COMP0009 Logic — Problem Sheet 3

Part A — Charlie's new checker

The story so far. On Sheet 1, Alice sent Bob ten coins over a network where nobody trusts anybody. The coins were locked behind a *script*, and every computer on the network, Charlie's among them, ran it on Bob's evidence. Mallory attacked with a script that built an enormous answer, and with Ω, a script that never stops; Charlie's defence was a *typing*, checked before running anything. On Sheet 2 the payments came with *certificates*, lists of lines justified by (I), (II), hypothesis or MP, and every certificate turned out to be a script in disguise. Certificates were easy to check and hard to write, until the Deduction Theorem gave a recipe, "assume A, prove B, discharge A", and a machine that turned the recipe into a real certificate. Run on scripts, the machine was a compiler: it took the placeholders out of definitions.

Now the network upgrades its language. Scripts may keep their placeholders, and they gain data types: pairs, choices, and a type with nothing in it. Certificates are drawn as *trees*, the way logicians have drawn proofs since 1934. In the four scenes below you will read trees, fill them in and draw your own; you will see that every tree is a program, that the data types you know are the words "and", "or" and "not", that running a program removes detours from its tree, and that Mallory's never-stopping script on Sheet 1 was a paradox all along.

Marks are in square brackets; the whole of Part A is worth 25.

A1. The parameter stays.

On Sheet 2, Charlie's checker could not read a definition with a placeholder, such as $\mathit{release}\ x := \mathit{rule1}\ x$. A machine had to compile the placeholder away first, into $\mathsf{S}\,(\mathsf{K}\,\mathit{rule1})\,(\mathsf{S}\,\mathsf{K}\,\mathsf{K})$. In the new language a script may have a parameter, written with λ:

$$\lambda x.\, M \quad \text{is the script that waits for an } x\text{, and then does } M.$$

It is `lambda x: M` in Python and `\x -> M` in Haskell. Application groups to the left, as before, and a λ reaches as far to the right as it can: $\lambda x.\, f\,x\,z$ means $\lambda x.\, (f\,x\,z)$. In types, the arrow groups to the right, as on Sheet 2: $A \to B \to C$ means $A \to (B \to C)$.

There is one rule for running, called β ("beta"):

$$(\lambda x.\, M)\,N \ \rightsquigarrow\ M \text{ with } N \text{ put in place of } x.$$

That is what a function call does: the argument arrives, and takes the parameter's place.

A line of a typing is now written $\Gamma \vdash_\lambda M : A$: "told Γ, the script M has type A". The *context* Γ lists the names we have been told about, each with its type, and each name at most once. With nothing on the left, $\vdash_\lambda M : A$ means that no names are around. Charlie's checker has three rules:

(var)	$\Gamma \vdash_\lambda x : A$	when $x : A$ is in Γ
(app)	$\Gamma \vdash_\lambda M\,N : B$	from earlier lines $\Gamma \vdash_\lambda M : A \to B$ and $\Gamma \vdash_\lambda N : A$
(abs)	$\Gamma \vdash_\lambda \lambda x.\, M : A \to B$	from an earlier line $\Gamma,\ x : A \vdash_\lambda M : B$

(abs) is the new one. It reads a definition the way a compiler does: to check $\lambda x.\, M$, add x to the context with the type it is to have, and check the body M. Then x leaves the context, and becomes the λ.

Example. The script $\lambda x.\, rule1\ x$ is *release*, with the placeholder kept. Its typing, from the rule $rule1 : y \to m$ ("if Alice says yes, the coins may move"), is the list

1. $rule1 : y \to m,\ x : y \vdash_\lambda rule1 : y \to m$ (var)
2. $rule1 : y \to m,\ x : y \vdash_\lambda x : y$ (var)
3. $rule1 : y \to m,\ x : y \vdash_\lambda rule1\ x : m$ (app) 1, 2
4. $rule1 : y \to m \vdash_\lambda \lambda x.\, rule1\ x : y \to m$ (abs) 3

Four lines, and the placeholder stays where it is; on Sheet 2 the machine needed eleven lines to compile it away. Given Alice's yes, $ok : y$, it runs in one step: $(\lambda x.\, rule1\ x)\ ok \rightsquigarrow rule1\ ok$ (ok for x).

The same typing as a tree. Each line becomes a node, and the lines it cites are drawn above it, with a bar between, labelled by the rule:

$$\dfrac{\dfrac{rule1 : y \to m,\ x : y \vdash_\lambda rule1 : y \to m \qquad rule1 : y \to m,\ x : y \vdash_\lambda x : y}{rule1 : y \to m,\ x : y \vdash_\lambda rule1\, x : m}\ (\text{app})}{rule1 : y \to m \vdash_\lambda \lambda x.\, rule1\, x : y \to m}\ (\text{abs})$$

Nothing has changed but the layout: the (var) lines are at the top, and the line at the bottom is the last line of the list.

