Complete S-m-n Theorem Exercise Methodology: Universal Pattern Guide

Core S-m-n Theorem Statement

Formal: Given $m, n \geq 1$, $\exists s_{m,n}: \mathbb{N}^{m+1} \to \mathbb{N}$ total computable such that $\forall e \in \mathbb{N}$, $\vec{x} \in \mathbb{N}^m$, $\vec{y} \in \mathbb{N}^n$: $\varphi_e^{(m+n)}(\vec{x}, \vec{y}) = \varphi_{s_{m,n}(e,\vec{x})}^{(n)}(\vec{y})$

Intuitive: Allows partial application - "fixing" some arguments to create new functions.

UNIVERSAL MECHANICAL METHODOLOGY

Algorithm for ANY S-m-n Exercise:

1. **IDENTIFY THE TARGET** (What do you need to prove exists?)
   - Look for: "show there exists $s: \mathbb{N} \to \mathbb{N}$ such that..."
   - Extract requirements for $W_{s(x)}$ and $E_{s(x)}$
2. **CLASSIFY THE EXERCISE TYPE** (Use patterns below)
3. **CONSTRUCT g(x, y)** using the appropriate pattern
4. **APPLY S-m-n THEOREM** to get desired function
5. **VERIFY** both domain and codomain properties

COMPLETE EXERCISE TYPE CLASSIFICATION

TYPE A: CARDINALITY CONTROL

Pattern Recognition: $|W_{s(x)}| = f(x)$, $|E_{s(x)}| = g(x)$

Template:

```
g(x, y) = {
    h(x, y),  if y < f(x)
    ↑,        otherwise
}
```

Examples:

- $|W_{s(x)}| = 2x$, $|E_{s(x)}| = x \to g(x,y) = qt(2,y)$ if $y < 2x$, $\uparrow$ otherwise
- $|W_{s(x,y)}| = xy \to g(x,y,z) = 0$ if $z < xy$, $\uparrow$ otherwise

TYPE B: SPECIFIC SET CONSTRUCTION

Pattern Recognition: $W_{s(x)}$ = specific set, $E_{s(x)}$ = specific set

Sub-patterns:

B1: Even/Odd Domain

```
g(x, y) = {
    f(x, y),  if y ≡ 0 (mod 2) [for even]
    ↑,        otherwise
}
```

B2: Arithmetic Progressions

```
g(x, y) = {
    f(x, y),  if y ≥ x [for y ≥ x]
    ↑,        otherwise
}
```

B3: Power Sets

```
g(x, y) = {
    something,  if ∃k: y = k^x
    ↑,          otherwise
}
```

TYPE C: EQUALITY CONDITIONS

Pattern Recognition: W_s(x,y) = {z | condition(x,y,z)}

Template:

```
g(x, y, z) = {
    value,  if condition(x,y,z) holds
    ↑,      otherwise
}
```

Examples:

- W_s(x,y) = {z | x*z* = *y} → g(x,y,z) = 0 if* x*z* = y, ↑ otherwise
- W_s(x) = {divisors of x} → g(x,y) = x*y if y divides x, 1 otherwise

TYPE D: REDUCTION CONSTRUCTIONS (for r.e. proofs)

Pattern Recognition: Used to show K ≤_m A or similar

Template:

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

Examples:

- g(x,y) = 1 if x ∈ K, ↑ otherwise
- g(x,y) = y+1 if x ∈ K, ↑ otherwise

TYPE E: COMPLEX FUNCTIONAL CONDITIONS

Pattern Recognition: Involves φ_x(x) or other computability conditions

Template:

```
g(x, y) = {
    expression,  if ϕ_x(something) ↓
    ↑,           otherwise
}
```

MECHANICAL CONSTRUCTION PATTERNS

Pattern 1: Bounded Range Construction

```
g(x, y) = {
    f(y),  if y < bound(x)
    ↑,     otherwise
}
= f(y) + μz.(y + 1 - bound(x))
```

Pattern 2: Modular Arithmetic

```
g(x, y) = {
    expression,  if y ≡ r (mod m)
    ↑,           otherwise
}
= expression + μz.rm(m, y)  [for r = 0]
```

Pattern 3: Divisibility

```
g(x, y) = {
    expression,  if y divides x
    alternative, otherwise
}
= expression * sg(rm(y, x)) + alternative * sg(rm(y, x))
```

Pattern 4: Conditional with K

```
g(x, y) = {
    value,  if x ∈ K
    ↑,      otherwise
}
= value * sc_K(x)
```

COMPLETE SOLUTION LIBRARY

Library Entry 1: Basic Cardinality

Problem: |W_s(x)| = 2x, |E_s(x)| = x

```
g(x, y) = qt(2, y) + μz.(y + 1 - 2x)
```

