# APL: Sheet Three

REMINDER:

Proofs are hard.

Don't worry too much about them. All we want from you is to have a go. Best thing to do is time box them so you don't waste time banging your head against the sheet. Your time is would be better spent understanding the solutions when they come out

If stuck in a proof:

- Make sure you have broken into cases if its by induction
- Do the easier cases first
- Write down all the facts you know that are related to what you are trying to prove e.g. induction hypothesis or the rules
- Take a break.
- Set a time: if you still have no idea by the time it ticks off, just leave it, we will give you feedback on what you have done and you should ensure you understand the solution when it comes!

1)

Here we go again with these trees. This is a staple of PL courses, take time to understand it. The only difference between here and sheet one is that the rules are more complex and plentiful.

(ps There are so many rules I won't write them out - see slides for rules)

(i) $plus(num[1]; num[1]) \mapsto num[2]$

I think of my rule selection as a pattern match, here I will want plus when applied to nums

$$\Rightarrow \text{D-PLUS}\ \frac{n_1 + n_2 = n}{plus(num[n_1]; num[n_2]) \mapsto num[n]}$$

$$\frac{1 + 1 = 2}{plus(num[1]; num[1]) \mapsto num[2]}\ \text{D-PLUS}$$

(ii)

$$\text{E-SE-TM}\ \frac{\text{D-PLUS}\ \dfrac{1 + 1 = 2}{plus(num[1]; num[1]) \mapsto num[2]}}{times(plus(num[1]; num[1]); num[2]) \mapsto times(num[2]; num[2])}$$

iii)

$$\frac{cat(str['a']; str['b'])}{cat(v; str['b'])\ [str['a']/v]}\ subst$$

$$\frac{\dfrac{}{let(str['a']; v.cat(v; str['b'])) \mapsto cat(str['a']; str['b'])}\ \text{D-LET}}{len(let(str['a']; v.cat(v; str['b']))) \mapsto len(cat(str['a']; str['b']))}\ \text{D-LEN-1}$$

② Here we can pick which part of the program to reduce. To reduce we exchange the LHS of a rule for the RHS - the rule is, if you can justify the transition with a free, we can transform.

i) $times(plus(num[1]; num[0]); num[2]) \longmapsto^* num[4]$

times(plus(num[1]; num[0]); num[2])
→ justified by L(i)
times(num[2]; num[2])
→
num[4]

Be careful to not bundle together single steps. What I do is think what I'm intuitively doing, and then double check a rule justifies it.

ii)

times(len(let(str['a']; v. cat(v; str['b']))); num[2])
→ justified by L(ii)
times(len(cat(str['a']; str['b'])); num[2])
→
times(len(str['ab']); num[2])
→
times(num[2]; num[2])
→
num[4]

③ Same thing except at the end we go wow a num/str. This is well typed.

(if we result in such a value)

Its a use case for these reduction bc showing you won't just do it bc a question says so.

(i) let(str['a']; z.plus(len(z); len(z)))

→ plus((len(z); len(z)))[str['a']/z]

= plus(len(str['a']); len(str['a']))

→ plus(num[1]; len(str['a']))

→ plus(num[1]; num[1])

→ num[2] This reduces to a num which is a value type => well typed

$$\frac{2 \in \mathbb{N}}{\text{num}[2]\ \text{val}}\ \text{Val-Num}$$

(ii) let(len(str['a']); z.plus(z; z))

→ let(num[1]; z.plus(z; z))

→ plus(num[1]; num[1])

→ num[2] num => well typed

(iii) plus(let(len(str['a']); z.plus(z; z)); num[1])

→* (ii) plus(num[2]; num[1])

→ num[3] num => well typed

④ Existing rules:

$$\frac{e_1 \mapsto e_1'}{plus(e_1; e_2) \mapsto plus(e_1'; e_2)}\ \text{D-PLUS-1}$$

$$\frac{e_1\ val \quad e_2 \mapsto e_2'}{plus(e_1; e_2) \mapsto plus(e_1; e_2')}\ \text{D-PLUS-2}$$

this guy insists upon full reduction of $e_1$ before working on $\Rightarrow$ swapsies

New rules:

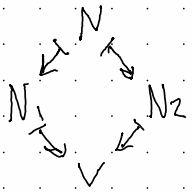
$$\frac{e_2 \mapsto e_2'}{plus(e_1; e_2) \mapsto plus(e_1; e_2')}\ \text{D-PLUS-1'}$$

$$\frac{e_2\ val \quad e_1 \mapsto e_1'}{plus(e_1; e_2) \mapsto plus(e_1'; e_2)}\ \text{D-PLUS-2'}$$

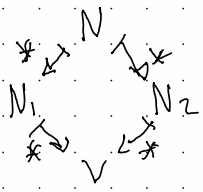
No, this can not affect the final value. It doesn't matter which way you go.