Rub out the programs. What is left is a proof, each node saying "from these hypotheses, this statement". The rules get their logic names: (var) becomes (hyp), (app) becomes (→E), the modus ponens of Sheet 2, and (abs) becomes (→I):

$$\dfrac{\dfrac{y \to m,\ y \vdash_{\mathrm{NJ}} y \to m \qquad y \to m,\ y \vdash_{\mathrm{NJ}} y}{y \to m,\ y \vdash_{\mathrm{NJ}} m}\ {\to}\mathrm{E}}{y \to m \vdash_{\mathrm{NJ}} y \to m}\ {\to}\mathrm{I}$$

Rub out the contexts. The context is the same all the way down until the (→I) bar takes y out of it. So don't copy it onto every node: write a hypothesis as a leaf at the top, once for each place it is used, and put brackets and a number on a leaf that a bar takes away, with the same number on that bar:

$$\dfrac{\dfrac{y \to m \qquad [y]^1}{m}\ {\to}\mathrm{E}}{y \to m}\ {\to}\mathrm{I},\,1$$

This is the proof the way logicians draw it. The leaf $[y]^1$ is *discharged* by bar 1. The leaf $y \to m$ has no brackets, so it is still assumed at the end, as $rule1$ is still in the context at line 4.

And back again. To turn a tree into its program, *decorate* it from the top down. An open leaf gets its name from the rulebook ($rule1$ for $y \to m$). A bracketed leaf gets a new name of your choosing (x for $[y]^1$), the same name for every leaf with the same number. A (→E) bar applies the left program to the right one; a (→I) bar puts a λ on the name of the leaves it discharges (if it discharges none, use a name that appears nowhere else):

$$\dfrac{\dfrac{rule1 : y \to m \qquad [x : y]^1}{rule1\, x : m}\ {\to}\mathrm{E}}{\lambda x.\, rule1\, x : y \to m}\ {\to}\mathrm{I},\,1$$

Logicians call these trees *natural deduction*. Rubbed out, Charlie's three rules are three ways to grow a tree:

$$\text{(hyp)}\ \ A \qquad\qquad (\to\text{E})\ \ \frac{A \to B \qquad A}{B} \qquad\qquad (\to\text{I})\ \ \frac{\begin{array}{c}[A]^n \\ \vdots \\ B\end{array}}{A \to B}\ {\to}\text{I},\, n$$

A leaf may be any statement. A (→I) bar with number n that concludes $A \to B$ discharges *every* leaf $[A]^n$ in the part of the tree standing on that bar, and only leaves $[A]$; it may also discharge none. Each bar that discharges gets a number of its own: any numbers will do, as long as different bars get different numbers.

(a) The two instructions of Sheet 1 are no longer built in. They are scripts like any other:

$$\mathsf{K} := \lambda a.\, \lambda b.\, a, \qquad\qquad \mathsf{S} := \lambda f.\, \lambda g.\, \lambda x.\, f\, x\, (g\, x).$$

Run $\mathsf{K}\, M\, N$ and $\mathsf{S}\, M\, N\, P$, one β step per line, writing beside each step what was put in place of what. (Here M, N and P stand for scripts that do not mention a, b, f, g or x.) How many β steps does each take? (On Sheet 1, $\mathsf{K}\, M\, N \rightsquigarrow M$ and $\mathsf{S}\, M\, N\, P \rightsquigarrow M\, P\, (N\, P)$ took one step each.) [1]

Hint: $\mathsf{K}\, M\, N$ is $(\mathsf{K}\, M)\, N$, so the first step puts M in place of a.

(b) Bob still wants the network to take his key before his password. The rule $rule2 : w \to k \to m$ takes the password first, so Charlie writes an adapter,

$$swap := \lambda r.\, \lambda b.\, \lambda a.\, r\, a\, b,$$

which takes a rule r and hands it its two arguments in the other order. (A gap below is an empty box. It may hold a context, a program, a statement, the name of a rule, or a number on a bar.)

(i) Here is its typing as a tree, with five gaps. Γ stands for the context $r : w \to k \to m,\ b : k,\ a : w$. Fill in the gaps. [1]

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash_\lambda r : w \to k \to m \qquad \Gamma \vdash_\lambda a : w}{\Gamma \vdash_\lambda r\, a : \square}\,(\text{app}) \qquad \Gamma \vdash_\lambda b : k}{\Gamma \vdash_\lambda \square : m}\,(\text{app})}{r : w \to k \to m,\ b : k \vdash_\lambda \lambda a.\, r\, a\, b : \square}\,(\text{abs})}{\square \vdash_\lambda \lambda b.\, \lambda a.\, r\, a\, b : k \to w \to m}\,(\text{abs})}{\vdash_\lambda \square : (w \to k \to m) \to k \to w \to m}\,(\text{abs})$$

(ii) Rub out the programs and the contexts, as in the example. Here is the shape of what is left: every statement and every number is a gap. Fill them all in. [1]

$$\dfrac{\dfrac{\dfrac{\dfrac{[\square]^{\square} \qquad [\square]^{\square}}{\square}\,{\to}\text{E} \qquad [\square]^{\square}}{\square}\,{\to}\text{E}}{\square}\,{\to}\text{I},\square}{\square}\,{\to}\text{I},\square}{(w \to k \to m) \to k \to w \to m}\,{\to}\text{I},\square$$

Hint: the three leaves are the three (var) lines at the top of your typing tree. A leaf that an (abs) bar takes out of the context gets brackets, and that bar's number.

(iii) Run $swap\ rule2\ key\ pw$, one β step per line. Which script do you end with? [1]

(c) Now read your tree from (b)(ii).

