Computability Exam Solutions

September 18, 2012

Exercise 1

Definition of Unbounded Minimization

Given a function $f : \mathbb{N}^{k+1} \to \mathbb{N}$, the unbounded minimization operation $\mu y.f(\vec{x},y)$ produces a function $g : \mathbb{N}^k \to \mathbb{N}$ defined by:

```
g(x⃗) = μy.f(x⃗,y) = {
  the least y such that f(x⃗,y) = 0  if such y exists
  ↑                                   otherwise
}
```

Proof that the set of computable functions is closed under unbounded minimization

Let $f : \mathbb{N}^{k+1} \to \mathbb{N}$ be computable, and define $g(\vec{x}) = \mu y.f(\vec{x},y)$.

Since f is computable, there exists a URM program P that computes f.

We construct a URM program Q that computes g as follows:

```
Algorithm for Q on input x⃗:
1. Initialize counter y = 0 in a working register
2. Loop:
   a. Compute f(x⃗,y) using program P
   b. If f(x⃗,y) = 0, return y
   c. Otherwise, increment y and repeat
```

Formal URM implementation:

- Store input $\vec{x}$ in registers $R_1,\ldots,R_k$
- Use register R_{k+1} for counter y (initialized to 0)
- Use additional registers for computation of f
- Use conditional jump to check if $f(\vec{x},y) = 0$
- If yes, move y to output register and halt
- If no, increment y and loop back

Since this algorithm systematically searches for the minimal y satisfying $f(\vec{x},y) = 0$, and uses only basic URM operations (which preserve computability), the function g is computable.

Therefore, the set of computable functions is closed under unbounded minimization.

Exercise 2

Question: Can there exist $f : \mathbb{N} \to \mathbb{N}$ with finite codomain, increasing, and non-computable?

Answer: No, such a function cannot exist.

Proof:

Let $f : \mathbb{N} \to \mathbb{N}$ be increasing (i.e., $x \leq y \implies f(x) \leq f(y)$) with finite codomain.

Since cod(f) is finite, let $\text{cod}(f) = \{c_1, c_2, \ldots, c_n\}$ where $c_1 < c_2 < \ldots < c_n$.

Since f is increasing and has finite codomain, f must eventually become constant. Specifically, $\exists N$ such that $\forall x \geq N$: $f(x) = c_n$ (the maximum value in the codomain).

Algorithm to compute f:

```
To compute f(x):
1. For i = 0, 1, 2, ..., x:
   - For each possible output value c ∈ {c₁, ..., cₙ}:
     - Check if assigning f(i) = c maintains the increasing property
     - Use brute force search within the finite possibilities
2. Since there are only finitely many valid increasing functions
   from {0,1,...,x} to {c₁,...,cₙ}, we can enumerate them all
3. Eventually we'll find the unique function f that matches the
   given constraints on the finite domain {0,1,...,x}
```

More precisely: Since f is increasing with finite codomain, the function is completely determined by the "jump points" where f changes value. There are only finitely many such configurations, making f computable by finite case analysis.

Therefore, no such non-computable function can exist.

Exercise 3

Classification of $A = \{x \in \mathbb{N} : \varphi_x(y) = y \text{ for infinitely many } y\}$

The set A is saturated since $A = \{x \mid \varphi_x \in \mathcal{A}\}$ where $\mathcal{A} = \{f \mid f(y) = y \text{ for infinitely many } y\}$.

A is not r.e.: We use Rice-Shapiro theorem. Consider the identity function $id \in \mathcal{A}$ since $id(y) = y$ for all y (hence infinitely many).

Consider any finite function $\theta \subseteq id$. If $\theta = \{(y_1,y_1), (y_2,y_2), \ldots, (y_k,y_k)\}$ for finitely many points, then $\theta(y) = y$ for exactly k points (finitely many), so $\theta \notin \mathcal{A}$.

Since $id \in \mathcal{A}$ and $\forall$ finite $\theta \subseteq id$: $\theta \notin \mathcal{A}$, by Rice-Shapiro theorem, A is not r.e.

