Non-Saturated Sets: Identification and Characterization

Definition and Core Concept

A set $A \subseteq \mathbb{N}$ is **saturated** if for all $x, y \in \mathbb{N}$:

$$x \in A \land \phi_x = \phi_y \implies y \in A$$

Equivalently, A is saturated iff $A = \{x \mid \varphi_x \in \mathcal{A}\}$ for some $\mathcal{A} \subseteq \mathcal{F}$ where $\mathcal{F}$ is the set of all partial computable functions.

A set is **non-saturated** when this property fails—when there exist indices computing the same function but having different membership status in the set.

Systematic Identification of Non-Saturated Sets

Method 1: Direct Counterexample Construction

To prove A is non-saturated, find $m, n \in \mathbb{N}$ such that:

- $\varphi_m = \varphi_n$ (same computed function)
- $m \in A \land n \notin A$ (different membership status)

Method 2: The Halting Set Pattern

Classic Example: $K = \{x \mid \varphi_x(x) \downarrow\}$

K is non-saturated because:

1. Construct a function φ_m where $\varphi_m(x) = \{1$ if $x = m$; $\uparrow$ otherwise$\}$
2. Then $m \in K$ since $\varphi_m(m) = 1 \downarrow$
3. Since any computable function has infinitely many indices, $\exists n \neq m$ such that $\varphi_n = \varphi_m$
4. But $\varphi_n(n) = \varphi_m(n) = \uparrow$, so $n \notin K$
5. Therefore: $\varphi_m = \varphi_n$ but $m \in K \land n \notin K$

Method 3: Syntactic vs Semantic Properties

Key Insight: Sets depending on program syntax rather than computed function are typically non-saturated.

Examples of Non-Saturated Sets:

- Length-based: $LEN_{10} = \{n \mid$ program P_n has length $\leq 10\}$
- Timing-based: $T_2 = \{e \mid P_e(e)$ terminates in exactly 2 steps$\}$
- Self-reference patterns: $K = \{e \mid e \in W_e\}$

Recognition Patterns

Pattern 1: $\varphi_x(x)$ Dependencies

Sets of the form {x | $\varphi_x(x)$ satisfies property P} are often non-saturated because:

- The property depends on applying the function to its own index
- Different indices of the same function behave differently when applied to themselves

Pattern 2: Index-Dependent Properties

If the set definition explicitly uses the index x in a way that's not purely functional:

- $\{x \mid x \in W_x\}$
- $\{x \mid \varphi_x(x) = x\}$
- {x | x appears in the codomain E_x}

Pattern 3: Complexity/Resource Bounds

Sets involving computational resources (time, space, program length) typically non-saturated:

- Different programs computing the same function may have different complexities
- The property depends on the specific implementation, not the function

Formal Verification Technique

To verify non-saturation of set A:

1. **Identify the problematic pattern**: Look for self-reference or index dependency
2. **Construct the witness function**: Find/construct a specific function that demonstrates the issue
3. **Use infinitude of indices**: Leverage that every computable function has infinitely many indices
4. **Apply the Second Recursion Theorem**: Often needed for rigorous construction of the counterexample

Common Non-Saturated Sets in Exercises

1. **K = $\{x \mid \varphi_x(x)\downarrow\}$** - Classic halting set
2. **$\{x \mid x \in W_x\}$** - Self-membership
3. **$\{x \mid \varphi_x(x) = x\}$** - Fixed-point property
4. **{x | |program_x| ≤ k}** - Syntactic length bounds
5. **{x | φ_x terminates in ≤ t steps on input x}** - Resource bounds

Quick Recognition Test

If a set A can be expressed as A = $\{x \mid \varphi_x \in \mathcal{A}\}$ for some $\mathcal{A} \subseteq \mathcal{F}$, then A is saturated.

If a set's definition inherently depends on the specific index x (not just the function φ_x), suspect non-saturation.

The statement "$\varphi_x(x)\downarrow$ or something" you mentioned to your student correctly identifies a key pattern—when the property depends on applying the function to its own index, this creates the index-dependence that typically breaks saturation.