Harry Langford
hjel2@cam.ac.uk

1 2004 Paper 3 Question 7

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2004p3q7.pdf

(a) What is Turing's Thesis?

The Turing's Thesis states that any computable algorithm can be realised as a Turing Machine.

This means that "there is no model of computation which is *more* powerful than a Turing Machine".

Don't formalise it like that -- we're explicitly talking about deterministic and finite algorithms -- we could define another model that's more powerful (infinite lookup)

So include "finite" and "deterministic" in the definition.

(b) Explain the action of a Turing Machine that is specified by a quintuplet description.

A Turing Machine is specified by a quadruple $M = (Q, \Sigma, s, \delta)$.

- Q is the set of states
- Σ is the symbols in the language the machine M reads
- s is the start state for the machine M
- $\delta = (Q \times \Sigma) \to (Q \cup \{acc, rej\} \times \Sigma \times \{L, R, S\})$ is the transition function.

A Turing Machine performs (potentially infinite) computation and halts when the state is either acc or rej.

Remember the restriction -- we cannot move to the right of the start symbol -- this is included in the definition to read invalid programs.

(c) Define the *configuration* of a Turing Machine at step t, and establish equations that specify the configuration of a k-symbol Turing machine at step $(t+1)$ in terms of the configuration of the previous step t.

This is just arbitrary (makes simulation easier) -- but you NEED to use this definition.

A Turing Machine Configuration is a triple (q, w, u) where $q \in Q$ is the state of the Turing Machine, $w = va \in \Sigma \times \Sigma^*$ is the list of symbols to the left of the tape head (including the symbol under the tape head) and u is the list of symbols to the right of the tape head.

A finite prefix of the symbols which cover all "non-blanks"

At each step, the Turing Machine inspects a (the last symbol in w) and performs the action specified by $\delta(q, a)$. If $\delta(q, a) = (q', \sigma, \{D\})$ then the Machine transitions into state q', sets the symbol underneath the head of the tape to σ and moves in direction D.

The initial configuration of a Turing Machine is $(s, \triangleright, t)$ where s is the start state, $\triangleright$ is the left endmarker.

With $c_t = (q, va, bu)$, and $\delta(q, a) = (q', \sigma, \{D\})$ the configuration at time $t+1$, c_{t+1} is given by:

This assumes U is non-empty -- you need a case for bu is empty! Just add some explanation.

$$c_{t+1} = \begin{cases} (q', v, \sigma bu) & \text{if } D = L \\ (q', v\sigma, bu) & \text{if } D = S \\ (q', v\sigma b, u) & \text{if } D = R \end{cases}$$

2 2005 Paper 3 Question 7

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2005p3q7.pdf

Show the contradiction in words!

Too formal

(a) Explain informally, i.e without reference to any particular model of computation, why the *Halting Problem* is undecidable.

By assuming the existence of a program that solves the Halting problem, we can derive a contradiction.

Algorithms can be represented as integers. Assume there is some program P which solves the halting problem. When $P(e, a)$ is passed the integer representation of a program e and its arguments a, returns 1 if the program halts when ran with those arguments and 0 if it does not halt.

A register machine has \mathbb{N}^n states -- while a turing machine has |Σ|^\mathbb{N} states.. uncountable
Because of how we define the input, the union of all inputs to a Turing machine is countable. So we only have countably many inputs.

Composition is with a function and a list of functions.
Primitive recursion is bounded iteration (for loop)
Minimization is unbounded iteration (while loop)

Primitive recursive:
start with primitive recursion, and composing them with primitive recursion you get the set of primitive recursive functions.

Partial recursion:
primitive recursion but also closed under the minimusation operator μ^n.

Total recursive functions:
partial recursive but are total

Primitive recursive \subseteq total recursive \subseteq partial recursive
Ackermann is total but not primitive resursive

Don't use mathematical terms when talking about Turings Theorem:
The idea is to include any algorithm (deterministic and finite) and has some result or may not terminate.

We can formalise random algorithms by adding a two-tape Turing machine. You then use this 2nd tape to generate random numbers.