Those that do TLC, this is because $\mapsto$ satisfies the diamond property, meaning that its reflexive transitive closure $\mapsto^*$ satisfies Church-Rosser/confluence $\Rightarrow$ no matter what path, we will reach the same result.

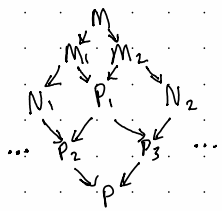
Diamond:




Confluence:




Proof by diagram chase:




you must draw diamonds for diamond prop

Now onto our proofs, which are ALL by induction. So let's remind ourselves of our recipe:

---

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 them if any can help.

---

⑤ Step one: decide who to induct on
We are talking about vals, so induction on e val makes sense

Proof by induction on e val (always announce)

Step two: tackle proof case by case, one case per rule

Rules: (always good to remind ourselves)

$$\frac{n \in \mathbb{N}}{\text{num}[n]\ \text{val}}\ \text{VAL-NUM} \qquad \frac{s \in \Sigma^*}{\text{str}[s]\ \text{val}}\ \text{VAL-STR}$$

(honestly writing the boring stuff helps me think about what next)

[Case: VAL-NUM]

⇒ e has shape num[n]

ASS (premise) = $n \in \mathbb{N}$

AIM = $\vdash e : \text{Num}$ or $\vdash e : \text{Str}$

Our typing rule:

$$\frac{n \in \mathbb{N}}{\vdash \text{num}[n] : \text{Num}}\ \text{Num}$$

By ASS (premise) $\vdash e : \text{Num}$ □ VAL-NUM

Notice how mechanical this is: I follow the recipe, write down relevant things and the proof falls out. All I need to do is connect the dots clearly and the recap/running facts will help.

[Case: VAL-STR]

=> e has shape Str(s)

Ass(premise) = $s \in \Sigma^*$

Aim = $\vdash e : Num$ OR $\vdash e : Str$

Typing rule:

$$\frac{s \in \Sigma^*}{\vdash Str[s] : Str}\ STR$$

By Ass(premise) $\vdash Str[s] : Str$

□ VAL-STR

□

oh look I have what I need to conclude

$\vdash Str[s] : Str$

swell

And that is all the cases

⑥ Here the hint does step one for us :)

Proof by induction on $e_1 \longrightarrow^* e_2$

What is next?

Case-split!

Rules:

$$\frac{}{e \longmapsto^* e}\ \text{D-MULTI-REFL}$$

zero steps is valid

$$\frac{e \longmapsto e' \quad e' \longmapsto^* e''}{e \longmapsto^* e''}\ \text{D-MULTI-STEP}$$

[Case: D-MULTI-REFL]

Remember, to show something is admissable, we must be able to produce a derivation of the conclusion from the premise.

GOAL: $e_1 \longmapsto^* e_3$

From the case we are in, we know the shape of the first premise is

$$e \longmapsto e$$

$\Rightarrow e_1 = e_2$.

GOAL': $e_2 \longmapsto^* e_3$

ASS(other premise) = $e_2 \longmapsto^* e_3$

Given that $e_1 = e_2$ from our case, and our other premise we can derive the conclusion $e_1 \longmapsto^* e_3$ showing this rule to be admissable in this case □D-MULTI-REFL

[Case: D-MULTI-STEP]

⇒ the first premise has shape $e_1 \mapsto^* e_2$ and derivation:

$$\frac{e_1 \mapsto e' \quad e' \mapsto^* e_2}{e_1 \mapsto^* e_2}$$

Giving us: $e_1 \mapsto e'$ (SHAPE1)
$e' \mapsto^* e_2$ (SHAPE2)

