COMP0009 Logic — Problem Sheet 1

Part A — Charlie's week

Alice wants to send Bob ten coins, but nobody on the network trusts anybody. So the coins are locked behind a small program, called a *script*. To collect them, Bob supplies some *evidence*, and every computer on the network runs the script on it. If the run ends in *true*, the coins move to Bob. If it ends in *false*, they stay with Alice.

Charlie runs one of these computers, and you are going to help him. What must never happen is a script that never stops, because then every computer on the network is stuck. In the four scenes below you will run scripts, build new ones, meet a script that never stops, and then help Charlie to spot such a script *before* he runs it.

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

A1. Charlie checks a payment.

Alice has locked her ten coins with a script called *lock*. Bob can collect in two ways: if Alice has approved the payment, or, if she has not, by proving who he is with both his password and his key.

The evidence is three yes/no answers, each $true$ or $false$:

a: has Alice approved? w: is the password right? k: is the key right?

Charlie's computer runs $lock\,a\,w\,k$.

Alice also has a *freeze switch*: a boolean z that is $true$ while her account is frozen. While z is $true$, nothing may be released, whatever the evidence.

These are the definitions that Charlie's computer uses. The first two say what $true$ and $false$ do:

$$true\,x\,y := x, \qquad false\,x\,y := y,$$
$$not\,b := b\,false\,true, \qquad and\,p\,q := p\,q\,false, \qquad or\,p\,q := p\;true\;q,$$
$$lock\,a\,w\,k := or\;a\;(and\,w\,k).$$

> *Example.* A boolean is a choice: $true$ picks the first thing after it, $false$ the second. That is how the definitions above were found. To define something that depends on a boolean p, ask two questions: what should the answer be when p is $true$, and what when p is $false$? Then write p, followed by those two answers.
>
> - $and\,p\,q$: if p is $true$ the answer is q; if p is $false$ the answer is $false$. So $and\,p\,q := p\,q\,false$.
> - $not\,b$: if b is $true$ the answer is $false$; if b is $false$ the answer is $true$. So $not\,b := b\,false\,true$.
>
> Here is a complete run of $and\;true\;true$, one step per line:
>
> $$\begin{array}{lll} and\;true\;true & \rightsquigarrow\; true\;true\;false & \text{definition of } and \\ & \rightsquigarrow\; true & true\,x\,y := x \end{array}$$
>
> A run ends when no step is left. What remains is the answer.

(a) Define $guard\,z\,a\,w\,k$, which runs to $true$ exactly when the account is not frozen (z is $false$) and $lock\,a\,w\,k$ runs to $true$. [1]

Hint: it is an and of two things, and one of them is "not frozen".

(b) Alice has not approved; Bob's password is right, but his key is wrong. Run $lock\ false\ true\ false$, one step per line, and write the reason for each step beside it. Does the payment move? [2]

(c) Here is every possible piece of evidence, one per row. Fill in the last column: what does $lock\ a\ w\ k$ run to? [2]

a	w	k	$lock\ a\ w\ k$
$true$	$true$	$true$	
$true$	$true$	$false$	
$true$	$false$	$true$	
$true$	$false$	$false$	
$false$	$true$	$true$	
$false$	$true$	$false$	
$false$	$false$	$true$	
$false$	$false$	$false$	

Then finish the sentence: "the payment moves exactly when …".

Hint: do not work out each row by running it: that is eight long runs, and it is easy to slip. There is a much quicker way. Look again at the definition of lock.

A2. Charlie extends the library.

Alice, Bob and Charlie decide to share the coins as partners. From now on, a payment needs at least two of the three to approve. Charlie must put this rule into the library, and you will write it. (This language has no `if` and no loops.)

Under the bonnet, the language has just two built-in instructions, K and S. Each takes one step:

$$\mathsf{K}\ x\ y \rightsquigarrow x, \qquad \mathsf{S}\ x\ y\ z \rightsquigarrow x\ z\ (y\ z).$$

K keeps the first thing and drops the second. S gives z to both x and y, and then applies the first result to the second. Every other name is a definition, built on top of these two.