Computation starts in the start state and is either infinite or halts in acc / rej state
Computation halts iff it is finite

2.5 Exercise 8

$$one^n = \mathrm{succ} \circ \mathrm{zero}^n$$
$$add = \rho^1(\mathrm{proj}_1^1, succ \circ \mathrm{proj}_3^3])$$
$$mult = \rho^1(\mathrm{zero}^1, add \circ [\mathrm{proj}_1^3, \mathrm{proj}_3^3])$$
$$exp = \rho^1(one^1, mult \circ [\mathrm{proj}_1^3, \mathrm{proj}_3^3])$$
$$pred = \rho^0(\mathrm{zero}^0, \mathrm{proj}_1^2)$$
$$minus = \rho^1(\mathrm{proj}_1^1, pred \circ \mathrm{proj}_3^3)$$
$$ifzero = \rho^2(\mathrm{proj}_1^2, \mathrm{proj}_2^4) \circ (\mathrm{proj}_2^3, \mathrm{proj}_3^3, \mathrm{proj}_1^3)$$
$$g = \rho^n(\mathrm{zero}^n, add \circ [\mathrm{proj}_{n+2}^{n+2}, f \circ (\mathrm{proj}_1^{n+2}, .., \mathrm{proj}_{n+1}^{n+2})])$$

Consider the algorithm A' which when run with argument e, runs $A(\_A'\hat{}, \_e\hat{})$ loops infinitely if and only if it returns 1.

$$\begin{aligned}&A'(\_e\hat{}) \text{ halts} \implies\\ &A(\_A'\hat{}, \_e\hat{}) \text{ returns } 0 \implies\\ &A'(\_e\hat{}) \text{ does not halt}\end{aligned}$$

So the program A' ran with any argument $\_e\hat{}$ halts if and only if the program A' ran with arguments $\_e\hat{}$ does not halt. This is a contradiction. Since, A' was constructed (computably) from the algorithm A we can conclude that no algorithm A can exist. Therefore, there exists no algorithm that solves the Halting Problem.

(b) Briefly describe two mathematical problems, other than the *Halting Problem* that are algorithmically undecidable.

Two mathematical problems which are algorithmically undecidable are the Entscheidungsproblem and whether or not there exist solutions to Diophantine Equations.

- The Entscheidungsproblem asks for a proof there is an algorithm which can determine in a finite number of steps whether or not there exists a proof for any mathematical formula. This is an undecidable problem – no such algorithm exists.
- Whether or not there is an integer solution to a Diophantine Equation is undecidable.

  A Diophantine equation is an equation consisting only of constants multiplied by polynomial terms. For example $3x^2y^9z - 93x^4y^3 = 0$.

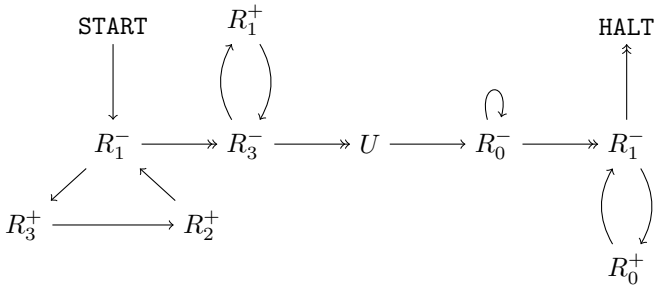
3 Exercise Sheet

4. Show that there is a register machine computable partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ such that both $\{x \in \mathbb{N} \mid f(x) \downarrow\}$ and $\{y \in \mathbb{N} \mid (\exists x \in \mathbb{N}) f(x) = y\}$ are register machine undecidable.

$$f(e) = \begin{cases} e & \text{if } \phi_e(e) \downarrow \end{cases}$$

The definition of a register machine computable function is as follows:
A function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ register machine computable if and only if there exists a register machine with at least $n + 1$ registers which when ran with $R_0 = 0$, $R_1 = x_1, \ldots, R_n = x_n$ and all other registers zeroed will terminate with $R_0 = y$ if and only if $f(x) = y$.

