Complete Primitive Recursive Functions Guide: Every Exercise Type

Definition of Primitive Recursive Functions (PR)

Formal Definition: The class PR of primitive recursive functions is the **smallest** class of functions containing:

Base Functions:

1. **Zero function:** 0: ℕ → ℕ defined by 0(x) = 0 for all x ∈ ℕ
2. **Successor function:** s: ℕ → ℕ defined by s(x) = x + 1 for all x ∈ ℕ
3. **Projection functions:** U_j^k: ℕ^k → ℕ defined by U_j^k(x_1,...,x_k) = x_j

Operations (Closed Under):

1. **Generalized Composition:** If f_1,...,f_n: ℕ^k → ℕ and g: ℕ^n → ℕ, then h(x⃗) = g(f_1(x⃗),...,f_n(x⃗))
2. **Primitive Recursion:** Given f: ℕ^k → ℕ and g: ℕ^(k+2) → ℕ, define h: ℕ^(k+1) → ℕ by:

```
h(x⃗, 0) = f(x⃗)
h(x⃗, y+1) = g(x⃗, y, h(x⃗, y))
```

Universal Exercise Methodology

Step 1: Identify Exercise Type

Look for these patterns in the exercise text:

- **"prove that function ... is primitive recursive"** → Construction exercise
- **"characteristic function of set ..."** → Boolean/set exercise
- **"using only the definition"** → Must build from base functions
- **Power/exponential functions** → Repeated operations
- **Arithmetic operations** → Basic building blocks

Step 2: Choose Construction Strategy

- **Direct primitive recursion** for recursive patterns
- **Composition** for combining known PR functions
- **Helper functions** for complex constructions

Step 3: Write Primitive Recursive Definition

- Base case: h(..., 0) = ...
- Recursive case: h(..., y+1) = ...
- Ensure you only use known PR functions

Step 4: Verify Primitive Recursion

- Check base case uses only previous PR functions
- Check recursive case uses $g(\vec{x}, y, h(\vec{x}, y))$ pattern
- Verify all helper functions are also PR

Complete Exercise Type Classification

TYPE 1: BASIC ARITHMETIC FUNCTIONS

Pattern: Addition, Multiplication, Powers

Exercise: "Prove that sum(x,y) = x + y is primitive recursive"

Step-by-Step Solution:

```
sum(x, 0) = x                    [base case: f(x) = U₁¹(x) = x]
sum(x, y+1) = sum(x, y) + 1      [recursive: g(x,y,z) = s(z)]
```

Why this works:

- Base case: Adding 0 to x gives x (identity)
- Recursive case: To add y+1, we add y then add 1 more
- Uses only successor function and previous result

Template for similar functions:

```
h(x, 0) = initial_value(x)
h(x, y+1) = operation(x, y, h(x,y))
```

Example: Multiplication

```
mult(x, 0) = 0                   [base: f(x) = 0]
mult(x, y+1) = mult(x, y) + x    [recursive: g(x,y,z) = sum(z, x)]
```

Example: Power Function (2^y)

```
pow2(0) = 1                      [base: f() = 1]
pow2(y+1) = double(pow2(y))      [recursive: g(y,z) = double(z)]
```

where double(x) is primitive recursive:

```
double(0) = 0
double(y+1) = double(y) + 2
```

TYPE 2: CHARACTERISTIC FUNCTIONS

Pattern: χ_A(x) = 1 if x ∈ A, 0 otherwise

Exercise: "Prove that χ_P (characteristic of even numbers) is primitive recursive"

Step-by-Step Solution:

```
χ_P(0) = 1                        [0 is even]
χ_P(y+1) = sḡ(χ_P(y))           [flip the previous result]
```

Helper function sḡ (negated sign):

```
sḡ(0) = 1
sḡ(y+1) = 0
```

Why this works:

- Start with 0 being even (χ_P(0) = 1)
- For each successor, flip the result using sḡ
- Pattern: even → odd → even → odd...

