CS412 Lab session 8

More refinement machines

This lab continues with more about the use of refinement machines in the B tool. This week, we've been looking at things from the perspective of proof obligations for refinement and the lab gets you to explore the material in this week's exercise sheet.

The refinement relationship

We've encountered variations of the *AbSet* machine and possible refinements this week in lectures (and Q1 in Exercises too). The machine and two candidates for refinement are provided on the website. Do you think that these *are* actually refinements? Confirm your answers by investigating these machines in the toolkit. Investigate using the prover to understand where POs fail and why. Make sure that your suggested corrections really do work. (NB: the prover doesn't contain enough rules about iseq to prove POs relating to this, but as in previous weeks, examining the POs should help you check.)
Now add a further operation, *set_del* to *AbSet* allowing an element given as input to be deleted. Can either or both of the refinements be altered to take account of this (without adding variables or changing their type)? Write the machine(s) where possible and check them out.

A less obvious example

In many cases, the operations of the refinement machine are defined in a very similar way to those in the abstract machine, but in some cases the relationship isn't quite so obvious or so apparently correct. What is important is whether the behaviour is correct from a user's perspective (remember - the user can't observe the state directly). This is what the refinement rules check. An example is the *Colours* machine (given on the website and more discussion in Schneider) and the suggested refinement *ColoursR*. This is a somewhat extreme example in which the use of nondeterminism tests the refinement rules to the limit!

Look at the behaviour of each of these machines and notice the linking invariant. Consider this behaviour from a user's point of view and how it relates to the general objectives of refinement. What pogs do you expect the tool to generate for this refinement?

Check out *Colours* and *ColoursR* in the toolkit, generating and examining the associated proof obligations. (Warning: I found previously that the prover had a bug which meant it didn't translate properly between a set and its own internal representation of the set in order to prove obviously correct POs. For example, in proving the correctness of the Colours abstract machine it was previously getting confused about *col* and trying to say it was a mapping rather than a set. When I altered the type of cols in the spec to be subset of ran(COLOUR)

the typechecker quite rightly complained. But weirdly when I changed it back to the original - it then type-checked and autoprovesed Very annoying and I hope now fixed, but you might try something similar if you're having problems. Or just work on the exercises by hand)
The prover will probably not be able to cope with the proofs involving ∃. You may want to just inspect them for now as they are provable but a little tricky. If you want to try to prove them here is a hint: if your goal is in the form of an ∃ you can instantiate the variable with se(value) where value is the term you are suggesting will satisfy it.

Suggest an additional operation which would make the *ColoursR* state inadequate. You can investigate in the prover to confirm that your operation generates a refinement obligation which is false.

Exercise sheet 8 suggests some further explorations with the *Colours* machine (questions 2 and 3). Work through these by hand first using the proof obligations from lectures. Then test out your understanding by using the toolkit to specify and examine the abstract machine and its refinement. Use the prover to investigate the problem with pogs that don't prove.

Remember that the toolkit prover won't necessarily be able to prove all obligations - but this may be because of its limitations rather than the obligations being false. Answering the exercises by hand first will give you a feel for the refinement conditions and you'll know what you are expecting to be a valid refinement. Check this matches with what the prover tells you (exploring failed conditions in interactive mode).