APL : Sheet One

(1) Unlike TLC, if you did it, "upside down" derivation trees will not be allowed.

If you did TLC, you will be familiar with the derivation game and how it is all about applying the correct and matching rules.

Here is a little recipe for doing these:

1. Remind yourself of the rules
2. Write down what you want to prove
3. Until you hit axioms and have dealt with all proof obligations:
   - select rule (with premise that matches what you wanna prove
   - apply it

Odd/Even Rules:

Because the concepts of odd and even are defined using rules that depend on each other, this is an example of "simultaneous generation of judgements".

$$\text{EVENZ}\ \frac{}{\text{zero even}}$$

This "axiom", called so as it has no premises, tells us that zero is even.

$$\text{ODD}\ \frac{n\ \underline{\text{even}}}{\text{succ}(n)\ \underline{\text{odd}}}$$

If the predecessor of something is even we know it is odd

$$\text{EVEN}\ \frac{n\ \underline{\text{odd}}}{\text{succ}(n)\ \underline{\text{even}}}$$

If the predecessor of something is odd we know it is even

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\text{zero}\ \underline{\text{even}}}\ \text{EVENZ}}{\text{succ}(\text{zero})\ \underline{\text{odd}}}\ \text{ODD}}{\text{succ}(\text{succ}(\text{zero}))\ \underline{\text{even}}}\ \text{EVEN}}{\text{succ}(\text{succ}(\text{succ}(\text{zero})))\ \underline{\text{odd}}}\ \text{ODD}$$

↳ "3 is odd"

The rule naming here is super helpful because to prove something is odd, we need to reach for the odd rule.

② i)

I'm a Haskell gal, and Alex already drew conversions between rules and Haskell data types, so I'll start with what I know:

```
> data NatList
>     = Empty
>     | Cons Nat NatList
```

=> I'll need two rules for list of nats, and I'll need to call out to our nat rules

Let's do the easy one first:

$$\frac{}{[]\ \underline{\text{natlist}}}\ \text{EMPTY}$$

same as consNeter

Haskell syntax

what I've called my judgement / type

Now for the more complicated one. The consNeter has two args, so I will need two premises (just the succ needed one):

$$\frac{n\ \underline{\text{nat}} \qquad ns\ \underline{\text{natlist}}}{n : ns\ \underline{\text{natlist}}}\ \text{CONS}$$

Seems fit. We can test it by seeing if we can show zero : [] is a nat list:

$$\frac{\dfrac{}{\text{zero}\ \underline{\text{nat}}}\ Z \qquad \dfrac{}{[]\ \underline{\text{natlist}}}\ \text{EMPTY}}{\text{zero} : []\ \underline{\text{natlist}}}\ \text{CONS}$$

ii)

This will be very similar to the induction rule for natural numbers:

Let P be a property of natural numbers.
If
- $P(zero)$
- whenever $P(n)$, $P(succ(n))$ holds

then $P(n)$ holds for all n nat.

It seems that an induction principle needs a proof obligation for each rule, and that if a rule has premises we can assume them (induction hypothesis) to prove the conclusion.

Now let's adjust this for lists:

Let P be a property of lists of natural numbers.
If
- $P([])$ -- empty
- whenever n nat and $P(ns)$, $P(n:ns)$ holds -- cons

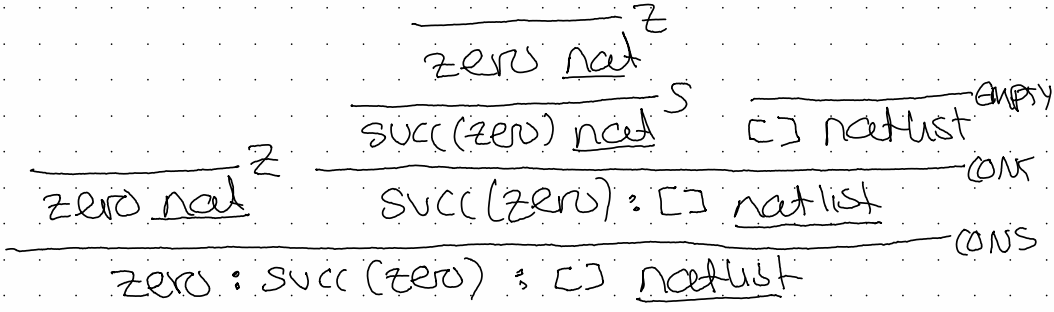
then $P(ns)$ holds for all ns natlist.