IH = the following rule is admissable:

$$\frac{e_1 \mapsto^* e_{12} \quad e_{12} \mapsto^* e_2}{e_1 \mapsto^* e_2}$$

In other words, from $e_1 \mapsto^* e_{12}$ and $e_{12} \mapsto^* e_2$ we can conclude $e_1 \mapsto^* e_2$ for some $e_{12}$

ASS = we have a derivation for $e_2 \mapsto^* e_3$ (ASS)

GOAL: $e_1 \mapsto^* e_3$

$$\frac{\text{SHAPE1}: \quad e_1 \mapsto e' \qquad \text{Applying IH to SHAPE2 and ASS to give us } e' \mapsto^* e_3 \text{: } e' \mapsto e_3}{e_1 \mapsto^* e_3}\ \text{D-MULTI-STEP}$$

Specialised IH

$$\frac{e' \mapsto^* e_2 \ (\text{SHAPE2}) \qquad e_2 \mapsto^* e_3 \ (\text{ASS})}{e' \mapsto e_3}$$

(7)

Preservation: If $\vdash e : \tau$ and $e \mapsto e'$ then $\vdash e' : \tau$

Proof by induction on $e \mapsto e'$.

[Case: D-PLUS]

$$\frac{n_1 + n_2 = n}{plus(num[n_1]; num[n_2]) \mapsto num[n]}\ \text{D-PLUS}$$

ASS1 = $n_1 + n_2 = n$ (premise)
ASS2 = $\vdash e : \tau$ (antecedent)

GOAL = $\vdash e' : T$

In this case $e' = num[n]$

=> we can use the NUM rule

$$\frac{n \in \mathbb{N}}{\vdash num[n] : Num}\ \text{NUM}$$

□ D-PLUS

[Case: D-PLUS-1]

$$\frac{e_1 \mapsto e_1'}{plus(e_1; e_2) \mapsto plus(e_1'; e_2)}\ \text{D-PLUS-1}$$

IH = if $\vdash e_1 : \tau$ and $e_1 \mapsto e_1'$ then $\vdash e_1' : \tau$

ASS1 = $plus(e_1; e_2) : \tau$ (antecedent)

To unlock our IH, we need the type of $e_1$, luckily we can use inversion to tell us that.

From inversion, we know $e_1 : Num$ (INV1)
and $e_2 : Num$ (INV2)

GOAL = plus(e1; e2) : τ

In this case we know τ = Num

GOAL' = plus(e1; e2) : Num

We can provide the IH with INV1 to get $e_1 : Num$ (IH1)

that is all the facts I can think of, so let's try and derive our goal and hope we have the required premises:

$$\frac{\vdots\ IH1 \qquad \vdots\ INV2}{\vdash e_1 : Num \qquad \vdash e_2 : Num}\ PLUS$$
$$\vdash plus(e_1; e_2) : Num$$

[case : D-LET]

$$\frac{\text{[D-PLUS-1]}}{let(e_1; x.e_2) \longmapsto e_2[e_1/x]}\ D\text{-}LET$$

ASS1 = $\vdash let(e_1; x.e_2) : \tau$

GOAL = $e_2[e_1/x]$

Since we want to learn about the type of $e_1$, lets turn to inversion

Inversion tells us $\exists G$ s.t. $\vdash e_1 : G$ and $x : G \vdash e_2 : \tau$

This is just the substitution lemma:

If $\Gamma \vdash e : \tau$ and $\Gamma, x : \tau \vdash u : G$
then $\Gamma \vdash u[e/x] : G$

where we specialise $\Gamma$ to empty, e to $e_1$, etc. [D-LET]

The rest of the cases are just remixes of these.

⑧ if $\vdash e : \tau$ then either eval or $e \mapsto e'$ for some $e'$

Following the theme, this is a proof by induction, so again we will use our recipe.

What is step one?

Decide who we will induct on!

Since we want this to be the case for all $\vdash e : \tau$, makes sense to induct on that.

Proof by induction on $\vdash e : \tau$.

You should know the drill by now, what is next?

Case split!

[Case: Var]

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ \text{VAR}$$

$\Rightarrow$ our context is non-empty.

This means our antecedent ($\vdash e : \tau$) is not fulfilled, so we are done.