With U as the Universal Register Machine, this register machine is given below:



If $\phi_e(e)$ does not halt, neither does this register machine. And if $\phi_e(e)$ does halt, then this register machine halts with that value. Therefore, this register machine computes the value of f.

S_1=S_2

If $\{x \in \mathbb{N} \mid f(x)\downarrow\}$ is decidable then we can solve the halting problem. $\{y \in \mathbb{N} \mid (\exists x \in \mathbb{N}) f(x) = y\}$ is similar – if it were decidable then the halting problem would be solved. However, since the halting problem is uncomputable, a contradiction is reached and therefore both sets are uncomputable.

5. Suppose S_1 and S_2 are subsets of $\mathbb{N}$. Suppose $f \in \mathbb{N} \to \mathbb{N}$ is a register machine computable function satisfying: for all $x \in \mathbb{N}$, $x \in S_1 \iff f(x) \in S_2$. Show that S_1 is register machine decidable if S_2 is.

   The characteristic function of a set is total

   Assume S_1 is register machine decidable. Take an arbitrary x. Since f is register-machine computable, we can compute $f(x)$. S_2 is decidable, therefore we can determine whether $f(x)$ is in S_2. So we can determine whether x is in S_1. Since x was arbitrary, we can conclude that if S_1 is register machine decidable, then S_1 is register machine decidable.

6. Show that the set of codes $\langle e, e' \rangle$ of pairs of numbers e and e' satisfying $\phi_e = \phi_{e'}$ is undecidable.

   Easiest to compare with a program that NEVER halts -- in which case we just determine whether the program halts.

   Let e be the integer representation of the register machine which is constantly 0. Let e' be arbitrary. In order to determine whether $\phi_{e'} = \phi_e$, we must now determine whether $\phi_{e'}$ halts with value 0 on all inputs. Consider an arbitrary input x. We must now determine whether $\phi_{e'}(x)$ halts with value 0. So we must establish whether $\phi_{e'}(x)$ halts – this is uncomputable. Therefore, it's uncomputable to determine whether $\phi_{e'} = \phi_e$ and therefore the set of codes $\langle e, e' \rangle$ of pairs such that $\phi_e = \phi_{e'}$ is undecidable.

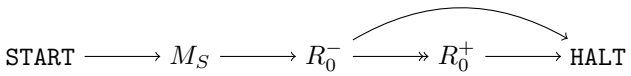
could be more formal

4 Decidable Sets

(a) Show that decidable sets are closed under union, intersection and complementation.

A set S is decidable if and only if its characteristic function $\chi_S(x) = \begin{cases} 0 & \text{if } x \in S \\ 1 & \text{otherwise} \end{cases}$ is computable. Let M_S be a register machine which computes the characteristic function for the set S.

Define the new register machine M'_S as follows:



The result of the register machine M'_S is given by:

$$M'_S(x) = \begin{cases} 0 & \text{if } M_S(x) = 1 \\ 1 & \text{otherwise} \end{cases} = \begin{cases} 0 & \text{if } x \in S \\ 1 & \text{otherwise} \end{cases} = \begin{cases} 1 & \text{if } x \in \bar{S} \\ 0 & \text{otherwise} \end{cases}$$

So the machine M'_S computes the complement of the set S. Since S was an arbitrary decidable set we can conclude that decidability is closed under complementation.

what about union and intersection?

(b) Do these closure properties hold for undecidable languages?

The closure property only holds for complementation.

Assume there exists some arbitrary undecidable set S such that the complement of S, $\bar{S}$ is decidable. If $\bar{S}$ is decidable, then its characteristic function $\chi_{\bar{S}}$ is decidable. Decidability is closed under function composition. Therefore $f \circ \chi_{\bar{S}}$ is also computable for $f = \{(0,1),(1,0)\}$. So $f \circ \chi_{\bar{S}} = \chi_S$. So the characteristic function for S is computable and therefore S is decidable. However, this contradicts the original assumption that S was undecidable. Therefore, there can exist no undecidable set S such that its complement $\bar{S}$ is computable. Hence undecidability is closed under complementation.

I prove "undecidable languages" are not closed under ... "undecidability" is a property -- only sets can be closed or not closed.