Example. A definition has a name, some placeholders, and a body:

$$twice\ f\ x := f\ (f\ x)$$

It says: given any f and any x, apply f to x, and then apply f to that. When Charlie sees $twice$ followed by two things, he puts them in the places of f and x. So $twice\ g\ y \rightsquigarrow g\ (g\ y)$.

Three things to keep in mind.

- A definition is *one pattern*, not a list of cases. The left-hand side is a name followed by placeholders, and the body has to work for every boolean that comes in their place.
- To do that, use the recipe from A1: ask what the answer is when the first placeholder is $true$, and what it is when it is $false$. Each answer may itself use and, or and not.
- The body may use only K, S, the placeholders, and names defined *earlier*. A definition cannot call itself, and that is why this language has no loops.

(a) Charlie opens the library file, and finds that $true$ and $false$ are not defined there at all. In their place are two short scripts: K and S K. Run K $a\ b$ and

$\mathsf{S}\,\mathsf{K}\;a\,b$, one step per line, and say which of $true$ and $false$ each one behaves like. [1]

What you have found is that the library needs nothing but K and S: every definition on this sheet, *lock* and the one you are about to write included, can be written out with those two letters alone.

(b) Define $maj\;a\,b\,c$, which runs to $true$ exactly when at least two of a, b, c are $true$. (Here a, b, c say whether Alice, Bob and Charlie approve.) [2]

Hint: ask the two questions about a. If a is $true$, what must be true of b and c for there to be two approvals? And if a is $false$?

(c) Test your definition: run $maj\;true\;false\;true$ and $maj\;false\;true\;false$, one step per line. Are the answers what you expected? [2]

(d) **Mallory's first attack.** Mallory wants to jam the network. Her first idea is a script that does stop, but only after building an enormous answer. She builds it from numbers. Remember that a number means "do it n times":

$$\overline{3}\;f\;x \rightsquigarrow f\,(f\,(f\,x)) \qquad\qquad \overline{2}\;f\;x \rightsquigarrow f\,(f\,x)$$

(i) Run $\overline{2}\;\overline{2}\;f\;x$, one step per line, and count the copies of f at the end. So $\overline{2}\;\overline{2}$ does the same job as which number?

(ii) Each $\overline{2}$ doubles. So $\overline{n}\;\overline{2}\;f\;x$ ends with n doublings: $2 \times 2 \times \cdots \times 2$ (n twos) copies of f. Use this and (i) to say, without running anything, how many copies of f there are at the end of $\overline{2}\;\overline{2}\;\overline{2}\;f\;x$, and at the end of Mallory's script, $\overline{2}\;\overline{2}\;\overline{2}\;\overline{2}\;f\;x$. [2]

Hint: application groups to the left, so $\overline{2}\;\overline{2}\;\overline{2}$ is $(\overline{2}\;\overline{2})\;\overline{2}$. By (i), that does the same job as some number followed by $\overline{2}$.

A3. Mallory's trap.

Mallory is back, with a trap in her script. Charlie and Bob both run it. Both follow the rules, and still they disagree. The reason is that a script sometimes offers more than one step, and the two choose differently.

- **Charlie is lazy.** He takes the step nearest the front, and never tidies up an argument before the function needs it.
- **Bob is eager.** He first finishes everything inside the script, from the left, and only then takes the step that uses it. Most programming languages are eager.

Example. Recall from A2 that $\mathsf{K}\,M\,N \rightsquigarrow M$ (keep the first thing, drop the second). Here is how Charlie and Bob run $\mathsf{K}\;true\;(and\;true\;true)$:

Charlie: $\mathsf{K}\;true\;(and\;true\;true) \rightsquigarrow true$

Bob: $\mathsf{K}\;true\;(and\;true\;true) \rightsquigarrow \mathsf{K}\;true\;(true\;true\;false)$
$\rightsquigarrow \mathsf{K}\;true\;true \rightsquigarrow true$

Charlie takes one step: K drops its second argument unread. Bob tidies up that argument first, and takes three. The answer is the same; the work is not.