obligation points

assume premises

iii) Here we already almost did this but with a simpler example

This is the same game as (i) but with our rules



"[0, 1]"

(3)

Derivable means that the derivation of the rule ends in its premise

Our "derivable" rule is

$$\frac{n \text{ even}}{succ(succ(n)) \text{ even}}$$

So we expect our derivation of succ(succ(n)) to end up being a derivation for n even.

Let's write our conclusion, apply the rules and see if this happens:

$$\dfrac{\dfrac{n \text{ even}}{succ(n) \text{ odd}}\ \text{ODD}}{succ(succ(n)) \text{ even}}\ \text{EVEN}$$

Yay it did! This means it is indeed derivable, and by presenting the derivation steps between the conclusion and the premise, we have shown that.

(4)

To show that a rule is admissable, we need to be able to take a derivation of its premises and be able to construct a derivation of the conclusion from it.

This question asks us to do this for

$$\frac{n\ \underline{\text{even}}}{n\ \underline{\text{nat}}}$$

"if something is even, then it is a nat"

Let's run some examples and see if we can spot a pattern for creating a derivation of n nat from n even

When $n = \text{zero}$ ...

derivation of premise: $\underline{\text{zero even}}$ EVENZ

derivation of conclusion: $\underline{\text{zero nat}}$ Z

Well, in this case we just need to perform some renaming:

even $\rightsquigarrow$ nat

EVENZ $\rightsquigarrow$ Z

What about succ(succ(even))?

$$\frac{}{\text{zero even}}\ \text{EVENZ} \qquad \frac{\text{succ(zero) odd}}{\text{succ(succ(zero)) even}}\ \text{EVEN} \qquad \frac{}{\text{zero nat}}\ \text{Z} \qquad \frac{\text{succ(zero) nat}}{\text{succ(succ(zero)) nat}}\ \text{S}$$

Nice! It seems like we just swap out ODDs and EVEN's for Ss.

This rule is admissable because we can construct a derivation of the conclusion by performing the following relabellings on the derivation of the premise:

EVENZ	→	Z
ODD	→	S
EVEN	→	S
odd	→	nat
even	→	nat

WAIT WAIT WAIT

Something fishy is going on here. Nothing in our premise says we can work with odd numbers

⇒ We need to strengthen the statement to accommodate odd numbers by adding

$$\frac{n\ \text{odd}}{n\ \text{nat}}$$

Now that was my thought process and how I worked this out but we should present this more formally.

Really what is happening here is proof by mutual induction of n odd and n even as

the rewrite rules are covering what to do case by case.

Let's now present our answer more formally:

The following rules are admissible:

$$\frac{n\ \underline{even}}{n\ \underline{nat}} \qquad \frac{n\ \underline{odd}}{n\ \underline{nat}}$$

Proof by mutual induction on n *even* and n *odd*

[Case: EVENZ]

=> n is zero, which is trivially a *nat*

$$\frac{}{zero\ \underline{nat}}\ Z$$ BEVENZ

[Case: EVEN]

IH = if n *odd* then n *nat*

The even rule tells us the shape of what we must show to be a nat:

$$\frac{x\ \underline{odd}}{succ(succ(x))\ \underline{even}}\ EVEN$$

By IH x is a nat. (1)

Now we must show succ(succ(x)) to be a nat

$$\frac{\dfrac{}{x}\ (1)}{succ(succ(x))}\ S$$

BEVEN

[Case: ODD] Analogous to EVEN case. □

Proof by induction recipe:

1. Decide who to induct on.
   - In general this is the most relevant judgement that you have all components for
   - When you have a choice between two of the same type, pick the one with the most interesting rule in what you are trying to prove
2. Take the proof case by case; one case per rule
   - remember to assume your premises
   - write down the IH if it is a recursive rule.