□ VAR

[Case: Num]

$$\frac{n \in \mathbb{N}}{\Gamma \vdash \text{num}[n] : \text{Num}}\ \text{NUM}$$

ASS = $n \in \mathbb{N}$
GOAL = eval or $e \mapsto e'$

Since we have the premise (ASS) to show

num[n] val, lets do that:

$$\dfrac{\dfrac{}{n \in \mathbb{N}}\,\text{Ass}}{\text{num}[n]\ \underline{\text{val}}}\,\text{Num}$$

[Case: STR]

Num, but stringified.

[Case: PLUS]

$$\dfrac{\Gamma \vdash e_1 : \text{Num} \qquad \Gamma \vdash e_2 : \text{Num}}{\Gamma \vdash \text{plus}(e_1; e_2) : \text{Num}}\,\text{PLUS}$$

IH1 = either $e_1$ val or $e_1 \mapsto e'$ for some $e'$

IH2 = either $e_2$ val or $e_2 \mapsto e'$ for some $e'$

GOAL = either $\vdash \text{plus}(e_1; e_2) : \tau$ or $\text{plus}(e_1; e_2) \mapsto e'$ for some $e'$.

Focussing on IH1

[Subcase: $e_1$ val]

[Subsubcase: $e_2$ val]

So they are both values and they are both added, inductively plus checks out that they are both nums. Luckily there is a lemma that says exactly that.

By the canonical forms lemma:

1. If $\tau = \text{num}$ then $e = \text{num}[n]$ for some $n$

2. If $\tau = \text{Str}$ then $e = \text{str}[s]$ for some $s$

$e_1 = \text{num}[n_1]$ and $e_2 = \text{num}[n_2]$ for some $n_1, n_2 \in \mathbb{N}$

In other words, in this case

$e = \text{plus}(\text{num}[n_1]; \text{num}[n_2])$

Happily, this is the LHS of an axiom for which we have a $\mapsto$, so we can prove that side of the goal.

$$\text{D-PLUS}\ \frac{}{e = \text{plus}(\text{num}[n_1]; \text{num}[n_2]) \mapsto \text{num}[n_1 + n_2]}$$

$\Rightarrow \exists e'$ s.t. $e \mapsto e'$, specifically $e' = \text{num}[n_1 + n_2]$

D-PLUS $e_2$ val subsubcase

[Subsubcase: $\exists e'$ s.t. $e_2 \mapsto e'$]

Reminder we have:

- $e_1$ val — Ass 1
- $\exists e'$ s.t. $e_2 \mapsto e'$ — Ass 2

and we wanna make a value or a $\mapsto$ thing.

Since we have a $\mapsto$ thing already, we don't gonna make no value.

BUT what we have for this the requirement for D-PLUS-2

$$\text{D-PLUS-2}\ \frac{e_1\ \underline{\text{val}}\ \text{Ass 1} \qquad e_2 \mapsto e'\ \text{Ass 2}}{\text{plus}(e_1; e_2) \mapsto \text{plus}(e_1; e')}$$

$\Rightarrow \exists$ same term namely $plus(e_1; e')$.

□ PLUS subcase $e_2 \mapsto e'$

[Subcase: $\exists e'$ s.t. $e_1 \to e'$]

$ASS = \exists e'$ s.t. $e_1 \to e'$

Looking at our rules again, the case we are in gives us the premise we need to make a $\mapsto$ using:

$$\frac{e_1 \mapsto e'}{plus(e_1; e_2) \mapsto plus(e'; e_2)}\ \text{D-PLUS-1}\quad \text{ASS}$$

□ PLUS subcase $\exists e'$ s.t. $e_1 \to e'$

□ PLUS

[Case: LET]

$$\frac{\Gamma \vdash e_1 : \sigma_1 \qquad \Gamma, x : \sigma_1 \vdash e_2 : \sigma_2}{\Gamma \vdash let(e_1; x.e_2) : \sigma_2}\ \text{LET}$$

Happily this is an easy one, as let terms always reduce with the D-LET axiom:

$$\frac{}{let(e_1; x.e_2) \mapsto e_2[e_1/x]}\ \text{DLET}$$

DLET

The rest are like PLUS.

□

(9) I've got nothing to add to the actual answers - just goes to show how important step one is. Think carefully about what you induction - could make your life easier.