I prove undecidability is not closed under intersection and union by example. Consider the undecidable set S and its complement $\bar{S}$.

$S \cap \bar{S} = \emptyset - \emptyset$ is decidable and therefore undecidability is not closed under intersection.

$S \cap \bar{S} = \mathbb{N} - \mathbb{N}$ is decidable and therefore undecidability is not closed under intersection.

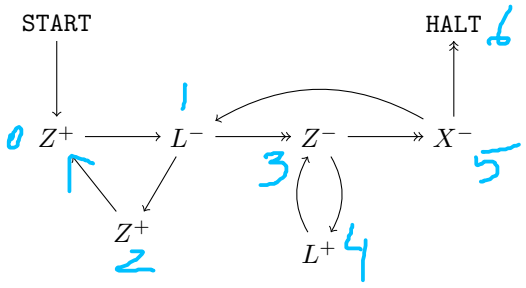
Remember to give counterexamples in the exam

5 2006 Paper 3 Question 7

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2006p3q7.pdf

(a) Give a graphical representation of the following register machine program.

$$
\begin{aligned}
&L0 : Z^+ \to L1\\
&L1 : L^- \to L2, L3\\
&L2 : Z^+ \to L0\\
&L3 : Z^- \to L4, L5\\
&L4 : L^+ \to L3\\
&L5 : X^- \to L1, L6\\
&L6 : \texttt{HALT}
\end{aligned}
$$



(b) Assuming the contents of register Z is initially 0, when the program is run starting at instruction $L0$ what functions of the initial contents of registers X and L are computed in X and L when the machine halts.

yep :)

This is `PUSH` from the lectures. It pushes the contents of X onto the list held in L. So given an initial state of (X, L), the final state is given by $2^X(2 \cdot L + 1)$.

(c) (i) What is meant by a Turing Machine, it's *configurations*, *transition relation* and the *computations* is carries out? What does it mean to say that a computation *halts*.

A Turing Machine is a theoretical model of computation which recognises the language of recursively enumerable grammars – a function is computable if and only if it can be computed on a Turing Machine.

Turing Machines can be described as a quadruple: (Q, Σ, s, δ). Q is the set of states the Turing Machine can be in (N.B. $acc, rej \in Q$), Σ is the set of symbols which can be on the Turing Machine tape, $s \in Q$ is the initial state and $\delta \subseteq (\Sigma \times Q) \to (Q \times \Sigma \times \{L, S, R\})$ is the transition function.

$Q \supseteq \{\texttt{acc}, \texttt{rej}\}$

what about restrictions on it?

A Turing Machine configuration is given by a triple (q, w, u), where q is the state the Turing Machine is currently in, $w \in \{\triangleright\} \times \Sigma^*$ is the list of symbols on the tape to the left of the head (including the one under the head) and $u \in \Sigma^*$ is the list of non-blank symbols to the right of the head.

A Turing Machine starts in the configuration $(s, \triangleright, u)$ and follows the transition function. If the Turing Machine is in state (q, wa, u) and $\delta(a, u) = (q', a', d)$ then the Turing Machine moves into state q', overwrites the current symbol with a' and moves in direction d.

A computation halts if the Turing Machine state ever reaches acc or rej.

(ii) Given a Turing Machine, is it decidable whether or not for all possible initial configurations the machine will not halt after 100 steps of transitions? Justify your answer.

It is decidable. A Turing Machine with less than 100 transitions looks at at most 100 symbols on the tape.