(i) On Sheet 2 (A3), Bob proved this statement with the recipe: assume $w \to k \to m$, k and w; modus ponens twice gives m; discharge w, then k, then $w \to k \to m$. Which bars of your tree are the modus ponens steps, and which are the discharges? In which order, from the top down, are the leaves discharged? [1]

(ii) A certificate on Sheet 2 could use only (I), (II), hypotheses and MP, so the machine had to turn the recipe into a certificate, and that certificate came out at 161 lines. Charlie's tree has five bars. Which of the three rules is doing the machine's work? [1]

(d) **Mallory's tree.** Mallory sends Charlie a tree which, she says, shows that the key alone is enough: from $rule2 : w \to k \to m$, with no password, $k \to m$.

$$\dfrac{\dfrac{\dfrac{w \to k \to m \qquad [w]^1}{k \to m}\,{\to}\mathrm{E} \qquad [k]^1}{m}\,{\to}\mathrm{E}}{k \to m}\,{\to}\mathrm{I},1$$

One leaf of the tree is wrong. Which, and why? Put it right: what does the tree really show? [1]

A2. Pairs, choices and nothing.

The network adds three new kinds of statement: $\wedge$, $\vee$ and $\bot$. Like $\to$, each of them is a data type you have used.

statement	read	as a type (Haskell)	built by	taken apart by
$A \to B$	if A then B	a function `A -> B`	$\lambda x.\, M$	application
$A \wedge B$	A and B	a pair `(A, B)`	(M, N)	$fst\, M$, $snd\, M$
$A \vee B$	A or B	`Either A B`	$Left\, M$, $Right\, M$	a **case**
$\bot$	the impossible	`Void` (no values)	nothing	$absurd\, M$
$\neg A$	not A	short for $A \to \bot$	$\lambda x.\, M$	application

Two reading rules: $\neg$ binds most tightly, then $\wedge$ and $\vee$, then $\to$, so $w \wedge k \to m$ means $(w \wedge k) \to m$ and $\neg w \wedge k$ means $(\neg w) \wedge k$; and fst, snd, $Left$, $Right$ and $absurd$ are applied like functions, so $fst\, p\, (snd\, p)$ means $(fst\, p)\, (snd\, p)$.

A rule that builds a value is a *constructor*; logicians call it an *introduction* rule (I). A rule that takes a value apart is a *pattern match*; logicians call it an *elimination* rule (E). Here are the new rules as bars of a tree. Beside each is the program it builds, from the programs M, N, P, Q of the trees above the bar.

$$\dfrac{A \qquad B}{A \wedge B}\,{\wedge}\mathrm{I} \qquad (M, N) \qquad\qquad \dfrac{A}{A \vee B}\,{\vee}\mathrm{I}_1 \qquad Left\, M$$

$$\dfrac{A \wedge B}{A}\,{\wedge}\mathrm{E}_1 \qquad fst\, M \qquad\qquad \dfrac{B}{A \vee B}\,{\vee}\mathrm{I}_2 \qquad Right\, M$$

$$\dfrac{A \wedge B}{B}\,{\wedge}\mathrm{E}_2 \qquad snd\, M \qquad\qquad \dfrac{\bot}{C}\,{\bot}\mathrm{E}\ \ (\text{any } C) \qquad absurd\, M$$

and the one with three premises:

$$\dfrac{A \vee B \qquad \begin{matrix}[A]^i \\ \vdots \\ C\end{matrix} \qquad \begin{matrix}[B]^j \\ \vdots \\ C\end{matrix}}{C}\ \vee\mathrm{E}, i, j \qquad\qquad \textbf{case}\ M\ \textbf{of}\ \{\, Left\, a \Rightarrow P \mid Right\, b \Rightarrow Q \,\}$$

To use "A or B", you argue by cases: if you can get C supposing A, and get C supposing B, then you have C. The bar discharges $[A]^i$ in the middle tree and $[B]^j$ in the right-hand tree; they are the names a and b of the two branches. Nothing builds $\bot$, so you can never get it outright; but if you ever have it, *absurd* turns it into anything at all.

Example. The new terminals send the password and the key together, as a pair $(pw, key) : w \wedge k$. Here is the tree of a program that swaps a pair round:

$$\dfrac{\dfrac{\dfrac{[w \wedge k]^1}{k}\ \wedge\mathrm{E}_2 \qquad \dfrac{[w \wedge k]^1}{w}\ \wedge\mathrm{E}_1}{k \wedge w}\ \wedge\mathrm{I}}{w \wedge k \to k \wedge w}\ \to\mathrm{I}, 1$$

One bar discharges both leaves $[w \wedge k]^1$: they are the two uses of the parameter. Decorate the tree, calling both leaves p, and each bar builds the program beside it in the table:

$$\dfrac{\dfrac{\dfrac{[p : w \wedge k]^1}{snd\, p : k}\ \wedge\mathrm{E}_2 \qquad \dfrac{[p : w \wedge k]^1}{fst\, p : w}\ \wedge\mathrm{E}_1}{(snd\, p,\ fst\, p) : k \wedge w}\ \wedge\mathrm{I}}{\lambda p.\, (snd\, p,\ fst\, p) : w \wedge k \to k \wedge w}\ \to\mathrm{I}, 1$$

A programmer would write this function without a thought of logic. The type was a theorem all along.

