1 Overview of SWEN421

In TLA+ and PlusCal every specification is written in a module. In TLA+ any module is a machine readable Mathematical statement. PlusCal is a high level language designed to be, relatively, easy for programmers to learn. It is translated into TLA+ before being validated. You will need to gain some understanding of both in order to build specifications any where near the level of complexity used in industry.

A Module behaves like a State Machine, that is contains variables that are initialised to some value and labelled steps that change the values stored in the variables. You need to understand the TLA+ syntax that is used to build mathematical definitions, both in TLA+ and the more high level PlusCal. Using this understanding we learn how to define modules - **State Machine**s :

1. variables
2. how to initialise the variables - `Init`
3. steps, or actions, that perform state change Pre Post specifications - `Next`

A behaviour of a state machine is a sequence of states that it may pass through by executing a sequence of the steps defined in it.

The behaviours of a State Machine can be defined by:

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

Where `Next` is a disjunction of TLA+ steps and vars a sequence of variables. In a state from which the precondition of the `Next` action is `FALSE` means the action can not be performed. This is a consequence of the Actions being interpreted as Logic.

You can build models from this specification and run the model checker **TLC** on any of these models. But at this point very little will be checked. This is because `Spec` actually defines the behaviour of the module, the module does not have a separate implementation that can be checked if it satisfies the `Spec`. You can check that the `Spec` behaves in the way you intended by extending the model with temporal properties and with invariants. Plus assert statements can be added to the specification itself. Now running the Model checker all of these will be checked.

A Module may contain a single PlusCal process or a set of processes.

Think of a PlusCal `process` as a *'program'* having its own thread of execution, parameters are passed in when the Model is built when steps are executed there is no direct way to pass parameters.

This is one place where the difference between programming and specifying becomes apparent. A step of a process has no parameters passed to it yet using the `with` clause the behaviour of the step is defined over a set of possible values, just like for a method with a parameter.

The only interaction between two processes is via shared state. There are two obvious places when you can not conceptually stay at the more abstract PlusCal level and will need to understand TLA+.

First is the definition of data and functions. Second is the specification of liveness conditions, fairness conditions and of refinement.

It is common to check if an implementation satisfies a specification writen s $Imp \sqsubseteq Spec$. The $\sqsubseteq$ operator is refered to both as implication and a refinement. In TLA+ you

only have specifications hence a Modules uses `Spec` to define it's behaviour. To get the most out of the model checker TLC you need to define both `Spec` and properties that you wish to check.

In addition in TLA+ you can define two separate modules one for Imp the other for $Spec$. In TLA+ for one module to be a refinement of another, $M_{Imp} \sqsubseteq M_{Spec}$ is calculated using its logical interpretation as implication $M_{Imp} \implies M_{Spec}$ that is by model checking that M_{Spec} is an invariant of M_{Imp} To be able to do this you need a good understanding of how one module can be imported into another.

Basic concepts

These concepts are used outside of TLA+ but is is essential to understand both the concept itself and how it is used in TLA+.

Safety properties - do nothing wrong.

Liveness properties - do some thing right.

Weak fairness - if an event is offered forever then it must eventually be taken.

Strong fairness - if an event is offered infinitely often then it must eventually be taken.

The difference is that weak fairness only applies if an event is offered without a break where as strong fairness applies even if there are breaks.

TLA+ allows you to specify temporal properties $TempProp$ and the TLC model checker works by building a finite state model of the TLA+ module and checking a property for every state of the model.

To establish that `SM1` $\sqsubseteq$ `SM2`, a refinement between two TLA+ state machines, we will need to:

1. build a relation between the states of the two machines using the `WITH` clause of the `S2==INSTANCE SM2 WITH` command in the `SM1` module.
2. use TLC to model check that the module `SM1` satisfies the `Spec` of `SM2` by adding `S2!Spec` to the temporal properties of a model of the `SM1` module

Why Specify

Getting the design write may save a vast amount of development and testing time. Users have a very abstract view of the system that may be restricted to a set of requirements. Lists of requirements frequently have many missing corner cases and may contain internal contradictions.

The early ability to analyse the requirements and seek clarification can save a lot time and money.

What to Specify. You can never specify everything. So be aware of your selection: choose the fragment of the system you think either most likely to have bugs in or have bugs that will be most costly to correct. Having selected the fragment you wish to specify then select the level of detail at which to model the system.

In TLA+ an important decision is: what are the **atomic** steps of your system. Atomicity is particularly important in concurrent or distributed systems. On occasions it is

worth specifying the same system more than once, each specification using atomic steps of different size, and then showing one specification to be a refinement of the other.

Specification template

First pick the variables you need to represent the state of the system, when you build a model some may need to become parameters or `CONSTANTS`. Decide upon `Init`, the initial state of the system.

Second select the steps, `Next`, the system takes and the effect they have on the system state. What may appear to be a single step you may later have to decompose into several steps.

When both the variables and the steps have been specified you can build a Model and check it properties. At this point you are likely to have to refactor both the variables and steps you selected. Once your model can be processed by TLC you should check for simple errors like it deadlocking when it should not or its state space being unexpectedly large or even unexpectedly small. To increase your confidence in your specification add invariants to be checked, these should include both type invariants and temporal properties.