1. Primitive Recursive Functions (PR)

Definition Check Template

Given function f, prove f ∈ PR using exclusively:

- **Base functions**: 0, s (successor), U^k_j (projections)
- **Composition**: $h(x\square) = g(f_1(x\square),\ldots,f_n(x\square))$
- **Primitive recursion**: $h(x\square,0) = f(x\square)$, $h(x\square,y+1) = g(x\square,y,h(x\square,y))$

Standard Building Blocks

1. **¬sg(x)**: ¬sg(0) = 1, ¬sg(y+1) = 0
2. **sg(x)**: sg(0) = 0, sg(y+1) = 1
3. **x∸y**: x∸0 = x, x∸(y+1) = (x∸y)∸1
4. **$rm_2(x)$**: $rm_2(0) = 0$, $rm_2(x+1) = \neg sg(rm_2(x))$
5. **eq(x,y)**: eq(x,y) = ¬sg(|x-y|)

Strategy by Function Type

- **Characteristic functions**: Use sg, ¬sg, rm_2 patterns
- **Arithmetic**: Build through bounded recursion
- **Conditional**: Use multiplication by characteristic functions
- **Bounded search**: μz≤x.P(z) = minimize with explicit bound

---

2. SMN Theorem Applications

Standard Construction Pattern

Goal: Construct s: N → N such that $W_{s(x)}$ and $E_{s(x)}$ have specified properties.

Template:

```
Step 1: Define g(x,y) = {
  target_expression  if condition(x,y)
  ↑                 otherwise
}

Step 2: Verify g is computable:
g(x,y) = [expression] + μz.[divergence_condition]
```

```
Step 3: Apply SMN: ∃s total computable: ϕₛ₍ₓ₎(y) = g(x,y)

Step 4: Verify:
- Wₛ₍ₓ₎ = {y | g(x,y)↓} = [desired domain]
- Eₛ₍ₓ₎ = {g(x,y) | y ∈ Wₛ₍ₓ₎} = [desired codomain]
```

Common Patterns

- **Finite sets**: Use bounded conditions `y < bound(x)`
- **Arithmetic progressions**: `g(x,y) = f(x) + h(y)`
- **Specific codomains**: Map systematically to target range

---

3. Function Computability Analysis

Diagonalization Template

For functions f: N → N of form:

```
f(x) = {
  expr₁(ϕₓ(x))  if ϕₓ(x)↓
  expr₂(x)      if ϕₓ(x)↑
}
```

Non-computability Test:

- **Find**: computable h such that $\chi_k(x) = h(f(x),x)$
- **Common patterns**:
  - $\chi_k(x) = sg(f(x) \dot{-} 2x)$
  - $\chi_k(x) = \neg sg(|f(x) - x^2|)$
  - $\chi_k(x) = \neg sg(|f(x) - (x+1)|)$

Totality Verification

- **By construction**: Show f(x) defined for all x
- **By cases**: Verify both branches produce values

---

4. Set Recursiveness Classification

Decision Tree Algorithm

Phase 1: Saturation Check

Set $A \subseteq N$ is **saturated** iff: $x \in A \wedge \varphi_x = \varphi_y \Longrightarrow y \in A$

Test: $A = \{x \mid \varphi_x \in \mathcal{A}\}$ for some property $\mathcal{A}$ of functions?

- **YES**: Apply Rice's Theorem or Rice-Shapiro
- **NO**: Proceed to direct analysis

Phase 2: Rice's Theorem Application

If A saturated:

- **$A \neq \emptyset$, N**: Then A, $\bar{A}$ not recursive
- **Further classification**: Use Rice-Shapiro for r.e. analysis

Phase 3: Rice-Shapiro Theorem

For saturated $A = \{x \mid \varphi_x \in \mathcal{A}\}$:

A not r.e. iff: $\exists f \notin \mathcal{A}$ such that $\forall \theta \subseteq f$ finite: $\theta \notin \mathcal{A}$ **$\bar{A}$ not r.e. iff**: $\exists f \in \mathcal{A}$ such that $\forall \theta \subseteq f$ finite: $\theta \in \mathcal{A}$

Standard witnesses:

- Use id, $\emptyset$, constants, finite functions as test cases
- Check subset relationships carefully

Phase 4: Direct Analysis (Non-saturated)

To prove non-recursive: Show $K \leq_m A$ or $\bar{K} \leq_m A$

Reduction Construction Template:

```
g(x,y) = {
  target_value  if x ∈ K
  ↑             otherwise
}

Verification:
- g computable: g(x,y) = target_value · scₖ(x)
- SMN gives s: N → N total computable
- x ∈ K ⇔ s(x) ∈ A (verify both directions)
```

To prove r.e.: Construct semi-characteristic function

```
scₐ(x) = 1(μw.P(x,w))
```

where P is decidable and captures membership condition.