(a) The rule $rule2 : w \to k \to m$ takes the password and the key one at a time, but the new terminals send a pair. Charlie needs an adapter $adapt$ of type $(w \to k \to m) \to w \wedge k \to m$. Here is a tree for that statement, with a run of five gaps down the middle. Fill them in. Then decorate the tree: call the leaf $[w \to k \to m]^2$ r, and both leaves $[w \wedge k]^1$ p. The program at the bottom is the adapter. [1]

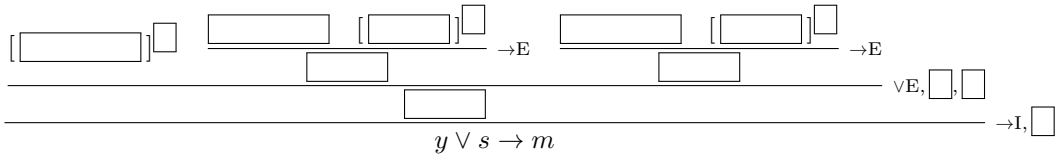
$$\dfrac{\dfrac{\dfrac{\dfrac{[w \to k \to m]^2 \qquad \dfrac{[w \wedge k]^1}{\boxed{}}\ \wedge\mathrm{E}_1}{\boxed{}}\ \to\mathrm{E} \qquad \dfrac{[w \wedge k]^1}{\boxed{}}\ \wedge\mathrm{E}_2}{\boxed{}}\ \to\mathrm{E}}{\boxed{}}\ \to\mathrm{I}, 1}{(w \to k \to m) \to w \wedge k \to m}\ \to\mathrm{I}, 2$$

Then run $adapt\ rule2\ (pw, key)$ to the end, one step per line. Besides β, you need $fst\,(M, N) \rightsquigarrow M$ and $snd\,(M, N) \rightsquigarrow N$. [1]

Hint: fill the gaps from the top down: each bar's rule tells you what comes out of what is above it.

(b) The rulebook has $rule1 : y \to m$ and $pay : s \to m$ ("signed payments may move"). Evidence now comes as one value of type $y \vee s$: "Alice said yes, or the payment is signed, and here is which". Charlie wants a program of type $y \vee s \to m$.

(i) Here is the shape of a tree for $y \vee s \to m$: only the bars, their rule names and the bottom statement are given. Fill in every box. [1]



(ii) Decorate your tree. The open leaves are $rule1 : y \to m$ and $pay : s \to m$; call the bracketed leaves e, a and b (for $y \vee s$, y and s). Which program do you get? Which hypotheses are still open at the end? [1]

Hint: argue by cases: the middle branch supposes y, the right-hand one supposes s, and each must reach m.

(c) Alice freezes her account. The rulebook gains $freeze : m \to \bot$, that is, $\neg m$: "the coins may not move". The rule $rule1 : y \to m$ is still there. Now draw your own trees. Work from the bottom up. Write the statement you want at the bottom. If it is an implication $A \to B$, draw a ($\to$I) bar above it with a new number n, and aim for B, using leaves $[A]^n$ as often as you like. Otherwise, look for a hypothesis that an elimination bar can take apart to give what you want.

(i) Draw a tree for $y \to \bot$, that is $\neg y$, whose open hypotheses are $y \to m$ and $m \to \bot$. Then decorate it, calling the bracketed leaf a: which program do you get? What does it say about Alice's yes? [1]

(ii) But Alice had said yes the day before: $ok : y$ is still in the rulebook too. Let t be the statement "the coins may go to Mallory". Draw a tree for t whose open hypotheses are y, $y \to m$ and $m \to \bot$, and decorate it to get its program. What does this show about a rulebook that contradicts itself? [1]

Hint: (i) needs three bars, the last of them ($\to$I). (ii) needs ($\bot$E).

A3. Running a certificate.

Here are all the steps of running, one for each kind of data:

$(\lambda x.\, M)\, N \;\rightsquigarrow\; M$ with N in place of x	(β)
$fst\,(M, N) \;\rightsquigarrow\; M$ $\qquad$ $snd\,(M, N) \;\rightsquigarrow\; N$	(fst, snd)
case $(Left\, M)$ **of** $\{\, Left\, a \Rightarrow P \mid Right\, b \Rightarrow Q \,\} \;\rightsquigarrow\; P$ with M in place of a	(**case**)
case $(Right\, M)$ **of** $\{\, Left\, a \Rightarrow P \mid Right\, b \Rightarrow Q \,\} \;\rightsquigarrow\; Q$ with M in place of b	(**case**)

A step may be taken anywhere inside a script, even inside a λ. In every one of them, something is built (a λ, a pair, a $Left$ or a $Right$) and at once taken apart. In a tree you can see it: an introduction bar whose conclusion is at once the *main premise* of an elimination bar for the same connective (the premise with that connective in it: the left-hand one in ($\to$E), the first one in ($\vee$E)). That is a *detour*: the proof went out of its way to build something, and then took it straight back. Running a script removes a detour from its tree, and never changes the statement at the bottom.

Example. From $pw : w$ and $key : k$, a padded tree for w, and the tree with the detour removed:

$$\dfrac{\dfrac{w \qquad k}{w \wedge k}\,\wedge\mathrm{I}}{w}\,\wedge\mathrm{E}_1 \qquad \text{becomes} \qquad w \qquad \text{just as} \qquad fst\,(pw, key) \;\rightsquigarrow\; pw.$$

The ($\wedge$I) bar built "w and k", and the ($\wedge\mathrm{E}_1$) bar at once took w back out; the proof of k was never needed, and is thrown away. If the tree you copy has numbered bars of its own, give each copy fresh numbers. For a ($\to$I) bar met at once by ($\to$E), the tree of the argument is put in place of every leaf that the ($\to$I) bar discharged: the argument takes the parameter's place. That is β.

