CS412 Lab session 6

Structuring

1 Practice with structuring

A Warehouse system

Warehouse machine

First, define an abstract machine *Warehouse* with variable:

$$stock : ITEM \rightarrow \mathbb{N}$$

The stock function records the number currently in stock of each item. Define the following operations:

name	**inputs**	**description**	**outputs**
add_stock	*ii nn*	adds *nn* of item *ii* to the stock	
remove_stock	*ii nn*	removes *nn* of item *ii* from the stock	
stock_query	*ii nn*	queries whether we have at least *nn* of item *ii* in stock and returns a Boolean	*bb*

Write this machine, animate to check its behaviour and check consistency in the prover.

Orders machine

Next, do the same for a separate machine, *Orders* which stores information about orders which have been received. When an order is serviced, the information about that order is removed. The machine keeps the information in 2 functions: one recording the item associated with each order and the other the amount. These variables are:

$$orderitem : OID \nrightarrow ITEM$$
$$orderamt : OID \nrightarrow \mathbb{N}_1$$

and they have have the same domain. The operations are:

name	**inputs**	**description**	**outputs**
place_order	*ii nn*	chooses a fresh identifier and adds new mappings to orderitem and orderamt	
service_order	*oo*	removes *oo* from the orders	

Again, make sure your machine is working properly.

Combined machine

What needs to be done to incorporate both of the previous machines in a new machine, *Big_Business*? The new machine should allow us to add stock and place orders exactly as before and to perform the same queries. But now we have:

- an order can only be serviced when there is enough stock;
- stock is only removed in conjunction with servicing an order;
- a new query is required to find out which orders could not be currently met.

Write the abstract machine *Big_Business* and animate/verify its behaviour.

2 Guiding the interactive prover

A previous lab sheet discussed how you could investigate proof obligations which don't prove automatically by clicking Ip (interactive prover) and looking at any POs with a red cross next to them. We'll take that a bit further now, firstly, looking at an obligation we are confident about (one that we know the automatic prover can do) and step through in interactive mode. This just gives you some basics. For more information look at the Prover user manual and reference manual.
Consider once again the ProjectsLab3 machine (a correct version is reproduced again below). We know we can prove this automatically (make sure if you like!) so the obligations are all true. (If you have already proved it, within the main B window select it and then click Component → Proof → Unprove to reset.) Now click *Ip* to start the interactive prover.
You can select a PO to work on by clicking it - select PO2 from *signup* which has the goal $projects \cup \{ss \mapsto pp\} : STUDENT \nrightarrow PROJECT$
First, apply deduction by clicking the yellow *dd* button (either from the top tool bar or to the RHS of the command window) - the facts we can work with have now been made into hypotheses and the goal is clearly displayed. To see what hyps we have to play wih, type $sh(a)$ in the command window. We see from the hypotheses that *projects* is a function and *ss* is not in its domain so we should be able to prove the goal.
We need to suggest a rule to apply. We can look at the theory list with applicable rules only shown (make sure the Theory List option is checked in the View options and it's displayed as "View as list"). The names of the theories provide a bit of guidance. As our goal is about membership of a function it makes sense to look in the InFunctionXY theory. It's showing rules 1 and 12 in this theory as being applicable. Look at number one by clicking it. The binhyp means that this is something the tool will be expecting to find in the hypotheses. Basically, if you know something to be functional, then it's also functional on any source u and target v which contain the domain and range respectively. But our expression pattern-matching to f is $projects \cup \{ss \mapsto pp\}$ and we *don't* know that this is a function of any kind. That's the whole point! So this rule doesn't seem helpful. Another clue that it's not what we want is that the subgoals do not appear in our hypotheses - so how could we prove them?
Rule InFunctionXY.12 is specifically for a goal with a union function expression and the subgoals look as if we should be able to prove them. Let's apply this rule by right clicking on the rule name and choosing the $ar(..., Once)$ option. Here, *ar* means "apply rule". Once means apply this rule just once (you can also get it to loop and apply as many times as possible).

