Math 114C: Homework 2

Due on January 29, 2026 at 11:59pm

Professor Arant

Darsh Verma

Problem 1

We use that the divisibility relation $y \mid x$ is recursive. Define:

$$\text{Prime}(x) \iff x > 1 \land (\forall y)_{1<y<x}[\neg(y \mid x)]$$

This can be written as:

$$\text{Prime}(x) \iff x > 1 \land (\forall y)_{y<x}[y \leq 1 \lor \neg(y \mid x)]$$

Since recursive relations are closed under:

- Boolean operations ($\land$, $\lor$, $\neg$)
- Bounded quantification ($(\forall y)_{y<x}$)
- Comparison relations ($>$, $\leq$)

and $y \mid x$ is recursive, we can conclude that $\text{Prime}(x)$ is recursive. $\square$

Problem 2

Define f by:

$$f(0) = \mu z[z \in A]$$
$$f(n+1) = \mu z[z \in A \wedge z > f(n)]$$

f is total: Since A is infinite, for any n, there exist infinitely many elements of A greater than $f(n)$. Hence the search $\mu z[z \in A \wedge z > f(n)]$ always terminates.

f is recursive: Define the auxiliary function:

$$g(n, y) = \begin{cases} \mu z[z \in A] & \text{if } n = 0 \\ \mu z[z \in A \wedge z > y] & \text{if } n > 0 \end{cases}$$

Then we can define f by primitive recursion:

$$f(0) = \mu z[z \in A]$$
$$f(n+1) = \mu z[z \in A \wedge z > f(n)]$$

Since A is recursive, "$z \in A$" is a recursive relation. The relation "$z \in A \wedge z > y$" is also recursive. By closure under bounded minimization, f is recursive. $\square$

Problem 3

For a sequence code $u = \langle x_0, \ldots, x_{n-1} \rangle$:

- $\mathrm{lh}(u)$ gives the length n (recursive)
- $(u)_i$ gives the i-th element x_i (recursive)
- $\mathrm{Seq}(u)$ determines if u is a valid sequence code (recursive)

Define:

$$R(u) \iff \mathrm{Seq}(u) \wedge (\forall i)_{i < \mathrm{lh}(u) \dot{-} 1}[(u)_i < (u)_{i+1}]$$

This relation is recursive because:

- $\mathrm{Seq}(u)$ is recursive
- $(u)_i$ and $(u)_{i+1}$ are recursive functions
- $<$ is a recursive relation
- $\mathrm{lh}(u) \dot{-} 1$ is recursive
- Bounded universal quantification preserves recursiveness
- Conjunction preserves recursiveness

Therefore R is recursive. $\square$

Problem 4

($\Rightarrow$) Assume R is recursive. Then:

$$R'(u) \iff \text{Seq}(u) \wedge \text{lh}(u) = n \wedge R((u)_0, (u)_1, \ldots, (u)_{n-1})$$

Since Seq(u), lh(u), and $(u)_i$ are all recursive, and R is recursive, R' is a Boolean combination of recursive relations and hence recursive.
($\Leftarrow$) Assume R' is recursive. Then:

$$R(x_0, \ldots, x_{n-1}) \iff R'(\langle x_0, \ldots, x_{n-1} \rangle)$$

Since the sequence coding function $\langle x_0, \ldots, x_{n-1} \rangle$ is recursive, and R' is recursive, the composition is recursive. Hence R is recursive. $\square$

Problem 5

Define $F : \mathbb{N}^{k+1} \rightharpoonup \mathbb{N}$ by:

$$F(\vec{x}, y) = \langle f_0(\vec{x}, y), f_1(\vec{x}, y) \rangle$$

We show F satisfies a standard primitive recursion. For the base case:

$$F(\vec{x}, 0) = \langle f_0(\vec{x}, 0), f_1(\vec{x}, 0) \rangle = \langle g_0(\vec{x}), g_1(\vec{x}) \rangle$$