Remember $and\;p\;q := p\;q\;false$, $true\;x\;y := x$ and $false\;x\;y := y$. Mallory's trap is the script Ω of Chapter 1, which never stops: $\Omega := \omega\,\omega$, where $\omega\;x := x\;x$.

(a) Charlie and Bob both run $and\ false\ (and\ true\ true)$. Each of them starts like this:

Charlie: $and\ false\ (and\ true\ true) \rightsquigarrow false\ (and\ true\ true)\ false$

Bob: $and\ false\ (and\ true\ true) \rightsquigarrow and\ false\ (true\ true\ false)$

Carry on until each run ends, one step per line. How many steps did each take, and do they end with the same answer? [2]

(b) Now Mallory's trap itself. Complete the table: what does each computer end with? Write "never" where it never finishes, and add a short reason. (Be careful with the third row.) [2]

script	Charlie	Bob
$\mathsf{K}\ true\ \Omega$		
$and\ false\ \Omega$		
$and\ true\ \Omega$		

(c) Mallory's payment $\mathsf{K}\ true\ \Omega$ is released by Charlie, while Bob hangs. What could the network's rulebook say, so that this cannot happen? Give one rule, and say what it costs. [1]

A4. The pre-flight check.

Charlie is tired of finding out the hard way. From now on, every script must arrive with a *typing*: a short list of lines saying what type each part of the script has. Charlie checks the typing line by line, and runs the script only if it checks. (Chapter 1 proves that a script that passes the check always stops.)

A *type* is a basic type (Int, $String$) or $A \to B$, a function that takes an A and gives a B. The arrow groups to the right: $A \to B \to C$ means $A \to (B \to C)$. Each line of a typing says that a script has a type ($\vdash_{\mathrm{KS}}$ just means that no variables are around), and gives one of five reasons:

(K)	$\mathsf{K} : A \to B \to A$, for any types A and B
(S)	$\mathsf{S} : (A \to B \to C) \to (A \to B) \to A \to C$, for any types A, B, C
(var)	$x : A$, when we are told that x has type A
(given)	a ready-made fact: $1 : Int$, $\texttt{"hi"} : String$, $+ : Int \to Int \to Int$, $\mathsf{I} : A \to A$
(app)	$M_1\ M_2 : B$, from two *earlier* lines $M_1 : A \to B$ and $M_2 : A$

Example. Take the script $\mathsf{K}\ 1\ \texttt{"hi"}$: the constant K, given the number 1 and the word `"hi"`. K keeps the first thing, so the answer is the number 1, which has type Int. Here is the typing, line by line:

1. $\vdash_{\mathrm{KS}} \mathsf{K} : Int \to String \to Int$	(K), with $A = Int$, $B = String$
2. $\vdash_{\mathrm{KS}} 1 : Int$	(given)
3. $\vdash_{\mathrm{KS}} \mathsf{K}\ 1 : String \to Int$	(app) 1, 2
4. $\vdash_{\mathrm{KS}} \texttt{"hi"} : String$	(given)
5. $\vdash_{\mathrm{KS}} \mathsf{K}\ 1\ \texttt{"hi"} : Int$	(app) 3, 4

Line 3: line 1 is a function that takes an Int and gives a $String \to Int$; line 2 is an Int; so $\mathsf{K}\ 1$ has type $String \to Int$.