Example: Characteristic of {2^n - 1 | n ∈ ℕ}

Strategy: First define a(n) = 2^n - 1, then check membership

```
a(0) = 0                          [2⁰ - 1 = 0]
a(n+1) = 2·a(n) + 1             [2^(n+1) - 1 = 2(2^n - 1) + 1]

chk(x, 0) = sḡ(x)               [check if x = a(0) = 0]
chk(x, m+1) = chk(x, m) + eq(x, a(m+1))

χ_A(x) = chk(x, x)              [check membership up to x]
```

TYPE 3: TRUNCATED SUBTRACTION AND ABSOLUTE DIFFERENCE

Pattern: x ∸ y = max(0, x-y)

Exercise: "Prove that $p_2(y) = |y - 2|$ is primitive recursive"

Step-by-Step Construction: First build $p_1(y) = |y - 1| = y ∸ 1$:

```
p₁(0) = 1                       [|0 - 1| = 1]
p₁(y+1) = y                     [|y+1 - 1| = y]
```

Then build $p_2(y) = |y - 2|$:

```
p₂(0) = 2                        [|0 - 2| = 2]
p₂(y+1) = p₁(y)                  [|y+1 - 2| = |y - 1|]
```

General Pattern for |x - k|:

```
pₖ(0) = k
pₖ(y+1) = pₖ₋₁(y)
```

TYPE 4: DIVISION AND REMAINDER FUNCTIONS

Pattern: Half function (integer division)

Exercise: "Prove that half(x) = $\lfloor x/2 \rfloor$ is primitive recursive"

Step-by-Step Solution: First define remainder mod 2:

```
rm₂(0) = 0
rm₂(x+1) = sḡ(rm₂(x))          [alternates 0,1,0,1,...]
```

Then define half:

```
half(0) = 0
half(x+1) = half(x) + rm₂(x)    [add 1 every two steps]
```

Why this works:

- $rm_2(x)$ gives the remainder when x is divided by 2
- We add 1 to half(x) only when x is even ($rm_2(x) = 0$)
- This gives us $\lfloor x/2 \rfloor$

TYPE 5: SUMMATION AND PRODUCT FUNCTIONS

Pattern: Σ and Π operations

Exercise: "Prove that t(x) = Σ(y=0 to x) y = 0+1+2+...+x is primitive recursive"

Step-by-Step Solution:

```
t(0) = 0                         [sum from 0 to 0 is 0]
t(y+1) = t(y) + (y+1)           [add next term]
```

This uses the sum function which we know is primitive recursive.

General Pattern for Σ(i=0 to x) f(i):

```
sum_f(0) = f(0)
sum_f(x+1) = sum_f(x) + f(x+1)
```

Example: Sum of k arguments

Exercise: "Prove that $sum\_k(x_1,...,x_k) = \Sigma x_i$ is primitive recursive"

Recursive construction:

```
sum₂(x₁, x₂) = x₁ + x₂           [base case for k=2]
sum_{k+1}(x₁,...,xₖ₊₁) = sum_k(x₁,...,xₖ) + xₖ₊₁
```

TYPE 6: BOUNDED SEARCH AND COUNTING

Pattern: Count elements satisfying a property

Exercise: "Prove that $cpr(x,y) = |\{p \mid x \leq p < y \wedge p \text{ prime}\}|$ is primitive recursive"

Step-by-Step Solution: Use helper function $cpr'(x,k) = |\{p \mid x \leq p < x+k \wedge p \text{ prime}\}|$:

```
cpr'(x, 0) = 0                    [no primes in empty interval]
cpr'(x, k+1) = cpr'(x, k) + χ_Prime(x+k)
```

Then: $cpr(x,y) = cpr'(x, y \dot{-} x)$

General Pattern for counting:

```
count(property, start, 0) = 0
count(property, start, k+1) = count(property, start, k) + property(start+k)
```

TYPE 7: COMPLEX RECURSIVE PATTERNS

Pattern: Nested recursions and multiple base cases

Exercise: "Prove that the largest prime ≤ x is primitive recursive"

Step-by-Step Construction:

```
count_primes(x) = Σ(i=1 to x) χ_Prime(i)  [count primes up to x]
largest_prime(x) = p_{count_primes(x)}      [use counting to index]
```

Complex helper construction:

```
ip(x) = μy≤x. sg(div(p_{x-y}, x))          [bounded search]
lp(x) = p_{x-ip(x)} · sg(x-1) + sḡ(x-1)    [handle edge cases]
```

PRACTICAL CONSTRUCTION STRATEGIES

Strategy 1: Build Helper Functions First

Pattern: Complex function requiring multiple steps

1. Identify needed operations (sg, $s\bar{g}$, rm, div, etc.)
2. Build these as primitive recursive functions
3. Combine using composition and recursion

Strategy 2: Use Characteristic Functions

Pattern: Functions involving set membership

1. Define characteristic function of relevant set
2. Use Boolean operations (sg, $s\bar{g}$, multiplication)
3. Combine with arithmetic operations

Strategy 3: Bounded Operations

Pattern: "For all $x \leq$ bound" or "There exists $x \leq$ bound"