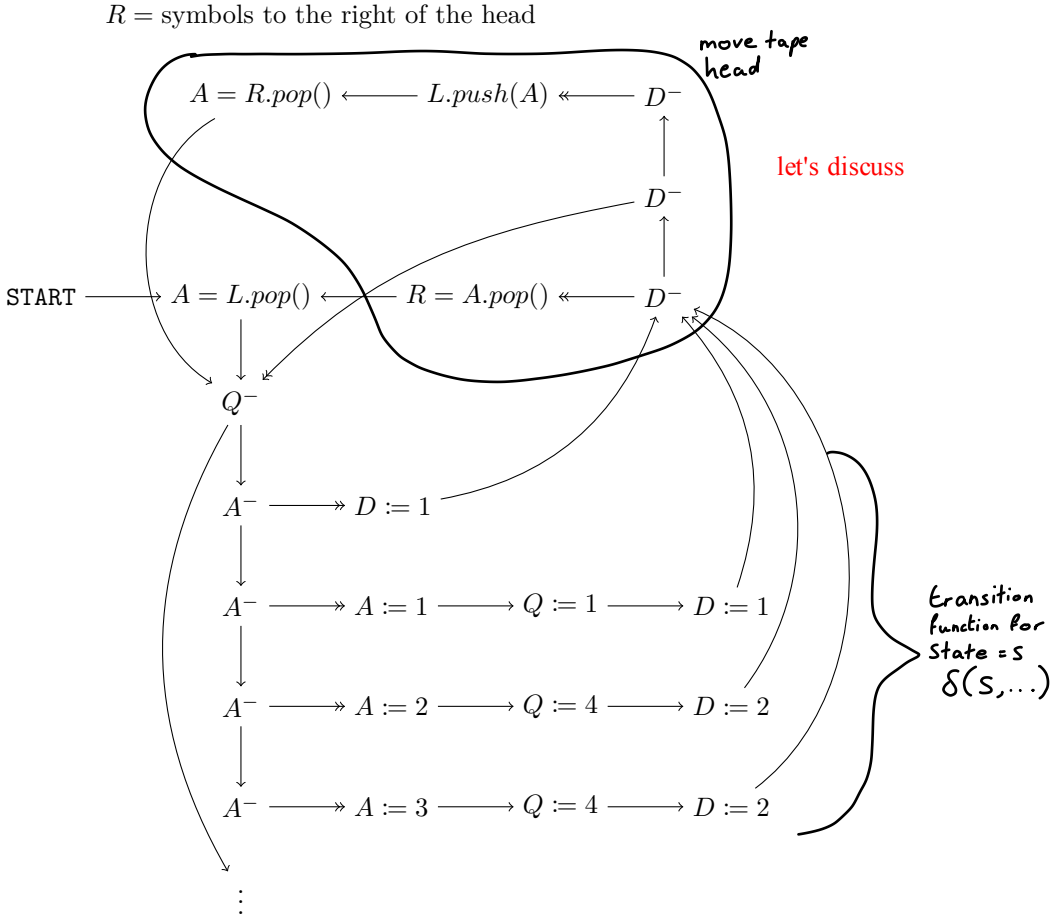
Number of states is |-1|^{100}

Since the number of possibilities that symbol can be is finite, the set of initial configurations distinguishable within 100 steps is also finite $\left(|\Sigma|^{100}\right)$. So we can enumerate all distinguishable initial configurations and check whether the Turing Machine halts for all of them. This method applies for any *finite* number of steps n.

6 Exercise Sheet

7. For the example Turing machine given on slide 64, give the register machine program implementing $(S, T, D) := \delta(S, T)$ as described on slide 70. [Tedious! – maybe just do a bit.]

$Q =$ current state , $\{s \mapsto 0, q \mapsto 1, q' \mapsto 2, acc \mapsto 3, rej \mapsto 4\}$
$A =$ current tape symbol , $\{\triangleright \mapsto 0, \_ \mapsto 1, 0 \mapsto 2, 1 \mapsto 3\}$
$D =$ current direction of tape head , $\{L \mapsto 0, R \mapsto 1, S \mapsto 2\}$
$L =$ symbols to the left of the head
$R =$ symbols to the right of the head



I've shown only part of the diagram. The other cases continue where ⋮ is. There will be two more cases similar to the one shown for q and q'. The cases for `acc` and `rej` will halt. For brevity, I've omitted `halts` in D^- and A^- where they're not possible due to invariants maintained by the Turing Machine. I further assume that $X = L.pop()$ will set X to 0 if $L = 0$ (the list corresponding to the integer L is empty).