You should now see a subgoal displayed:

$projects : STUDENT \nrightarrow PROJECT$

This has been generated as it is one of the antecedents of the rule we've applied. It is in the hypotheses so should be ok. We just need to get the prover to make the step of ticking it off. Clicking the yellow *pr* button telling the automatic prover to try to finish off this step - and it does. You can see this by the fact that the Proof window has marked the branch with a *pr* and the *Next* marker has moved to the next subgoal. This new subgoal now appears as our current goal in the proof window: $\{ss \mapsto pp\} : STUDENT \nrightarrow PROJECT$

This is the 2nd subgoal generated by the rule we applied. The list of applicable rules should have changed accordingly. The goal is again about something being in a function. So let's try looking at the InFunctionXY theory again. Number 1 is applicable again, and also number 17. The latter seems exactly what we want: a goal about adding a single pair. Apply this rule. It shows the new subgoal $ss : STUDENT$ from the antecedents of the rule. We can see this is in the hypotheses, so click *pr* to discharge this subgoal. The next subgoal is now displayed: $pp : PROJECT$. Another hypothesis - click *pr* again. The 3rd subgoal from the original InFunctionXY.12 is now displayed:

$dom(projects) \lhd \{ss \mapsto pp\} = dom(\{ss \mapsto pp\}) \lhd projects$

This could get very confusing if we had to keep track as we have to nest the proofs as we consider each subgoal which may itself generate subgoals etc etc. However, the prover is keeping track of what has and hasn't been discharged and will present us with the next step. Again, our new goal looks proveable and we can look for a helpful rule in a suitably-named theory. Another way to do it might be to search the theory list to locate a term of interest. For example, the RHS of our goal includes the term $dom(\{ss \mapsto pp\})$. We know this should simplify (what to?!) - but is there a rule for it in the rule base and if so where? In the command line window (in the middle) we can type:

$sr(All, Goal)$ To search all theories for rules which match our goal. That sometimes gives a lot of rules, so you can also narrow down the search to rules which mention a certain term rather than match the whole goal:

$sr(All, dom(\{a \mapsto b\}))$

The a and b are wildcards and will match to anything. Either of these searches will be ok here. In this case the bottom rule displayed in the "Search rule result" window is SimplifyRelDomXY.21 and it's a rewrite rule that looks like what we need.

To apply this you can go back to the command line window and type:

$ar(SimplifyRelDomXY.21, Goal)$

Before, we just clicked on the rule itself and chose one of the *ar* (apply rule) options. But we can also do it explicitly via the command line. Because this is a rewrite rule the 2nd parameter is interpreted differently and Goal means apply the rewrite in the Goal.

The RHS of the goal should now have reduced to $\{ss\} \lhd projects$. But remember that *ss* isn't in the domain of *projects* (this was in the precondition of our *signup* operation and is hence now a hypothesis) so we would expect the RHS to reduce to $\{\}$. Again, we need to see if a rule exists to help us with this. Typing $sr(All, \{a\} \lhd b)$ gives us lots of rules. We could cut it down by also looking for something we're expecting to occur in the antecedents, such as $not(a : dom(b))$.

To do this we write: $sr(All, \{a\} \lhd b, not(a : dom(b)))$ and you should find two responses.

Q1 One is helpful - which one? Identify it and apply it. (If it's in the "View as List" window you can apply by clicking. But you can also apply by hand by typing $ar(rulename, Goal)$) Notice that this concludes this branch of the proof - no further subgoals are generated.

This is because the rule has been written using a "binhyp" statement which means the antecendent must be present in the hypotheses and is then immediately ticked off. The other 2 antecendents prevent capture of free variables.
Q2 Find a suitable rule for making further progress with the proof and discharge this PO completely.
When you have done this the PO will appear highlighted in green in the proof window and a green tick appears agaisnt that PO in the situation window.

Different options in the interactive prover

Above, we were doing things by hand to get the idea of finding rules and applying them. However, at various points in this proof we could have pressed *pr* and the automatic prover could have discharged the step we were looking at. In general (if you weren't doing this for practice) you'd always want to try this at each step. You would suggest a manual step to get over a sticking point - then see if *pr* can continue. In more complicated proofs, you can also try the "quick" version - *pr*(*Red*) - it doesn't try quite so hard (doesn't attempt case analysis) and is less likely to get stuck.

The predicate prover

You may have noticed the *pp* buttons in the prover. These refer to the predicate prover. It is part of the automatic prover which deals with statements as pure logic. For example, if you are trying to prove:
$(\mathit{foo}(x) : S \wedge (\mathit{baa}(x,y) \Rightarrow \neg\, \mathit{foo}(x) : S)) \Rightarrow \neg\, \mathit{baa}(x,y)$
you don't need to know anything about *foo* or *baa* to know that this is true because it's an instance of the tautology: $(p \wedge (q \Rightarrow \neg\, p)) \Rightarrow \neg\, q$. If you're working through an interactive proof and you get to a step which is justified by pure logic, the quickest way to discharge it is probably calling the predicate prover. In this case, the goal is true on its own - the predicate prover doesn't need to look at any hypotheses. In this case, just click pp_0. If it's a case where it will need to use something in the hyps, click pp_1

Set simplification

Another button to try if you're doing an interactive proof and have a set expression you think may simplify easily then try clicking *ss*.