Library Entry 2: Even Domain, Offset Codomain

Problem: W_k(n) = P (evens), E_k(n) = {y | y ≥ n}

```
f(n, x) = x/2 + n + μz.rm(2, x)
```

Library Entry 3: Divisor Construction

Problem: E_k(n) = divisors of n

```
g(n, x) = (x * n) * sg(rm(x, n)) + sg(rm(x, n))
```

Library Entry 4: Equation Solving

Problem: W_s(x,y) = {z | x*z = y}

```
f(x, y, z) = μw.(x*z - y) + (y - x*z)
```

Library Entry 5: Square Construction

Problem: W_s(x) = $\{(k+x)^2 \mid k \in \mathbb{N}\}$

```
g(x, y) = μk.|(x + k)² - y|
```

Library Entry 6: Power Construction

Problem: W_k(n) = $\{z^n \mid z \in \mathbb{N}\}$

```
f(n, x) = qt(x, z) + μz.rm(x, z)  [where z represents the base]
```

Library Entry 7: Intersection Cardinality

Problem: $|W_x \cap E_x| = 1$

```
g(x, y) = y if y ∈ both_domain_and_codomain, ↑ otherwise
```

Library Entry 8: Union Construction

Problem: $x \in W_x \cup E_x$

```
g(x, y) = 1 if x ∈ K, ↑ otherwise (for reductions)
```

Library Entry 9: Complement Conditions

Problem: $W_x \subseteq P$ (subset of evens)

```
g(x, y) = 1 if x ∈ K, ↑ otherwise (creates W = ℕ when x ∈ K)
```

Library Entry 10: Functional Value Conditions

Problem: $\varphi_x(x) = x^2$ when $x \in W_x$

```
g(x, y) = y² if x ∈ K, ↑ otherwise
```

STEP-BY-STEP APPLICATION GUIDE

For ANY exercise, follow this algorithm:

1. **Read the exercise and extract:**
   - What needs to be proven? (∃s such that...)
   - What are the constraints on W_s(x)?
   - What are the constraints on E_s(x)?
2. **Pattern match to exercise type:**
   - Type A: Cardinality control
   - Type B: Specific sets
   - Type C: Equality conditions
   - Type D: Reduction construction
   - Type E: Complex functional
3. **Select appropriate construction pattern:**
   - Bounded range for cardinality
   - Modular arithmetic for even/odd
   - Divisibility for factor conditions
   - Conditional with K for reductions
4. **Construct g(x, y):**
   - Use library entry if exact match
   - Adapt template for similar patterns
   - Ensure computability using primitive recursive functions
5. **Verify correctness:**
   - W_s(x) = {y | g(x, y) ↓} matches requirement
   - E_s(x) = {g(x, y) | y ∈ W_s(x)} matches requirement
6. **Write formal proof:**
   - State S-m-n theorem
   - Define g(x, y) explicitly
   - Show g is computable
   - Apply S-m-n to get s
   - Verify both properties

ADVANCED PATTERNS FOR COMPLEX EXERCISES

Pattern: Multiple Parameter Functions

For s: $\mathbb{N}^2 \rightarrow \mathbb{N}$ with W_s(x,y) = something:

```
g(x, y, z) = construction_based_on_all_three_variables
```

Pattern: Totality Requirements

When φ_k(n) must be total:

```
g(n, x) = always_defined_expression
```

Pattern: Infinite Set Construction

For infinite sets like "all powers of x":

```
g(x, y) = μk.condition_for_y_to_be_power_of_x
```

Pattern: Negation Conditions

For "y ∉ W_x" type conditions:

```
g(x, y) = 0 if ¬H(x, y, z) for all z, ↑ otherwise
```

MECHANICAL VERIFICATION CHECKLIST

✓ **g(x, y) is computable** (uses only primitive recursive functions) ✓ **Domain correct:** W_s(x) = {y | g(x, y) ↓} ✓ **Codomain correct:** E_s(x) = {g(x, y) | y ∈ W_s(x)} ✓ **S-m-n application valid:** ∃s such that φ_s(x)(y) = g(x, y) ✓ **All edge cases handled**

EMERGENCY PATTERNS (When Nothing Else Works)

Pattern: Direct Encoding

```
g(x, y) = μz.(complex_condition_encoding_the_requirement)
```

Pattern: Case Analysis

```
g(x, y) = {
    case1_result,  if condition1
    case2_result,  if condition2
    ↑,             otherwise
}
```

Pattern: Composition Construction

```
g(x, y) = h(f₁(x, y), f₂(x, y), ...)
```

This methodology guarantees you can solve ANY S-m-n theorem exercise by:

1. Pattern recognition
2. Template application
3. Mechanical verification

The key insight: Every S-m-n exercise reduces to constructing the right g(x, y) function that encodes the desired domain/codomain relationship.