(a) Bob sends the typing below for the script $\mathsf{K}\ \texttt{"yes"}\ 3$ (K, given the word `"yes"` and the number 3), whose answer should have type $String$. Charlie rejects it. Find the first line that is wrong, say what is wrong with it, and repair the typing by changing *one* line. [2]

1. $\vdash_{\mathrm{KS}} \mathsf{K} : \mathit{String} \to \mathit{Int} \to \mathit{String}$ (K), with $A = \mathit{String}$, $B = \mathit{Int}$
2. $\vdash_{\mathrm{KS}}$ `"yes"` $: \mathit{String}$ (given)
3. $\vdash_{\mathrm{KS}} \mathsf{K}$ `"yes"` $: \mathit{String} \to \mathit{Int}$ (app) 1, 2
4. $\vdash_{\mathrm{KS}} 3 : \mathit{Int}$ (given)
5. $\vdash_{\mathrm{KS}} \mathsf{K}$ `"yes"` $3 : \mathit{String}$ (app) 3, 4

(b) Alice sends the script $+$ (K 1 `"hi"`) 2, which adds 2 to the answer of K 1 `"hi"`. Below is a typing for it, of type Int, with gaps. Lines 1–5 are the five lines of the example. Fill in the gaps. [2]

6. $\vdash_{\mathrm{KS}} + :$ _____ (given)
7. $\vdash_{\mathrm{KS}} +$ (K 1 `"hi"`) $:$ _____ (app) 6, ___
8. $\vdash_{\mathrm{KS}} 2 : \mathit{Int}$ (given)
9. $\vdash_{\mathrm{KS}} +$ (K 1 `"hi"`) $2 :$ _____ (app) ___, ___

(c) Now Mallory sends $x\ x$: the variable x, applied to itself. Is there a type A that x could have, so that $x\,x$ passes the check? Remember that (app) needs a function on the left, and an argument of exactly the type that the function wants. [2]

(i) Try $A = \mathit{Int}$. Why can the check not pass?

(ii) Try $A = \mathit{Int} \to \mathit{Int}$. Now x is a function. What type of argument does it want, and what type does the argument (which is x again) actually have?

(iii) Whatever A you try, the same thing goes wrong. Explain why.

Hint: in (iii), if A is $C \to D$, then x wants an argument of type C. Which is longer, C or A?

(d) The identity I ($\mathsf{I}\ x \rightsquigarrow x$) is built as $\mathsf{S}\,\mathsf{K}\,\mathsf{K}$. Written with only K and S, ω is $\mathsf{S}\,\mathsf{I}\,\mathsf{I}$, and Ω is $\mathsf{S}\,\mathsf{I}\,\mathsf{I}\,(\mathsf{S}\,\mathsf{I}\,\mathsf{I})$. Take on trust that $\mathsf{S}\,\mathsf{I}\,\mathsf{I}$ has no typing: it is the same trouble as in (c). Charlie runs Mallory's script $\mathsf{K}\ \mathit{true}\ \Omega$ to true in one step. Why does his checker reject it all the same? [1]

(e) Mallory tries once more, with her script from A2(d) and one more $\overline{2}$: $\overline{2}\ \overline{2}\ \overline{2}\ \overline{2}\ \overline{2}\ f\ x$. This script *does* pass the check (take that on trust), so it is sure to stop. How many copies of f will its answer have? Should Charlie run it? [1]

Hint: by A2(d), the first four $\overline{2}$s do the same job as $\overline{65536}$.

Charlie's problem is real. Bitcoin's script language has no loops at all, and it limits how long a script may be, so that every script stops, and stops quickly. Ethereum's language does allow loops, but it charges a small fee, called *gas*, for every step: a script that runs too long runs out of money and fails. A4(e) shows why a check that only promises "it stops" would not be enough.

Part B — Practice

Practice, for when you want more of the same. The last two, B13 and B14, go beyond this week and lean on the notes: try them only if you are curious. The rules you need:

- Application groups to the left: $f\,x\,y$ means $(f\,x)\,y$.
- $\mathsf{K}\,M\,N \rightsquigarrow M$ and $\mathsf{S}\,M\,N\,P \rightsquigarrow M\,P\,(N\,P)$. A definition $f\,x_1 \cdots x_n := \text{body}$ is used by putting each argument in the place of its x_i.
- A script with no step left is in *normal form*: its answer.
- The library is the one of Part A: $\mathsf{I}\ x := x$, $true\ x\,y := x$, $false\ x\,y := y$ (each one step), the numerals $\overline{n}\ f\,x := f\,(f\,(\cdots(f\,x)))$ with n copies of f, and not, and, or. In the drills a, b, c, d, f, g, x are variables.