For the recursive case, note that from $F(\vec{x}, y) = \langle f_0(\vec{x}, y), f_1(\vec{x}, y) \rangle$, we can extract:

$$\begin{aligned} f_0(\vec{x}, y) &= (F(\vec{x}, y))_0 \\ f_1(\vec{x}, y) &= (F(\vec{x}, y))_1 \end{aligned}$$

Then:

$$\begin{aligned} F(\vec{x}, y+1) &= \langle f_0(\vec{x}, y+1), f_1(\vec{x}, y+1) \rangle \\ &= \langle h_0(\vec{x}, y, f_0(\vec{x}, y), f_1(\vec{x}, y)), h_1(\vec{x}, y, f_0(\vec{x}, y), f_1(\vec{x}, y)) \rangle \\ &= \langle h_0(\vec{x}, y, (F(\vec{x}, y))_0, (F(\vec{x}, y))_1), h_1(\vec{x}, y, (F(\vec{x}, y))_0, (F(\vec{x}, y))_1) \rangle \end{aligned}$$

So F satisfies:

$$\begin{aligned} F(\vec{x}, 0) &= G(\vec{x}) \quad \text{where } G(\vec{x}) = \langle g_0(\vec{x}), g_1(\vec{x}) \rangle \\ F(\vec{x}, y+1) &= H(\vec{x}, y, F(\vec{x}, y)) \end{aligned}$$

where $H(\vec{x}, y, z) = \langle h_0(\vec{x}, y, (z)_0, (z)_1), h_1(\vec{x}, y, (z)_0, (z)_1) \rangle$.
Since g_0, g_1, h_0, h_1 are recursive, and sequence coding/decoding are recursive, G and H are recursive. By closure under primitive recursion, F is recursive.
Finally, $f_0(\vec{x}, y) = (F(\vec{x}, y))_0$ and $f_1(\vec{x}, y) = (F(\vec{x}, y))_1$ are recursive by composition. $\square$

Problem 6

Part (a)
Using the recursive equations:

$$\begin{aligned} &A(1,1) = A(0, A(1,0)) \quad \text{(third equation with } m = 0, n = 0) \\ &A(1,0) = A(0,1) \quad \text{(second equation with } m = 0) \\ &A(0,1) = 1 + 1 = 2 \quad \text{(first equation)} \\ \text{So } &A(1,0) = 2 \\ &A(1,1) = A(0,2) = 2 + 1 = 3 \end{aligned}$$

Therefore $\boxed{A(1,1) = 3}$.

Part (b)
Outer induction on m: We prove $(\forall n)A(m,n)\downarrow$ for all m.
Base case ($m = 0$): For any n, $A(0,n) = n+1\downarrow$. ✓
Inductive step: Assume $(\forall n)A(m,n)\downarrow$ (IH). We prove $(\forall n)A(m+1,n)\downarrow$ by induction on n.

- *Base case ($n = 0$):* $A(m+1,0) = A(m,1)$. By IH (with $n = 1$), $A(m,1)\downarrow$. So $A(m+1,0)\downarrow$. ✓
- *Inductive step:* Assume $A(m+1,n)\downarrow$ (inner IH). Then:

  $$A(m+1,n+1) = A(m, A(m+1,n))$$

  By inner IH, $A(m+1,n)\downarrow$, say $A(m+1,n) = k$ for some $k \in \mathbb{N}$. By outer IH (with $n = k$), $A(m,k)\downarrow$. Therefore $A(m+1,n+1)\downarrow$. ✓

$A(m,n)\downarrow$ for all $m,n \in \mathbb{N}$. $\square$

Problem 7

Part (a)
($\Rightarrow$) By double induction on m and n:
Outer induction on m:
Base case ($m = 0$): For any n, $A(0, n) = n + 1$. The single-element sequence $((0, n, n + 1))$ is an Ackermann computation by condition (1). ✓
Inductive step: Assume for all n', if $A(m, n') = y'$, there is an Ackermann computation ending in (m, n', y'). We prove for $m + 1$ by inner induction on n:

- *Base ($n = 0$):* $A(m + 1, 0) = A(m, 1) = y$ for some y. By outer IH, there's an Ackermann computation C ending in $(m, 1, y)$. Append $(m + 1, 0, y)$ to C. This satisfies condition (2) with j being the index of $(m, 1, y)$. ✓
- *Inductive step:* Assume the result for n. Consider $A(m + 1, n + 1) = A(m, A(m + 1, n))$.

  Let $A(m + 1, n) = z$ and $A(m, z) = y$, so $A(m + 1, n + 1) = y$.

  By inner IH, there's computation C_1 ending in $(m + 1, n, z)$.

  By outer IH, there's computation C_2 ending in (m, z, y).

  Concatenate C_1, C_2, then append $(m + 1, n + 1, y)$. This is an Ackermann computation by condition (3), with j indexing $(m + 1, n, z)$ and ℓ indexing (m, z, y). ✓

($\Leftarrow$) By induction on the length k of the computation $(m_0, n_0, y_0), \ldots, (m_{k-1}, n_{k-1}, y_{k-1})$:
For the tuple (m_i, n_i, y_i) at position i, one of three conditions holds:

- Condition (1): $m_i = 0$ and $y_i = n_i + 1$. Then $A(0, n_i) = n_i + 1 = y_i$. ✓
- Condition (2): $m_i = m + 1$, $n_i = 0$, and $(m, 1, y_i)$ appears earlier. By IH on that earlier tuple, $A(m, 1) = y_i$. So $A(m + 1, 0) = A(m, 1) = y_i$. ✓
- Condition (3): $m_i = m + 1$, $n_i = n + 1$, and earlier tuples $(m + 1, n, z)$ and (m, z, y_i) appear. By IH, $A(m + 1, n) = z$ and $A(m, z) = y_i$. So $A(m + 1, n + 1) = A(m, A(m + 1, n)) = A(m, z) = y_i$. ✓

Therefore $A(m_{k-1}, n_{k-1}) = y_{k-1}$, i.e., $A(m, n) = y$. □

Part (b)
By part (a):

$$A(m, n) = y \iff \exists \text{ Ackermann computation } C \text{ ending in } (m, n, y)$$

Encode a triple (a, b, c) as $\langle a, b, c\rangle$ and a sequence of triples as $\langle\langle a_0, b_0, c_0\rangle, \ldots, \langle a_{k-1}, b_{k-1}, c_{k-1}\rangle\rangle$.
Define the relation $\mathrm{AckComp}(u)$ to hold iff u codes an Ackermann computation. This is recursive since we can check:

- u is a sequence code
- Each element codes a triple
- For each triple, one of conditions (1), (2), (3) holds (all involve recursive checks on earlier elements)

Define the relation:

$$R(m, n, y, u) \iff \mathrm{AckComp}(u) \wedge \mathrm{lh}(u) > 0 \wedge (u)_{\mathrm{lh}(u) \dot{-} 1} = \langle m, n, y\rangle$$

This is recursive. By part (a), $A(m, n) = y$ iff $\exists u\, R(m, n, y, u)$.

Problem 7 continued on next page. . .

Since A is total (Problem 6b), we can write:

$$A(m,n) = (\mu\langle y,u\rangle[R(m,n,y,u)])_0$$

using the pairing function. Alternatively:

$$A(m,n) = \mu y[\exists u \leq B(m,n,y)\, R(m,n,y,u)]$$

for some recursive bound B. Since A is total, the unbounded minimization:

$$A(m,n) = (\mu z[R(m,n,(z)_0,(z)_1)])_0$$

terminates, and A is recursive. $\square$

Problem 8

Part (a)
Initial: $R_0 = 2$, $R_1 = 0$, $R_2 = 0$, instruction $= 0$.

