1. Proving Non-Computability

Using K-Reduction

To prove a set A is not recursive/computable, we can show $K \leq_m A$ by:

1. Define a computable function g(x,y) based on K's properties
2. Use the s-m-n theorem to get a total computable function s
3. Show s reduces K to A by proving:
   - $x \in K \Rightarrow s(x) \in A$
   - $x \notin K \Rightarrow s(x) \notin A$

Example:

```
Let A = {x | ϕₓ(x) > x}
Show K ≤ₘ A:
g(x,y) = x + 1 if x ∈ K, undefined otherwise
By s-m-n theorem get s, then:
x ∈ K ⇒ ϕₛ(ₓ)(s(x)) = s(x) + 1 > s(x) ⇒ s(x) ∈ A
x ∉ K ⇒ ϕₛ(ₓ)(s(x)) undefined ⇒ s(x) ∉ A
```

Using Diagonalization

For proving non-computability directly:

1. Assume f is computable
2. Define a function g differing from f at some point
3. Show g must be computable, leading to contradiction

Example:

```
Assume f computable where f(x) = ϕₓ(x) + 1
Define g(x) = f(x) + 1
Then g(x) = ϕₓ(x) + 2
But g must be computable, contradicting g ≠ ϕₓ for any x
```

2. Working with Predicates

Semi-Decidable Predicates

Key theorems:

- Structure theorem: Q($\bar{x}$) semi-decidable ⇔ ∃R decidable s.t. Q($\bar{x}$) = ∃y.R($\bar{x}$,y)
- Projection theorem: If P(x,$\bar{y}$) semi-decidable then ∃x.P(x,$\bar{y}$) semi-decidable

Apply by:

1. Express predicate in terms of H or S predicates
2. Use composition of semi-decidable predicates
3. Apply quantification closure properties

Example:

```
Q(x,y) = "ϕₓ(y) = z" is semi-decidable because:
Q(x,y) ⇔ ∃t.S(x,y,z,t)
```

3. Rice's Theorem and Saturation

Testing for Saturation

A set A is saturated if:

- It expresses a property of the computed function, not the program
- $\varphi_m = \varphi_n$ and m ∈ A implies n ∈ A

Example:

```
A = {x | ϕₓ total} is saturated
A = {x | ϕₓ(x) halts in 5 steps} is not saturated
```

Applying Rice's Theorem

To use Rice's theorem:

1. Show the set is saturated
2. Show it's non-trivial (≠ ∅ and ≠ N)
3. Conclude it's not recursive

Example:

```
Let A = {x | ϕₓ is total}
- Saturated: depends only on computed function
- Non-empty: contains constant functions
- Not all: doesn't contain undefined function
Therefore A is not recursive
```

Common Pitfalls

1. Not checking all conditions for Rice's theorem (especially non-triviality)
2. Confusing recursive with recursively enumerable
3. Not handling undefined cases in reductions
4. Forgetting to prove the reduction function is computable

Remember: When working with these concepts, always start by identifying which technique is most appropriate based on the problem's properties.

By systematically applying these approaches and carefully checking all conditions, you can effectively solve most computability theory problems.