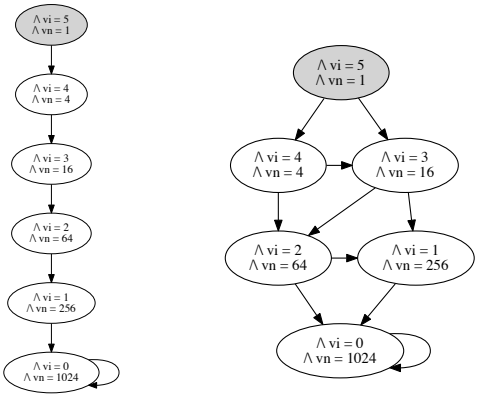
1. Explain `Spec = Init /\ [][Next]_vars` and write a TLA+ specification with a single variable that starts at `0`. With each step add one to the variable and then prints its value. Add a parameter, a `CONSTANT`, and use it to make state machine stop after a number of steps. Build a model that instantiates the parameter and visually inspect the state graph.

## End of Home Work

## 2 Example Specification

1. Define a TLA+ recursive function computing the $power(x, i) = x^i$.
2. Now define a *TLA+ State Machine* using Init, Step that increases the value by $\times N$, Next and Spec to compute the same thing. Only using multiplication and addition and produces the state graph shown below left.
3. Visually check that your definition produces the State Graph shown below.

4. Use the recursive function to check your *TLA+ State Machine* by adding Check to the Invariants set in Model Overview.
5. Add a second action that increases the value by $\times N^2$ and produces the state graph shown below right.



## Model Answer below and in power.pdf

MODULE *power*

Define a TLA+ recursive function computing the $\$power(x,\ i) = x\hat{}i\$$. Now define a "program" using *Init*, *Next* and *Spec* to compute the same thing. check the State Graph by bulding a model and checking the box in the Advanced Options Finally use the recursive function to check your "program" by adding *Check* to the Invariants set in Model Overview.

EXTENDS *Integers*, *Sequences*, *TLC*

CONSTANT $V,\ N$ $N\hat{}V$

RECURSIVE $power(\_,\ \_)$

$power(n,\ i) \triangleq$ IF $i = 0$ THEN $1$
ELSE $n * power(n,\ i-1)$

VARIABLES $vn,\ vi$

$Init \triangleq \land vn = 1$ initialise variables
$\land vi = V$

$Step \triangleq \land vi > 0$ pre condition
$\land vi' = vi - 1$ compute next state of $vi$
$\land vn' = vn * N$ compute next state of $vn$

$StepTwo \triangleq \land vi > 1$ pre condition
$\land vi' = vi - 2$ compute next state of $vi$
$\land vn' = vn * N * N$ compute next state of $vn$

$Result \triangleq \land vi = 0$
$\land vi' = vi$
$\land vn' = vn$
$\land PrintT(ToString(N) \circ \text{"}\hat{}\text{"} \circ ToString(V) \circ \text{" = "} \circ ToString(vn))$ prints $N\hat{}V = vn$

$Next \triangleq StepTwo \land Step \lor Result$ State Machine has two actions

$Spec \triangleq Init \land \Box[Next]_{\langle vi,\ vn\rangle}$ This is the Specification of the State Machine

detailed explaination of this term will come later in the course

$Check \triangleq vi = 0 \Rightarrow vn = power(N,\ V)$ In *TLC* Add to Model Invariant

$FailingInvariant \triangleq vi = 0 \Rightarrow vn = power(V,\ N)$

This simple specification has been defined in two distinct ways. Model checking the execution checks that they agree on the final result. Viewing the State Graph give confidence in the operational behaviour of TLA+. Mutually excelusive pre conditions results in unbranching state graph

* Modification History
* Last modified *Thu Feb* 20 15:56:25 *NZDT* 2020 by *dstr*
* Created *Wed Mar* 20 09:16:40 *NZDT* 2019 by *dstr*