If you are stuck, write out the facts you know in front of you, and consider in turn if any can help.

(3i) Here we go again! This is a great way to acquaint yourself with semantics.

$$\dfrac{\dfrac{\dfrac{}{zero\ nat}\,Z}{succ(zero)\ nat}\,S \qquad \dfrac{}{sum(zero, succ(zero), succ(zero))}\,BASE}{sum(succ(zero), succ(zero), succ(succ(zero)))}\,IND$$

1 + 1 = 2 :D

ii)

```
> sum :: Nat → Nat → Nat
> sum zero     b = b                    --BASE
> sum (succ a) b = succ (sum a b)       --IND
```

Yes Hyb uses pattern matching and you can almost think of Nat as deciding which rule you apply.

One case per rule. BASE is basically identical but IND requires a recursive call.

iii)

This smells like a proof by induction! Why? Well, so far in the course what is how you have been taught to prove things, this is the judgement / induction sheet after all, and finally not on a course context note, the things we are working with are defined using rules that come with induction principles.

The next question, is what do we induct on? The sum judgement seems like the best choice as it encompasses our premise.

Proof by induction on sum.

[Case: BASE]

$$\frac{b\ \underline{nat}}{sum(zero, b, b)}$$

remember we get to assume the premises

Zero is a nat by the zero rule:

$$\frac{}{zero\ \underline{nat}}\ Z$$

b is a nat as given by a premise.

Thus all three args to sum are nats

$\square_{BASE}$

[Case: IND]

$$\frac{sum(a, b, c)}{sum(succ(a), b, succ(c))}\ IND$$

IH

IH = a, b, c are nats.

This leaves us with obligations to show that succ(a) and succ(c) are nats.

This is done using the succ rule and the fact that a and c are nats from the IH.

□ IND

D

(iv)

Remember sum(a,b,c) ~ a+b=c.
So inherently we know there will be a c. Furthermore, if we see the judgement sum as specifying a function then better we'll be a uniquely defined output c.

But how to prove this?

We wanna show that for any a nat b nat there is a unique c nat

When it comes to showing that something is the case for all types defined by rules we always reach for our favourite tool: induction!

But we can't induct on a and b. This is science! Only change one variable at a time!

To know how to pick one to induct on, just look at the rules and see which has the most interesting stuff done in them

e.g. In sum b remains unchanged, so inducting on them won't tell us much.

a on the other hand is where the action is at!

Proof by induction on a nat.

[Case: ZERO]

When a = zero, the BASE rule of sum is triggered.

$$\frac{}{sum(zero, b, b)}$$

a b c

In this case, $c = b$. No need to create any result, we already have it in the form of b!

[Case : SUCC]

This triggers the IND case of sum:

$$\frac{sum(x, b, c')}{sum(succ(x), b, succ(c'))}\ IND$$

IH = there exists c' such that $sum(x, b, c')$

Using that and the IND rule, we can conclude that the c we want for $a = succ(x)$ is $succ(c')$

□ IND

□

(v) Oh gosh we need to prove something again... I wonder how...

INDUCTION!

Since we have all components to talk about sum, we can induct on it

Proof by induction on sum.

[Case: BASE]

In this case $c = b$, so since b doesn't change $c = c'$

$\square_{BASE}$

[Case: IND]

$$\frac{sum(x,b,y)}{sum(succ(x),b,succ(y))}\ IND$$

IH = our statement holds for $sum(x,b,y)$ i.e. if we had $sum(x,b,y')$ $y = y'$

$\Rightarrow c = succ(y)$ and $c' = succ(y')$

which are equal by transitivity of $=$

$c = succ(y) = succ(y') = c'$

(vi)

To answer this, we need to think about what it means to be a function: a function is a unique mapping between inputs and outputs

ah ha!

We have shown outputs to be unique.

Then if we wanted a total function we would need each input to produce an output

From our proofs of existence and uniqueness in (iv) and (v), we can conclude that sum is a total function on nats.

---

the moral of the story is: if in doubt, proof by induction.

things I noticed:

- People seem most comfortable inducting on n nat, but that is not always who we wanna induct on. BEWARE!