8. Show that the following recursive functions are all primitive recursive

   try converting them to the standard representation

   (a) Exponentiation, $\exp \in \mathbb{N}^2 \to \mathbb{N}$.

$$\begin{cases} \text{add}(x, 0) & \equiv \text{succ} \circ x \\ \text{add}(x, y+1) & \equiv \text{succ} \circ \text{add}(x, y) \end{cases}$$

$$\begin{cases} \text{mul}(x, 0) & = \text{zero}^1(x) \\ \text{mul}(x_1, x+1) & = \text{add}(x_1, \text{mul}(x_1, x)) \end{cases}$$

$$\begin{cases} \exp(x_1, 0) & \equiv \text{succ} \circ \text{zero}^1 \\ \exp(x_1, x+1) & \equiv \text{mul}(x_1, \exp(x_1, x)) \end{cases}$$

   (b) Truncated subtraction, $minus \in \mathbb{N}^2 \to \mathbb{N}$.

$$\begin{cases} \text{pred}(0) & \equiv \text{zero}^0 \\ \text{pred}(x+1) & \equiv \text{proj}_1^2(x, \text{pred}(x)) \end{cases}$$

why use proj here?

$$\begin{cases} \text{minus}(x_1, 0) & \equiv \text{zero}^1 \\ \text{minus}(x_1, x+1) & \equiv \text{pred} \circ \text{minus}(x, y) \end{cases}$$

   (c) Conditional branch on zero, $\text{ifzero} \in \mathbb{N}^3 \to \mathbb{N}$.

$$\begin{cases} \text{ifzero}(0, y, z) & \equiv \text{proj}_1^2(y, z) \\ \text{ifzero}(x+1, y, z) & \equiv \text{proj}_3^3(\text{ifzero}(x, y, z), y, z) \end{cases}$$

not the definition

why use proj here?

   (d) Bounded summation: if $f \in \mathbb{N}^{n+1} \to \mathbb{N}$ is primitive recursive then so is $g \in \mathbb{N}^{n+1} \to \mathbb{N}$ where

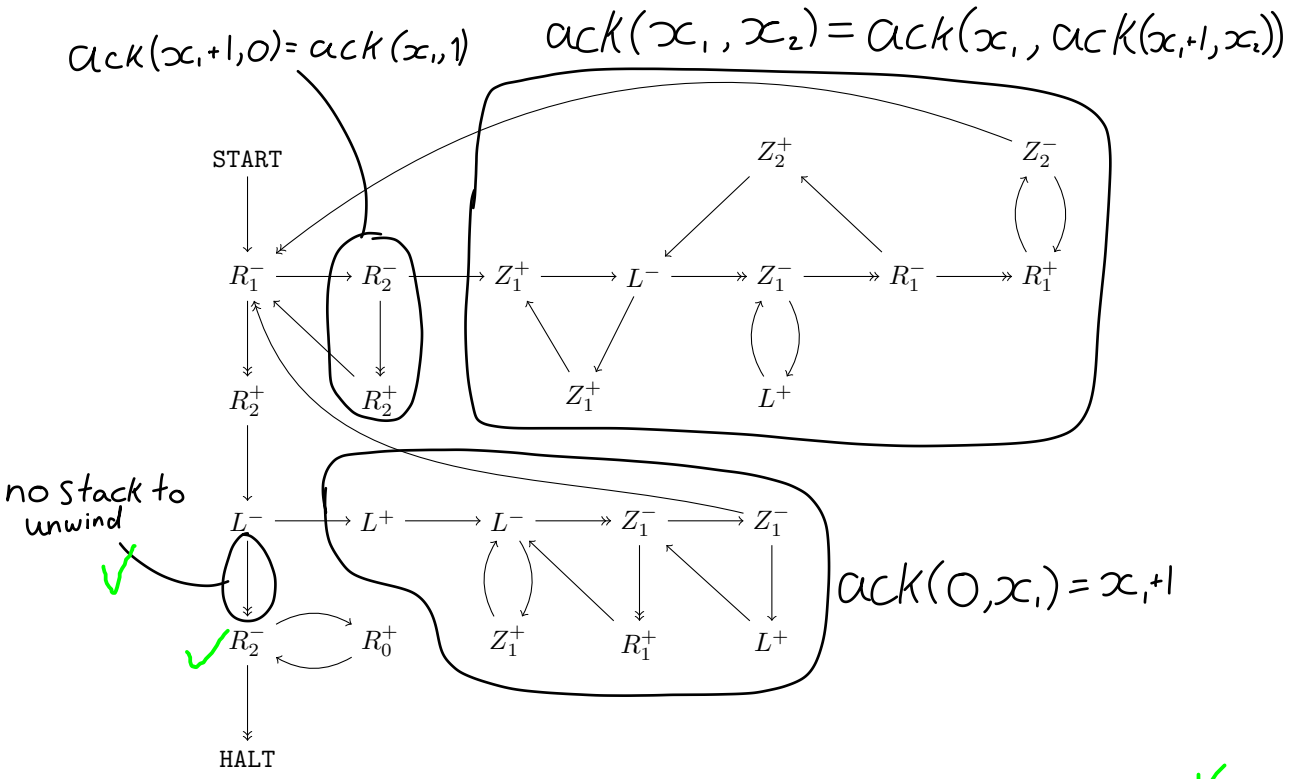
$$\begin{cases} g(\overrightarrow{x}, 0) & \equiv \text{zero}^n(\overrightarrow{x}) \\ g(\overrightarrow{x}, x+1) & \equiv \text{add}(f(\overrightarrow{x}, x), g(\overrightarrow{x}, x)) \end{cases}$$