Adding an hypothesis

Sometimes you can see that there is a useful fact that follows from the hypothesis which would be very helpful in working towards proving your goal. You can add this fact (let's call it H) by typing: $ah(H)$ You will first be asked to prove that this really does follow from the hypotheses (that is, H will become your subgoal). Once you've proved this, your original goal, G say, is then ammended to $H \Rightarrow G$. You can press *dd* top get H added to the hyps list and then it can be used as necessary.

Case analysis

You may be able to prove a result by showing that the goal you want follows from both P and $\neg\, P$ for a particular choice of predicate P. To try this type $dc(P)$ and the tool will generate

both branches of the proof and get you to discharge them one after the other.
Sometimes you know that a variable must take one of a limited number of values and you can show that your goal would follow from each of these. You can then use the command format $dc(xx, E)$ to set this up. For example, if you had a hypothesis $mm : 0, 1$ and you had the goal $mm \neq 3$. Using $dc(mm, 0, 1)$ allows you to show the result for each of the two possible cases.

Forcing use of a known equality

You may have an equality, $a = E$ in the hypotheses which you want to apply, either in the goal or to substitute in one or more of the other hyps. To apply in the goal use: $eh(a, E, Goal)$ For the other options use AllHyp and Hyp(h) instead of Goal.

Where this gets us

Working in this way allows us to work with the interactive prover when a PO does not prove automatically. It is useful in exploring a PO to break it down and see if we believe it should prove or whether there is a mistake in our spec.
In some cases where the prover is not clever enough to work out a strategy we may be able to point it in the right direction and thus prove the goal. You need to have a strategy in mind. How would you be proving this goal if you were doing it by hand? How can you get the tool to do that?
In other cases there may be true POs for which the current rule base is not sufficient. To prove them we need to add rules, but we'll leave that for now.
Q3 Go back to the signup op and use Ip to discharge the remaining 2 obligations. This is obviously still for practice because we know they autoprove, so don't use the pr or pp button (except if you want to tick off a goal that appears in the hyps or is an obvious truth like x=x or btrue). At each step, find and apply an appropriate rule.

Summary of some of the prover command window instructions

For full details of options available consult the online B prover manuals.

pr call the automatic prover to see if it can now make progress

pr(Red) (equivalent to pressing the *mp* button) call the prover, but don't let it attempt any case analysis which can sometimes pursue blind alleys.

sh(xx) show all hypotheses relating to xx

sh(a) show all hypotheses

sr(where,whatingoal,whatinantecedents) search for matching rules. See details on this is the reference manual. A useful basic one is sr(All,Goal). If you can guess which theory it's in, you can reduce the search by using the theory name isntead of All.

ar(rn, mode) apply rule - rn is the rule name; mode can be Once or Many. If it's a rewrite rule, setting mode to Goal tells it to look apply in the goal.

ah(hyp) add hypothesis - but you'll have to prove it!

eh(a,E,X) where X is one of Goal, AllHyp, Hyp(h). Will force application of the equality $a = E$ to the specified target. Must have $a = E$ in the hypotheses.

ProjectsLab3 machine

```
MACHINE          ProjectsLab3

SETS             STUDENT; PROJECT

CONSTANTS        maxmark, MARK

PROPERTIES       maxmark : NAT1 & MARK = 0..maxmark

VARIABLES        students, projects, marks

INVARIANT        students <: STUDENT & projects : STUDENT +-> PROJECT & dom(projects) = students &
                 marks : STUDENT +-> MARK & dom(marks) <: students

INITIALISATION   students := {} || projects := {} || marks := {}

OPERATIONS
signup(ss,pp) =
  PRE
     ss:STUDENT & pp:PROJECT  & ss /: dom(projects)
  THEN
     students := students \/ {ss} || projects := projects \/{ss |-> pp}
  END;

pp <-- assignproject(ss) =
  PRE  ss:STUDENT
  THEN
     ANY ppx
       WHERE ppx : PROJECT
       THEN pp := ppx
            || students := students \/ {ss}
            || projects(ss) := ppx
       END
  END;

pp <-- queryproject(ss) =
  PRE
    ss:STUDENT & ss : dom(projects)
  THEN
    pp := projects(ss)
  END;

 entermark(ss,mm) =
   PRE
     ss:STUDENT & mm:MARK & ss : dom(projects)
   THEN
     marks(ss) := mm
   END;

mm <-- querymark(ss) =
  PRE
    ss:STUDENT & ss : dom(marks)
  THEN
    mm := marks(ss)
  END
END
```