B1. Insert the brackets that left-grouping hides: (i) $\mathsf{K}\,a\,b\,c$; (ii) $\mathsf{S}\,x\,y\,z\,w$; (iii) $f\,(g\,x)\,y\,z$. Then say which of these strings are scripts: $\mathsf{K}\,(\mathsf{S}\,\mathsf{K})$; $(\mathsf{K}\,\mathsf{S}\,\mathsf{K};\ \mathsf{S}\,\mathsf{K})\,\mathsf{K}$; $x\,(y\,z)$; $\mathsf{K}\,\mathsf{K}\,\mathsf{K}\,\mathsf{K}$.

B2. A definition is $f\,x_1 \cdots x_n := M$ with distinct parameters, where M uses only K, S, the parameters and names defined *earlier*. Which of these are definitions? Give a reason for each that is not.

(i) $f\,x\,y := y\,x$
(ii) $g\,x\,y := x\,z$
(iii) $h\,x\,x := x$
(iv) $k\,x := k\,x$
(v) $m\,x := \mathsf{K}\,(\mathsf{S}\,x)$
(vi) $n := \mathsf{S}\,\mathsf{K}\,\mathsf{K}$
(vii) $r\,x\,y := \mathsf{K}\,x$

B3. Run each script to a normal form, one step per line. Count the steps.

(i) $\mathsf{K}\,a\,b\,c$
(ii) $\mathsf{K}\,(\mathsf{K}\,a)\,b\,c$
(iii) $\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,a$
(iv) $\mathsf{S}\,(\mathsf{K}\,a)\,(\mathsf{K}\,b)\,c$
(v) $\mathsf{S}\,\mathsf{K}\,\mathsf{S}\,\mathsf{K}$
(vi) $\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,(\mathsf{S}\,\mathsf{K}\,\mathsf{K})$

B4. Two more programs written from the two constants. Run (i) $\mathsf{S}\,(\mathsf{K}\,\mathsf{S})\,\mathsf{K}\,f\,g\,x$ and (ii) $\mathsf{S}\,\mathsf{S}\,(\mathsf{S}\,\mathsf{K})\,f\,x$ to normal form. What does each do? Count the steps.

B5. Numerals. With the definitions

$$succ\ n\,f\,x := f\,(n\,f\,x), \qquad add\ m\,n\,f\,x := m\,f\,(n\,f\,x), \qquad mult\ m\,n\,f := m\,(n\,f),$$

(i) Run $succ\ \overline{2}\ f\ x$. Which numeral does it equal?
(ii) Run $add\ \overline{3}\ \overline{2}\ f\ x$ and $mult\ \overline{2}\ \overline{3}\ f\ x$, and count the copies of f.
(iii) Let $iszero\ n := n\,(\mathsf{K}\,false)\,true$. Run $iszero\ \overline{0}$ and $iszero\ \overline{2}$.
(iv) How many copies of f are there in the normal forms of $\overline{3}\ \overline{2}\ f\ x$ and of $\overline{2}\ \overline{3}\ f\ x$? Work it out before running anything. Which is bigger, and why does the order matter?

B6. Booleans. Define $xor\ p\,q := p\,(not\,q)\,q$ and $nand\ p\,q := not\,(and\,p\,q)$ (with $not\ b := b\,false\,true$ and $and\ p\,q := p\,q\,false$). Fill in a table for each over the four pairs (p, q) by running them, and say in a word what each computes. Then run $iszero\ \overline{0}\ a\ b$ and $iszero\ \overline{2}\ a\ b$: what has $iszero$ turned a number into?

B7. The order of steps. Run $\mathsf{S}\,\mathsf{S}\,\mathsf{K}\,(\mathsf{K}\,a\,b)\,c$ twice: once always rewriting the leftmost-outermost redex, once always the leftmost-innermost redex. Compare the normal forms and the number of steps.