Standard Reduction Targets

- **$K \leq_m A$**: For sets containing "positive" computational behavior
- **$\bar{K} \leq_m A$**: For sets containing "negative" computational behavior
- **Mixed reductions**: Use appropriate conditional functions

---

5. Second Recursion Theorem Applications

Non-saturation Proof Template

Goal: Prove set C not saturated

Standard Construction:

```
Step 1: Define g(x,y) with self-reference property

Step 2: Apply SMN: ∃s total computable: ϕₛ₍ₓ₎(y) = g(x,y)

Step 3: Apply 2nd Recursion Theorem: ∃e: ϕₑ = ϕₛ₍ₑ₎

Step 4: Show e ∈ C by construction

Step 5: Find e' ≠ e with ϕₑ = ϕₑ' but e' ∉ C
```

Common Self-reference Patterns

- **Identity**: g(x,y) = x
- **Quadratic**: g(x,y) = x^2
- **Conditional**: g(x,y) = {value if condition(x,y); ↑ otherwise}
- **Domain control**: g(x,y) = {f(y) if y ∈ specific_set(x); ↑ otherwise}

Fixed Point Construction

Use when proving existence of special indices:

```
Want: ϕ_e with property P(e)
Define: g(x,y) encoding property P
Get: e such that ϕ_e = ϕ_s(e) and P(e) holds
```

---

6. Decidability and Semi-decidability

Classification Strategy

Decidable: χ_p: N^k → N computable **Semi-decidable**: sc_p: N^k → N computable

Structure Theorem Applications

P(x□) semi-decidable □ □Q decidable: P(x□) ≡ □y.Q(x□,y)

Proof patterns:

- **Forward**: P semi-decidable □ P(x□) ≡ □t.H(e,x□,t) for index e
- **Backward**: P(x□) ≡ □y.Q(x□,y) □ sc_p(x□) = 1(μy.|χQ(x□,y) - 1|)

Counterexample Construction

To show implication fails:

- Use variants of halting problem
- Construct predicates involving K, K̄
- Standard pattern: P(x,y) = "x ∈ K̄ ∧ y = 0"

---

7. URM Machine Variants

Comparison Methodology

Step 1: Instruction Simulation

New → Standard URM:

- Show each new instruction encodable as URM subroutine
- Prove encoding preserves semantics
- Conclude C_new ⊆ C

Standard → New URM:

- Show each URM instruction encodable in variant
- Or prove impossibility using invariants

Step 2: Inclusion Analysis

Proper inclusion $C_1 \subsetneq C_2$:

- Prove $C_1 \subseteq C_2$ by simulation
- Find function in $C_2 \setminus C_1$ using invariant properties

Common Invariant Arguments

- **Bounded values**: Max register value bounded by initial configuration
- **Monotonicity**: Register values can only increase/decrease
- **Reachability**: Certain values impossible to generate

---

8. Reduction Theory

Formal Definition

$A \leq_m B$ iff $\exists f: N \to N$ total computable: $x \in A \Leftrightarrow f(x) \in B$

Standard Constructions

Type 1: Conditional Functions

```
g(x,y) = {
  target_function(y)  if x ∈ source_set
  ↑                   otherwise
}
```

Type 2: Domain Manipulations

```
g(x,y) = expression_creating_desired_domain_codomain_relationship
```

Type 3: Diagonal Constructions

```
g(x,y) = expression_ensuring_diagonal_property
```

Verification Checklist

1. **Computability**: g is computable (explicit construction)
2. **SMN application**: Obtain s total computable
3. **Reduction property**: $x \in A \iff s(x) \in B$
4. **Direction verification**: Prove both $\Longrightarrow$ and $\Longleftarrow$

---

9. Problem-Solving Workflow

Phase 1: Problem Classification

- Identify keywords: "prove $f \in PR$", "classify A", "show non-computable"
- Determine technique category from above

Phase 2: Strategy Selection

- **Saturated sets**: Rice/Rice-Shapiro pathway
- **Function computability**: Diagonalization
- **Existence proofs**: 2nd Recursion Theorem
- **Construction problems**: SMN Theorem

Phase 3: Formal Execution

- Apply template precisely
- Verify all conditions explicitly
- Check edge cases and special values

Phase 4: Verification

- Confirm all required properties
- Validate computational claims
- Ensure logical completeness

---

10. Common Pitfalls and Precision Points

Rice-Shapiro Applications

- **Critical**: Verify finite subfunction relationships exactly
- **Common error**: Confusing $\subseteq$ with proper subset
- **Check**: Both directions of equivalence in theorem statement

SMN Constructions

- **Ensure**: Target function actually computable
- **Verify**: Domain/codomain properties hold exactly as stated
- **Check**: Parameter dependencies correctly handled

Diagonalization Arguments

- **Verify**: Constructed function differs from ALL computable functions
- **Check**: Totality when claimed
- **Ensure**: Reduction to halting problem is valid

Reduction Proofs

- **Critical**: Both directions of equivalence
- **Verify**: Function totality and computability
- **Check**: SMN application gives correct index function

This rigorous framework provides systematic approaches for all major computability exercise categories, with precise mathematical templates and verification procedures.