Refinement

Refinement allows us to define a formal relationship between a higher level specification and a lower level specification. This can be viewed as an "implements" relationship i.e. a lower level specification implements a higher level specification. In TLA+, for example, a lower level specification implements a higher level specification if it satisfies the same specification. It's quite easy to express this concept formally in TLA+: it's just logical implication. So, if we have a high level spec H and a low level spec L then the expression:

$$L \Rightarrow H$$

can be interpreted as meaning L implements H. In other words, every behavior of L satisfies the specification H. Put another way, every step in L is a valid step in H.

Refinement Mappings

This isn't quite the whole picture, though. In the simplest example, the lower level spec might be identical to the higher level spec but just add some variables. For example, consider the hour clock that has a "hour" and a "minute" hand. The high level spec might only model the `hour` but the lower level spec also models the `minute`. Every "minute" tick in the lower level spec L, then, is trivially a valid step of of the higher level spec H, because it leaves `hour` unchanged, which is valid since stuttering is always allowed. There might be cases, however, where the correspondence between lower level variables and higher level variables is not as obvious.

For example, maybe the low level spec tries to model a clock with minute level precision differently. Instead of storing an `hour` and a `minute` variable separately, maybe it just stores a single `minute` variable which counts the total number of minutes since the beginning of the day. So, for example, when a normal clock might read 1:30 AM, this clock might record that state as `minute=90` (since 90 minutes is 1.5 hours past midnight). The same information is being stored, just in a different representation. In this case, though, the lower level spec only has a single variable `minute`, in contrast to the `hour` variable in the higher level spec. Thus, we need to define some function that describes how a lower level state maps to a higher level state. This will just be some state function that depends on the variables in the lower level spec. In our

case, we can describe this function as `hour = minute / 60` (using integral division). This is the notion of a **refinement mapping** i.e. how do states in a lower level specification map to states in a higher level specification.

Checking Refinement with TLC

In TLA+ we can check refinement using TLC. Say we have a lower level spec `MinuteClockCompact` which models a high level `HourClock` spec using a single `minute` variable. To check refinement we can just ask TLC if every behavior of `MinuteClockCompact` satisfies the spec of `HourClock`, after applying the refinement mapping. We can instantiate an `HourClock` module with our own, defined substitutions, which is how we define the refinement mapping. For example, consider the following expression:

```
\* Our refinement mapping.
V == INSTANCE HourClock WITH hour <- (minute \div 60)
```

It instantiates an instance of the `HourClock` module but substitutes all references to `hour` with the expression `(minute \div 60)`. So, for example, the next state relation in the instantiated module `V` should be:

```
Next ==
    /\ hour' = ((minute \div 60) + 1) % 12
```

after substitution, where it was originally:

```
Next ==
    /\ hour' = (hour + 1) % 12
```

We can then check if `MinuteClockCompact` refines `HourClock` by checking `V!Spec` as a property in TLC. This is asserting that every behavior of `MinuteClockCompact` satisfies `HourClock` under the refinement mapping.

Hiding and Refinement

In general, we can say that any specification consists of some set of "observable" variables and

some set of "internal" variables. The values of the observable variables are what we (or the client of a system) really cares about. The "internal" variables are more like the "gears" that are internal to the system implementation. So, the philosophically "correct" way to write a spec is as follows:

$$\exists v_1, \ldots, v_n : Spec$$

where the $\exists$ symbol above is used to represent the "temporal existential quantifier". A formula $\exists x : Spec$ is true of any behavior if there exists a sequence of values we can assign to the variable x in that behavior that make $Spec$ true of that behavior. So, if we write the following:

$$Spec_2 \Rightarrow \exists v_1, \ldots, v_n : Spec_1$$

this is a statement that says if $Spec_2$ holds true of a behavior, then there exists some sequence of values that we can assign to $v_1, \ldots, v_n$ that make $Spec_1$ true of the behavior. This is how we define refinement. In this case it means that $Spec_2$ refines/implements $Spec_1$. In other words, $Spec_1$ and $Spec_2$ admit the same sequences of externally observable values. More generally, we can express refinement as:

$$\exists u_1, \ldots, u_n : Spec_2 \Rightarrow \exists v_1, \ldots, v_n : Spec_1$$

where $u_1, \ldots, u_n$ are the internal variables of $Spec_2$ and $v_1, \ldots v_n$ are the internal variables of $Spec_1$. If we only view the externally observable values of each spec then they admit the same behaviors.

Lamport points out, however, that this philosophically satisfying approach is not always useful in practice, since, for example, two specs might not be described at the same level of abstraction. In the formula above, we assumed that the externally observable variables of $Spec_2$ are given the exact same names as the observable variables of $Spec_1$. This might not always be the case. Therefore, the more reasonable way to express refinement is with a refinement mapping. In this view of things, we don't make a philosophical distinction between "internal" and "observable" variables. We just say that a spec has some set of variables and we can optiionally think of some as internal or external. Then, our refinement mapping just needs to determine how all the variables of a lower level spec map onto the variables of a higher level spec. If all variables have different names, for example, then this mapping could be arbitrary. It is a generalization of the definition with "hiding" using the temporal existential quantifier.

Questions/Thoughts

- Is using the technique of checking refinement better than just checking the necessary properties of the lower level spec? It doesn't seem cheaper since you would still have to test that every behavior of the lower level spec satisfies the higher level spec. If you prove a certain property about the higher level algorithm, though, you get to use it for free in the lower level algorithm, without explicitly re-proving it?
- Using refinement might teach you something about how to abstract from a real system to a high level spec. From that perspective it may be a good exercise in understanding your system from a conceptual/abstract point of view.