Friday Week 9.

Exercise 1

Work through the tutorial guide. In the "More Things to Try" section, try at least 1, 2, 5 and 6. For each proof you do:

1. Go back to the top of the proof tree and take a snapshot of your proof. Hand in your snapshots.
2. Save the theorem, using save_thm, in files called e1.proof, e2.proof etc and tell me where they are.

Exercise 2

Try proving some of the theorems in ~jane/oyster/bmlib

In ~jane/oyster/tactics there are various files with tactics in them. Load them into oyster, and you will be able to use the tactics in them. Feel free to copy the files. Don't worry about warning messages about singleton variables.

Try it out on the associativity of plus proof. There are certain things which you must do first in this system. For each theorem, at the top, the prolog goal:

```
?- set__wfftac,autotac,down,reclemmas.
```

This sets up an appropriate autotactic for the theorem, so you don't get bothered by too many well-formedness goals, and then it unravels the definitions to give you hypotheses corresponding to the base and step parts of each definition. these are more accessible to the tactics than the definition itself. This may take a moment. The SUN2s are a bit slow.

Then if you go down to the next goal, you should see something like this:

```
| ?- down,display.
a : [1,1] incomplete autotactic(repeat wfftac)
1. append(l,m)<==>list_ind(l,m,[t,~,v,t::v])
2. v0:m:pnat list->append(nil,m)=m in pnat list
3. v1:(v0:pnat->
         l:(pnat list)->
           m:(pnat list)->
             append(v0::l,m)=v0::append(l,m) in pnat list)
>> x:(pnat list)->
    y:(pnat list)->
      z:(pnat list)->
        append(x,append(y,z))=append(append(x,y),z) in pnat list
by _
```

Which is what you really want to work on. First, get rid of all the universal quantifiers, there is a tactic called **dequantify** to do this. from then on down, use these tactics and positioning commands to prove the theorem:

- **induction(_)**. This performs an induction on whatever variable you fill the blank in with, and sets up subgoals for the base and step cases.
- **takeout** This finds an expression involving a 0 or *nil* in a definitional argument position and does the appropriate rewrite, simplifying the term. It clears up any subgoals justifying the rewrite.
- **unfold** This selects a term which can be rewritten in terms of its step case definition, and rewrites it. e.g. $append(x' :: x, y) \mapsto x' :: append(x, y)$
- **prop** which should handle any goals which are propositionally true.

- **fertilise(_)** will use the named hypothesis to do a fertilisation for you. It works out what the hypothesis can be applied to.

Then try either exercise 3 or 4:

Exercise 3

Write a propositional contradiction finding tactic, as suggested at the end of the tutorial guide. So that it can detect that it has two hypotheses of the form

```
v1:a
v2:a->void
```

doesn't do..

and use the appropriate *elim* and *hyp* inferences to complete the branch of the proof.

Exercise 4 — Tuesday next.

Synthesise *plus* and use your synthesised function to add some numbers.

v0: (x: a|p of x→void)→void

v1: p of x→void.