$\bar{A}$ is not r.e.: Consider the constant function $f(x) = 0$. Then $f \notin \mathcal{A}$ since $f(y) = y$ only when $y = 0$ (finitely many: just one point).

Consider the finite function $\theta = \{(0,0)\} \subseteq f$. Then $\theta(y) = y$ for exactly one value ($y = 0$), so $\theta \notin \mathcal{A}$.

But we need $\theta \in \mathcal{A}$ for Rice-Shapiro to apply to $\bar{A}$. Let me reconsider.

Actually, consider any function $g \notin A$. For $\bar{A}$ to be not r.e. by Rice-Shapiro, we need: $\exists g \notin A$ such that $\forall$ finite $\theta \subseteq g$: $\theta \in A$.

But any finite function can equal the identity on at most finitely many points, so no finite function is in A.

Let me use a different approach. Since A is saturated and by Rice's theorem A is not recursive ($A \neq \emptyset$ since $id \in A$, and $A \neq \mathbb{N}$ since constant functions $\notin A$). Since A is not r.e., we have that A is not recursive but not r.e., which means $\bar{A}$ is also not r.e.

Final classification: A and $\bar{A}$ are both not r.e. (and hence not recursive).

Exercise 4

Classification of $B = \{x \in \mathbb{N} : f(x) \in E_x\}$

where $f : \mathbb{N} \to \mathbb{N}$ is a fixed total computable function.

B is r.e.:

```
scB(x) = 1(μ(y,t). S(x,y,f(x),t))
```

This searches for y,t such that $\varphi_x(y) = f(x)$ in exactly t steps, confirming $f(x) \in E_x$.

B is not necessarily recursive: The recursiveness of B depends on the specific function f.

Example where B is not recursive: Let f be the function that maps each x to x itself, i.e., $f(x) = x$. Then $B = \{x : x \in E_x\} = \{x : x \in cod(\varphi_x)\}$.

We can reduce from the halting problem. The classification depends on the specific properties of f.

Example where B is recursive: If f is a constant function, say $f(x) = 0$ for all x, then: $B = \{x : 0 \in E_x\}$

This set is r.e. (as shown above) and may or may not be recursive depending on further analysis.

General analysis: Since f is total and computable, the semi-characteristic function of B is computable, so B is always r.e.

For recursiveness, we need to analyze whether we can effectively determine when $f(x) \notin E_x$. This generally requires knowing when φ_x never outputs f(x), which is typically undecidable.

Typical classification: B is r.e. but not recursive; $\bar{B}$ is not r.e.

Exercise 5

Theorem: $f : \mathbb{N} \to \mathbb{N}$ is computable $\Longleftrightarrow$ $A_x = \{\pi(x, f(x)) : x \in \mathbb{N}\}$ is r.e.

where $\pi : \mathbb{N}^2 \to \mathbb{N}$ is the pair encoding function.

Proof:

(⇒) If f is computable, then A_x is r.e.

If f is computable, then $A_x = \{\pi(x, f(x)) : x \in \mathbb{N}\}$ is r.e. because:

```
scAₓ(z) = 1(μx. π(x, f(x)) = z)
```

Since f is computable and π is computable, this semi-characteristic function is computable.

Alternatively, A_x is the range of the computable function $g(x) = \pi(x, f(x))$, and ranges of computable functions are r.e.

(⇐) If A_x is r.e., then f is computable.

Suppose A_x is r.e. We need to show f is computable.

Since A_x is r.e., ∃ computable function h such that $A_x = \text{range}(h)$.

This means: $\forall x \in \mathbb{N}, \exists t$ such that $h(t) = \pi(x, f(x))$.

To compute f(x):

```
1. Systematically enumerate h(0), h(1), h(2), ...
2. For each h(t), compute π⁻¹(h(t)) = (a,b)
3. If a = x, then b = f(x), so return b
4. Since π(x, f(x)) ∈ Aₓ = range(h), this process will eventually terminate
```

The key insight is that for each x, there exists exactly one pair (x, f(x)) in A_x with first component x. Since A_x is r.e., we can enumerate its elements until we find the unique pair starting with x.

Therefore, f is computable.

Conclusion: f is computable $\iff$ A_x is r.e.