B8. The trap in two constants. Here $\omega\, x \coloneqq x\, x$. (i) Run $\mathsf{S}\,\mathsf{I}\,\mathsf{I}\,x$ (count $\mathsf{I}\,M \rightsquigarrow M$ as one step). What does it do to x? So which script is ω, when written with S and I only? (ii) Write X for $\mathsf{S}\,\mathsf{I}\,\mathsf{I}$, so that Ω is $X\,X$. Show that $X\,X$ comes back to itself after three steps. (Rewrite the *argument* $\mathsf{I}\,X$ before the front: if you do it the other way round, the script grows instead. Either way it never stops.)

B9. For each script, say whether every run stops, some runs stop and some do not, or no run stops. Here $\Omega = \omega\,\omega$ with $\omega\, x \coloneqq x\, x$.

(i) Ω
(ii) $\mathsf{K}\,a\,\Omega$
(iii) $\mathsf{K}\,\Omega\,a$
(iv) $\mathsf{S}\,\mathsf{I}\,\mathsf{I}\,\mathsf{K}$
(v) $\mathsf{S}\,\mathsf{K}\,\Omega\,a$

B10. Types. (i) Insert the brackets: $A \to B \to C \to D$; $(A \to B) \to C \to D$; $(A \to B \to C) \to D$. (ii) The schema (K) is $A \to B \to A$. Which of these are instances of it, for some choice of types for A and B?

(a) $Int \to String \to Int$
(b) $Int \to Int \to Int$
(c) $Int \to String \to String$
(d) $(p \to q) \to r \to (p \to q)$
(e) $p \to p$
(f) $p \to (q \to r) \to p$

(iii) The schema (S) is $(A \to B \to C) \to (A \to B) \to A \to C$. Which of these are instances of it?

(a) $(p \to q \to r) \to (p \to q) \to p \to r$
(b) $(p \to p \to p) \to (p \to p) \to p \to p$
(c) $(p \to q \to r) \to (p \to r) \to p \to r$
(d) $(Int \to Int \to Int) \to (Int \to Int) \to Int \to Int$

B11. Write a typing, in the form of A4, with the empty context, for (i) $\mathsf{K}\,\mathsf{K}$ and (ii) $\mathsf{S}\,\mathsf{K}\,\mathsf{S}$. Work backwards from the goal: what type does each constant have to take, for the (app) lines to match? State the type each script ends up with.

B12. Ready-made pieces: integers have type Int, $+ \colon Int \to Int \to Int$, and strings have type $String$. For each script, say whether it has a typing. If so, give its type; if not, say which application fails and what it expected and what it got.

(i) $+\ 1\ 2$
(ii) $+\ 1\ (+\ 2)$
(iii) $\mathsf{K}\ 1\ \texttt{"a"}$
(iv) $+\ (\mathsf{K}\ 1\ \texttt{"a"})\ 3$
(v) $+\ \texttt{"a"}\ 1$
(vi) $\mathsf{K} + 1$
(vii) $+\ \mathsf{K}\ 1$

B13. Subject reduction. (i) In the context $f \colon B \to C$, $g \colon A \to B$, $x \colon A$, the script $\mathsf{S}\,(\mathsf{K}\,f)\,g\,x$ has type C. Run it, and write the type of every script in the run. Give also the type of $\mathsf{K}\,f\,x$ and of $g\,x$ in the second script. (ii) Suppose a careless checker gave K the type $A \to B \to B$ instead of $A \to B \to A$. Find a script of the language of B12 whose type *changes* when it takes one step. Which part of the proof of subject reduction would break?

B14. Recursion. Add a constant *fix* to the language, with the step $\mathit{fix}\,f \rightsquigarrow f\,(\mathit{fix}\,f)$ and the type $(A \to A) \to A$ for every A. Run each script for at most five steps and give its type. (i) $\mathit{fix}\,(\mathsf{K}\,1)$. (ii) $\mathit{fix}\,\mathsf{I}$. (iii) $\mathit{fix}\,(\mathsf{K}\,\mathsf{K})$. What does `fix` *allow*, and what does it *force*?