(a) Here is a tree for m from $sign : y \to s$, $pay : s \to m$ and $ok : y$.

$$\dfrac{\dfrac{s \to m \qquad \dfrac{y \to s \qquad [y]^1}{s}\,\to\mathrm{E}}{m}\,\to\mathrm{E} \qquad y}{y \to m}\,\to\mathrm{I},1 \qquad \Big/\; m\;\Big\backslash\,\to\mathrm{E}$$

(i) Which two bars form a detour? Draw the tree without it. [1]

(ii) Decorate each tree, calling the leaf $[y]^1$ a, and check that the second program is one β step from the first. [1]

(b) Mallory claims a certificate with *no hypotheses at all*, no rulebook and no evidence, for the statement $(y \to y) \vee m$: "either Alice's yes implies Alice's yes, or the coins may move". Her script is $F\,(\lambda z.\, z,\ \mathit{Left}\,(\lambda x.\, x))$, where

$$F \coloneqq \lambda q.\, \textbf{case}\ snd\, q\ \textbf{of}\ \{\, \mathit{Left}\, f \Rightarrow \mathit{Left}\,(\lambda a.\, f\, a) \mid \mathit{Right}\, c \Rightarrow \mathit{Right}\, c \,\}.$$

It passes Charlie's check (take that on trust). Run it to the end, one step per line, naming each step: β, fst, snd or **case**. [1]

Hint: the first step is β: the pair on the right takes the place of q. The last step happens inside a λ.

(c) Three facts, to take on trust:

- Every script that passes Charlie's check stops. (Sheet 1 promised this for K and S; it holds for the new language too.)
- Running does not change the type, and never makes a name free.
- A stopped script that passes the check and has no free names starts with a constructor: $\lambda x.\, M$, (M, N), $\mathit{Left}\, M$ or $\mathit{Right}\, M$. (A stopped script that starts with a pattern match is a chain of pattern matches hanging off a name, and with no hypotheses there is no name for it to hang off.)

(i) Fill in the table: what must the type of each kind of stopped script look like? [1]

a stopped script with no free names	its type has the shape
$\lambda x.\, M$	
(M, N)	
$\mathit{Left}\, M$ or $\mathit{Right}\, M$	

(ii) Mallory now claims a certificate for m with no hypotheses: the coins may move, on no evidence at all. Without looking at it, Charlie throws it

away. Explain why he is right. Can there be a certificate for $\bot$ with no hypotheses? [1]

(d) **A proof of "or" says which.** Use the three facts to explain why every certificate with no hypotheses for a statement $A \vee B$ runs to $Left\, M$ or $Right\, M$, and say what M is then a certificate for. Mallory's last claim is a certificate with no hypotheses for $y \vee m$: "either Alice said yes, or the coins may move; I won't say which". Why may Charlie throw it away unread? [1]

A4. Mallory's paradox.

Write t for "the coins may go to Mallory". Mallory asks the network to add a statement that talks about itself, X: "if X holds, then t". So X says $X \to t$, and Mallory asks Charlie to treat the two as the same statement, $X = X \to t$: a script of type X is also a script of type $X \to t$, and the other way round.

> *Example.* Recall Ω from Sheet 1, the script that never stops. With λ it is $\Omega := (\lambda x.\, x\, x)\,(\lambda x.\, x\, x)$: the script "take x and apply it to itself", applied to itself. One β step puts $\lambda x.\, x\, x$ in place of each x in $x\, x$:
>
> $$(\lambda x.\, x\, x)\,(\lambda x.\, x\, x) \;\rightsquigarrow\; (\lambda x.\, x\, x)\,(\lambda x.\, x\, x).$$
>
> It is back where it started, and it never stops.

(a) Here is Mallory's certificate for t, as a tree with no open hypotheses. A bar marked with a star uses her equation: in a ($\to$E*) bar the left-hand premise X is read as $X \to t$, and a ($\to$I*) bar writes its conclusion $X \to t$ as X; they build the same programs as ($\to$E) and ($\to$I). Fill in the gaps: two statements, one rule name and one number. [1]

$$\dfrac{\dfrac{\dfrac{[X]^1 \qquad [X]^1}{t}\ \to\text{E}^*}{X \to t}\ \to\text{I},\,1 \qquad \dfrac{\dfrac{[X]^2 \qquad [X]^2}{\square}\ \to\text{E}^*}{X}\ \to\text{I}^*,\square}{\square}\ \square$$

Then read the tree as an argument in plain English, from the top down, starting with the left-hand half: "Suppose X. Since that is what X says, $X \to t$." [1]

(b) The root bar and the ($\to$I) bar above it on the left form a detour. Remove it, as in A3(a): the right-hand tree goes in place of both leaves $[X]^1$ (give the copies fresh numbers). What tree do you get? Which program is Mallory's tree, and what does removing the detour do to it? [1]

(c) Charlie refuses Mallory's equation.

(i) Which bars of the tree can no rule of the checker justify? Can any statement X built from letters and connectives be the same as $X \to t$? [1]

Hint: which is longer, X or $X \to t$?