9. Recall the definition of Ackermann's function *ack*. Sketch how to build a register machine M that computes $ack(x_1, x_2)$ in R_0 when started with x_1 in R_1 and x_2 in R_2 ad all other registers zero.

   Using registers as follows:

$$\begin{aligned} R_1 &= x_1 \text{ in current call to Ack} \\ R_2 &= x_2 \text{ in current call to Ack} \\ L &= \text{stack of } x_2\text{s} \\ Z_1 &= \text{temporary} \\ Z_2 &= \text{temporary} \end{aligned}$$

   Whenever the top-left R_1^- is run, the machine is at the start of a function call. If it has to recurse, it pushes the value of x_1 for the outermost call onto L and performs the innermost call. In the base case $Ack(0, x_2) = x_2 + 1$, if the stack is empty, return $x + 1$ else pop the top of the stack into x_1.



7 Minimisation

Use minimisation to show that the following partial functions are partial recursive:

1. The binary maximum function $\max : \mathbb{N}^2 \to \mathbb{N}$

use standard representation

$$\begin{cases} \max(x, 0) & \equiv x \\ \max(x, y) & \equiv \text{ifzero}(\text{pred}(\text{minus}(x, y)), \text{succ}(y), x) \end{cases}$$

2. The integer square root function $\text{sqrt}\mathbb{N} \to \mathbb{N}$, which is only defined if its argument is a perfect square.

$$\begin{cases} \text{equals}(x_1, 0) & \equiv \text{ifzero}(x_1, 0, 1) \\ \text{equals}(x_1, x + 1) & \equiv \text{ifzero}(\max(x_1, x), 0, 1) \end{cases}$$

$$\begin{cases} \text{sqrt}(0) & \equiv 0 \\ \text{sqrt}(x + 1) & \equiv \mu^2 \text{equals}(x + 1, \text{mul}(i, i)) \end{cases}$$

In exams, are we allowed to assume pred is defined in lectures?
Yes.
You can use any functions defined in lecture notes. Just say "x is primitive because it was defined in the lecture notes"

in rho^n, n is the number of constants (rho^n means |\overrightarrow{x}| is 2)
rho^n means "partial recurse with these functions".
If you define a partial function, it doesn't have to use rho^n

\mu^n means "n constants and the last variable is minimised"
\mu^n is a n argument function (the last is minimised)
\rho^n is a n + 1 argument function (we iterate on the last argument)

proj^n_n is the value of the previous iteration.
proj^n_{n-1} is the value of the counter

Don't think about primitive recursion as "recursion" -- think about it as starting with a base case and applying the second function as many times as you need to to get to the initial iterator.
Defined means defined -- not "any value"