Step	I	R_0	R_1	R_2	Action
0	$T(0,1)$	2	2	0	Copy R_0 to R_1; go to 1
1	$S(0)$	3	2	0	$R_0 := R_0 + 1$; go to 2
2	$S(2)$	3	2	1	$R_2 := R_2 + 1$; go to 3
3	$J(1,2,5)$	3	2	1	$R_1 \neq R_2$; go to 4
4	$J(0,0,1)$	3	2	1	$R_0 = R_0$; go to 1
5	$S(0)$	4	2	1	$R_0 := R_0 + 1$; go to 2
6	$S(2)$	4	2	2	$R_2 := R_2 + 1$; go to 3
7	$J(1,2,5)$	4	2	2	$R_1 = R_2$; go to 5

Output: $R_0 = \boxed{4}$.
Part (b)
$\boxed{f_P^{(1)}(x) = 2x}$

Problem 9

Start with $R_0 = x$, $R_1 = 0$:

- I_0: $R_1 := 1$
- I_1: $R_1 := 2$
- I_2: If $R_0 = R_1$, halt (go to 4); else go to I_3
- I_3: Unconditional jump to I_0 (since $R_0 = R_0$ always)

So each pass through the loop adds 2 to R_1. After k complete loops: $R_1 = 2k$.
The program halts when $R_0 = R_1$, i.e., when $x = 2k$ for some $k \geq 1$.
Case x even ($x = 2k$, $k \geq 1$): After k iterations, $R_1 = 2k = x$, so $J(0,1,4)$ succeeds and the program halts with output $R_0 = x$. Thus $f_P^{(1)}(x) = x \downarrow$.
Case $x = 0$: After first loop, $R_1 = 2 \neq 0$. Since R_1 only increases by 2 each loop and is always ≥ 2, we never have $R_1 = 0$. So the program loops forever: $f_P^{(1)}(0) \uparrow$.
Case x odd: R_1 takes values $2, 4, 6, \ldots$, all even. Since x is odd, $R_0 \neq R_1$ always. The program loops forever: $f_P^{(1)}(x) \uparrow$.

$\boxed{f_P^{(1)}(x) \downarrow \iff x \text{ is even and } x > 0}$ $\square$

Problem 10

Part (a)
True.
Proof: The instruction $J(0,0,0)$ means: "if $R_0 = R_0$, jump to instruction 0."
Since $R_0 = R_0$ is always true (for any value in register 0), executing $I_0 = J(0,0,0)$ always jumps back to instruction 0. The program enters an infinite loop at the very first instruction and never advances past I_0.
Therefore, no matter what input is given and no matter what other instructions follow, the computation never halts. $\square$
Part (b)
True.
Proof sketch: Both computations start with the same initial configuration:

- $R_0 = x_0, \ldots, R_{n-1} = x_{n-1}$
- $R_n = R_{n+1} = \cdots = 0$

The only difference is in which registers are considered "input." But the URM execution depends only on register values, not on the arity designation. The computation proceeds identically in both cases, and both output R_0.
Hence $f_P^{(n+k)}(x_0, \ldots, x_{n-1}, 0, \ldots, 0) = f_P^{(n)}(x_0, \ldots, x_{n-1})$ (both defined or both undefined, with equal values when defined). $\square$
(c)
True.
Proof sketch: Given program $P = (I_0, \ldots, I_{s-1})$, we can create infinitely many equivalent programs by appending "dead code" that is never executed.
For each $m \geq 1$, define P_m by appending m instructions after P:

$$P_m = (I_0, \ldots, I_{s-1}, \underbrace{Z(0), Z(0), \ldots, Z(0)}_{m \text{ times}})$$

These extra instructions are never reached. So $f_{P_m}^{(1)} = f_P^{(1)}$ for all m.
Since there are infinitely many choices of m, there are infinitely many such programs. $\square$