(ii) On Sheet 1 (A4(c)), the checker refused $x\, x$ for a reason of the same kind. What do Sheet 1's never-stopping script and Mallory's certificate for t have in common? [1]

(d) Most programming languages have recursion. One way to add it is a built-in script fix, which runs by $fix\ F \rightsquigarrow F\,(fix\ F)$ and has the type $(A \to A) \to A$ for every A. Draw a two-bar tree for t whose only open hypothesis is the leaf

$(t \to t) \to t$, which is fix. Run its program for two steps, taking the step furthest to the left. What would fix do to Charlie's logic? [1]

Charlie's checker is real. Proof assistants such as Lean, Rocq (called Coq until 2025) and Agda are built on the idea of this sheet: a proof is a program, and checking a proof means type-checking a program. They also guard against A4(d). Lean's reference manual warns that arbitrary recursive definitions "would make Lean's logic inconsistent", so Lean asks for a reason to believe that each recursive definition is safe. A definition marked `partial` may still run, but Lean treats it as a sealed box and will not look inside it in a proof. And on the Cardano network, every computer that checks a payment runs its script in Plutus Core, a language built on the untyped λ-calculus.

Part B — Practice

Practice, for when you want more of the same. Drills B5–B12 are a ladder of tree drills: read trees, convert a list into a tree, draw and decorate your own, check them, and remove detours. The last two, B13 and B14, go beyond this week: try them only if you are curious. The rules you need:

- Scripts: names x; applications $M\,N$; $\lambda x.\,M$; pairs (M, N), $fst\,M$, $snd\,M$; $Left\,M$, $Right\,M$ and **case** M **of** $\{\, Left\,a \Rightarrow P \mid Right\,b \Rightarrow Q \,\}$; $absurd\,M$. Application groups to the left, and a λ reaches as far to the right as it can; fst, snd, $Left$, $Right$ and $absurd$ are applied like functions.
- Statements: letters, $\bot$, $A \to B$, $A \wedge B$, $A \vee B$. The arrow groups to the right, and $\wedge$, $\vee$ bind more tightly than $\to$; $\neg$ binds most tightly of all. $\neg A$ is short for $A \to \bot$.
- Running: $(\lambda x.\,M)\,N \rightsquigarrow M$ with N in place of x; $fst\,(M, N) \rightsquigarrow M$; $snd\,(M, N) \rightsquigarrow N$; a **case** on $Left\,M$ or $Right\,M$ takes its first or second branch with M in place of a or b. A step may be taken anywhere inside a script. When there is a choice, take the step furthest to the left.
- Trees: a leaf is a hypothesis; $[A]^n$ is a leaf discharged by the bar numbered n below it; a leaf with no brackets is *open*. The bars are ($\to$E) and ($\to$I) of A1 and the bars of A2. In a ($\to$E) bar the premise $A \to B$ goes on the left. A tree with open hypotheses Γ and A at the bottom shows $\Gamma \vdash_{\mathrm{NJ}} A$.
- Programs: to read off a tree's program, *decorate* it from the top down. An open leaf gets its given name; a bracketed leaf gets a new name, the same for every leaf with that number. Each bar builds the program written beside it in A1 and A2, and a ($\to$I) bar is a λ on the name of the leaves it discharges. The program at the bottom is the tree's program.
- $\mathsf{K} \coloneqq \lambda a.\,\lambda b.\,a$ and $\mathsf{S} \coloneqq \lambda f.\,\lambda g.\,\lambda x.\,f\,x\,(g\,x)$.

In the drills, A, B, C, p and the story letters y, w, k, m, s are statements, and other letters are names.

B1. Reading scripts. In $\lambda x.\,M$, the λ *binds* every x in M that no inner λx has bound already. An x that no λ binds is *free*. A script with no free names is *closed*. (i)–(vi): list the free names of each script, and say which are closed. (vii)–(ix): insert the brackets hidden by the rules that application groups to the left and a λ reaches as far to the right as it can, and say which of them can take a β step.

(i) $\lambda x.\,x\,z$
(ii) $(\lambda x.\,x\,z)\,x$
(iii) $\lambda x.\,\lambda u.\,x\,(u\,z)$
(iv) $\lambda f.\,f\,(\lambda x.\,f\,x)\,x$
(v) $\lambda x.\,\lambda u.\,u\,x$
(vi) $(\lambda x.\,x)\,(\lambda u.\,u)$
(vii) $\lambda x.\,f\,x\,z$
(viii) $(\lambda x.\,f\,x)\,z$
(ix) $\lambda u.\,\lambda x.\,x\,u\,z$

B2. Run each script to the end, one β step per line, and count the steps.

(i) $(\lambda x.\, x)\, a$
(ii) $(\lambda x.\, \lambda u.\, u\, x)\, a\, f$
(iii) $(\lambda f.\, \lambda x.\, f\, (f\, x))\, h\, c$
(iv) $(\lambda x.\, x\, x)\, (\lambda u.\, u)$

(v) Write **2** for $\lambda f.\, \lambda x.\, f\, (f\, x)$, the numeral $\overline{2}$ of Sheet 1. Run **2 2** h c, taking the step furthest to the left each time.

B3. Two warnings. (i) Run $\Omega = (\lambda x.\, x\, x)\, (\lambda x.\, x\, x)$ for three steps. (ii) Run $(\lambda x.\, \lambda u.\, x)\, u$ for one step, first blindly, putting u in place of x wherever it stands, and then after renaming the bound u to v. Apply each answer to c. Which answer is the one meant?

B4. Find a type for each script, using A, B, C for any statements, or explain why it has none. Read each type you find as a statement: what does it say?

(i) $\lambda x.\, x$
(ii) $\lambda f.\, \lambda x.\, f\, (f\, x)$
(iii) $\lambda a.\, \lambda b.\, b$
(iv) $\lambda x.\, x\, x$
(v) $\lambda q.\, fst\, q\, (snd\, q)$
(vi) $\lambda x.\, Left\, x$
(vii) $\lambda x.\, fst\, x\, x$

B5. Reading trees. For each tree, say which hypotheses are open, write what it shows as "$\Gamma \vdash_{\mathrm{NJ}} A$", and write its program. Use the names f, g and e for the open leaves given beside each tree.

(i) $f : A \to B$, $g : B \to C$:

$$\dfrac{B \to C \qquad \dfrac{A \to B \qquad [A]^1}{B}\,{\to}\mathrm{E}}{\dfrac{C}{A \to C}\,{\to}\mathrm{I},\,1}\,{\to}\mathrm{E}$$

(ii) no open leaves:

$$\dfrac{\dfrac{\dfrac{\dfrac{[A \to B]^3 \qquad [A]^1}{B}\,{\to}\mathrm{E} \qquad \dfrac{[A \to C]^2 \qquad [A]^1}{C}\,{\to}\mathrm{E}}{B \wedge C}\,{\wedge}\mathrm{I}}{A \to B \wedge C}\,{\to}\mathrm{I},\,1}{(A \to C) \to A \to B \wedge C}\,{\to}\mathrm{I},\,2}{(A \to B) \to (A \to C) \to A \to B \wedge C}\,{\to}\mathrm{I},\,3$$

(iii) $e : A \vee B$, $f : A \to C$, $g : B \to C$:

$$\dfrac{A \vee B \qquad \dfrac{A \to C \qquad [A]^1}{C}\,{\to}\mathrm{E} \qquad \dfrac{B \to C \qquad [B]^2}{C}\,{\to}\mathrm{E}}{C}\,{\vee}\mathrm{E},\,1,\,2$$

(iv) $e : (A \wedge B) \vee (A \wedge C)$:

$$\dfrac{(A \wedge B) \vee (A \wedge C) \qquad \dfrac{[A \wedge B]^1}{A}\,{\wedge}\mathrm{E}_1 \qquad \dfrac{[A \wedge C]^2}{A}\,{\wedge}\mathrm{E}_1}{A}\,{\vee}\mathrm{E},\,1,\,2$$

B6. Draw trees with no open hypotheses for Sheet 2's two axioms,

(I) $A \to B \to A$ (II) $(A \to B \to C) \to (A \to B) \to A \to C$,

and decorate them to get their programs. Check that they are K and S. Which bar of the first tree discharges nothing? Which leaf of the second tree appears twice, and what does that say about the program?

B7. From a list to a tree. Here is a natural-deduction derivation as a list, like the typing in A1's example with the programs rubbed out, so (var) is called (hyp), where every line carries its context; Δ is $A \wedge (B \vee C)$. A rule cites lines with the same context as its own, except that ($\to$I) cites a line, and ($\vee$E) its last two lines, whose context has one more hypothesis, the one they discharge.

1. $\Delta \vdash_{\mathrm{NJ}} A \wedge (B \vee C)$ (hyp)
2. $\Delta \vdash_{\mathrm{NJ}} B \vee C$ ($\wedge\mathrm{E}_2$) 1
3. $\Delta,\ B \vdash_{\mathrm{NJ}} A \wedge (B \vee C)$ (hyp)
4. $\Delta,\ B \vdash_{\mathrm{NJ}} A$ ($\wedge\mathrm{E}_1$) 3
5. $\Delta,\ B \vdash_{\mathrm{NJ}} B$ (hyp)
6. $\Delta,\ B \vdash_{\mathrm{NJ}} A \wedge B$ ($\wedge$I) 4, 5
7. $\Delta,\ B \vdash_{\mathrm{NJ}} (A \wedge B) \vee (A \wedge C)$ ($\vee\mathrm{I}_1$) 6
8. $\Delta,\ C \vdash_{\mathrm{NJ}} A \wedge (B \vee C)$ (hyp)
9. $\Delta,\ C \vdash_{\mathrm{NJ}} A$ ($\wedge\mathrm{E}_1$) 8
10. $\Delta,\ C \vdash_{\mathrm{NJ}} C$ (hyp)
11. $\Delta,\ C \vdash_{\mathrm{NJ}} A \wedge C$ ($\wedge$I) 9, 10
12. $\Delta,\ C \vdash_{\mathrm{NJ}} (A \wedge B) \vee (A \wedge C)$ ($\vee\mathrm{I}_2$) 11
13. $\Delta \vdash_{\mathrm{NJ}} (A \wedge B) \vee (A \wedge C)$ ($\vee$E) 2, 7, 12
14. $\vdash_{\mathrm{NJ}} A \wedge (B \vee C) \to (A \wedge B) \vee (A \wedge C)$ ($\to$I) 13

Draw it as a tree, and write its program. In the context of how many lines of the list does Δ appear? How many leaves $A \wedge (B \vee C)$ does the tree have?

B8. Draw a tree with no open hypotheses for each statement. Then decorate it to get its program. What does each program do?

(i) $(A \to B \to C) \to B \to A \to C$
(ii) $(A \wedge B \to C) \to A \to B \to C$
(iii) $(A \to B \wedge C) \to (A \to B) \wedge (A \to C)$

Hint: work from the bottom up. To prove an implication, suppose its left-hand side: write it as a leaf with a number, and aim for the right-hand side. To prove an "and", prove both halves.

B9. Checking trees. Each tree below has exactly one mistake. Find the bar or leaf that is wrong, and say why.

(i) A tree that claims $\vdash_{\mathrm{NJ}} A \to B \to A$:

$$\dfrac{\dfrac{[A]^1}{B \to A}\ {\to}\mathrm{I},\,1}{A \to B \to A}\ {\to}\mathrm{I}$$

(ii) A tree that claims $\vdash_{\mathrm{NJ}} A \vee B \to A$:

$$\dfrac{\dfrac{[A \vee B]^1 \qquad [A]^2 \qquad [A]^3}{A}\ {\vee}\mathrm{E},\,2,\,3}{A \vee B \to A}\ {\to}\mathrm{I},\,1$$

(iii) A tree that claims $\vdash_{\mathrm{NJ}} A \wedge (A \to A)$:

$$\dfrac{[A]^1 \qquad \dfrac{[A]^1}{A \to A}\ {\to}\mathrm{I},\,1}{A \wedge (A \to A)}\ {\wedge}\mathrm{I}$$

(iv) A tree that claims $A \to B,\ B \vdash_{\mathrm{NJ}} A$:

$$\dfrac{A \to B \qquad B}{A}\ {\to}\mathrm{E}$$

B10. "Or" with "and". Draw a tree with no open hypotheses for each statement. Then decorate it to get its program.

(i) $A \vee B \to B \vee A$
(ii) $(A \to C) \wedge (B \to C) \to A \vee B \to C$
(iii) $A \vee B \to \neg(\neg A \wedge \neg B)$
(iv) $(A \wedge B) \vee (A \wedge C) \to A \wedge (B \vee C)$
(v) $A \vee (B \wedge C) \to (A \vee B) \wedge (A \vee C)$

Hint: every one of them uses ($\vee$E): suppose the "or", and argue by cases. In (ii), each branch takes the pair of rules apart. In (iii), $\neg(\neg A \wedge \neg B)$ is $(\neg A \wedge \neg B) \to \bot$: suppose $\neg A \wedge \neg B$, and reach $\bot$ as in (ii). In (iv) and (v), each branch must build a whole pair: prove both halves.

B11. Negation. Draw a tree with no open hypotheses for each statement. Then decorate it to get its program.

(i) $A \to \neg\neg A$ (ii) $(A \to B) \to \neg B \to \neg A$ (iii) $\neg\neg(A \vee \neg A)$

Hint: $\neg A$ is $A \to \bot$, so a statement $\neg A$ is used with ($\to$E) and proved with ($\to$I). For (iii), suppose $\neg(A \vee \neg A)$ and aim for $\bot$. Feed that leaf a Right of something, and build that something using the same leaf again.

B12. Removing detours. Each tree has one detour. Draw the tree without it, decorate both trees (the open leaves' names are beside each tree; name the bracketed leaves as you like), and check that the second is one step from the first.

(i) $g : C \to A,\ c : C,\ b : B$:

$$\dfrac{\dfrac{\dfrac{C \to A \qquad C}{A}\ {\to}\mathrm{E} \qquad B}{A \wedge B}\ {\wedge}\mathrm{I}}{B}\ {\wedge}\mathrm{E}_2$$

(ii) $f : A \to A \to B,\ g : C \to A,\ c : C$:

$$\dfrac{\dfrac{\dfrac{A \to A \to B \qquad [A]^1}{A \to B}\ {\to}\mathrm{E} \qquad [A]^1}{B}\ {\to}\mathrm{E}}{A \to B}\ {\to}\mathrm{I},1 \qquad \dfrac{C \to A \qquad C}{A}\ {\to}\mathrm{E}}{B}\ {\to}\mathrm{E}$$

(iii) $a : A,\ f : A \to C,\ g : B \to C$:

$$\dfrac{\dfrac{A}{A \vee B}\ {\vee}\mathrm{I}_1 \qquad \dfrac{A \to C \qquad [A]^1}{C}\ {\to}\mathrm{E} \qquad \dfrac{B \to C \qquad [B]^2}{C}\ {\to}\mathrm{E}}{C}\ {\vee}\mathrm{E},1,2$$

Hint: in (ii), the ($\to$I) bar discharges two leaves, so the argument's tree is put in two places.

B13. *(For the curious.)* Try to draw a tree with no open hypotheses for $\neg\neg p \to p$, where p is a letter, that is, $((p \to \bot) \to \bot) \to p$. Say where every attempt gets stuck.

B14. *(For the curious.)* Use the three facts of A3(c), and what A3(d) found, to show that there is no closed program of type $p \vee \neg p$, though B11(iii) found one of type $\neg\neg(p \vee \neg p)$.

Hint: a stopped script that passes the check and does not start with a constructor is a chain of pattern matches hanging off a name. Inside $\lambda x.\, N$, the only name around may be x.