1. Use bounded sum: $\Sigma(i=0 \text{ to } n)\ f(i)$
2. Use bounded product: $\Pi(i=0 \text{ to } n)\ f(i)$
3. Use bounded minimization: $\mu y \leq x.\ P(y)$

Strategy 4: Case Analysis

Pattern: Functions defined differently for different ranges

1. Use sg and $s\bar{g}$ to create "switches"
2. Multiply each case by its condition
3. Add all cases together

Template:

```
f(x) = case1(x)·condition1(x) + case2(x)·condition2(x) + ...
```

VERIFICATION CHECKLIST

For every PR construction, verify:

✓ **Base case uses only:** Zero, successor, projections, or previously proven PR functions ✓ **Recursive case follows pattern:** $h(\vec{x}, y+1) = g(\vec{x}, y, h(\vec{x}, y))$ ✓ **All helper functions are PR:** Each must be built from base functions ✓ **Composition is valid:** Output types match input types ✓ **No unbounded operations:** No μy without bound

COMMON MISTAKE PATTERNS

Mistake 1: Using Non-PR Functions

Wrong: "Since division is obvious..." **Correct:** Build division from rm and bounded search

Mistake 2: Unbounded Operations

Wrong: μy. P(y) (no bound) **Correct:** μy≤x. P(y) (bounded minimization)

Mistake 3: Wrong Base Cases

Wrong: Forgetting to handle f(0) properly **Correct:** Always specify base case explicitly

Mistake 4: Non-Primitive Recursion Pattern

Wrong: h(x, y+1) = h(x+1, y-1) (complex recursion) **Correct:** h(x, y+1) = g(x, y, h(x, y)) (simple pattern)

COMPLETE REAL EXAMPLES FROM YOUR MATERIALS

OFFICIAL EXERCISE 2.1: Power of 2 Function

Exact Exercise Text: "Prove that the function pow2 : ℕ → ℕ, defined by pow2(y) = 2^y, is primitive recursive."

Complete Step-by-Step Solution:

Step 1: Identify the pattern

- pow2(0) = 2^0 = 1
- pow2(1) = 2^1 = 2
- pow2(2) = 2^2 = 4
- Pattern: each step doubles the previous result

Step 2: Set up primitive recursion

```
pow2(0) = 1                    [base case]
pow2(y+1) = 2 * pow2(y)        [recursive case]
```

Step 3: Make it formal using helper function We need double(x) = 2*x to be primitive recursive:

```
double(0) = 0                  [base case]
double(y+1) = double(y) + 2    [recursive: add 2 each time]
```

Step 4: Complete definition

```
pow2(0) = 1                    [f() = 1]
pow2(y+1) = double(pow2(y))    [g(y,z) = double(z)]
```

Step 5: Verify it's primitive recursive

- Base case uses constant 1 (composition of successor and zero)
- Recursive case uses double (which we proved is PR) and previous result
- Follows h(y+1) = g(y, h(y)) pattern ✓

OFFICIAL EXERCISE 2.2: Characteristic Function of {2^n - 1}

Exact Exercise Text: "Prove that the characteristic function χ_A of the set A = {2^n - 1 : n ∈ ℕ} is primitive recursive."

Complete Step-by-Step Solution:

Step 1: Understand the set

- A = {0, 1, 3, 7, 15, 31, ...} = $\{2^0-1, 2^1-1, 2^2-1, 2^3-1, ...\}$

Step 2: Build helper function a(n) = 2^n - 1

```
a(0) = 0                          [2⁰ - 1 = 0]
a(n+1) = 2·a(n) + 1             [2^(n+1) - 1 = 2(2^n - 1) + 1]
```

Step 3: Build checking function We need chk(x, m) = 1 if ∃n≤m such that x = a(n), 0 otherwise:

```
chk(x, 0) = sḡ(x)               [check if x = a(0) = 0]
chk(x, m+1) = chk(x, m) + eq(x, a(m+1))  [accumulate matches]
```

Step 4: Final characteristic function

```
χ_A(x) = chk(x, x)              [check membership up to x]
```

Step 5: Verify all helpers are PR

- a(n) uses multiplication and addition (both PR)
- eq(x,y) = sḡ(|x-y|) uses truncated subtraction and sḡ
- chk uses addition and eq
- All compose correctly ✓

OFFICIAL EXERCISE 2.3: Even Numbers Characteristic Function

Exact Exercise Text: "Prove that χ_P, the characteristic function of the set of even numbers P is primitive recursive."

Complete Step-by-Step Solution:

Step 1: Understand the pattern

- χ_P(0) = 1 (0 is even)
- χ_P(1) = 0 (1 is odd)
- χ_P(2) = 1 (2 is even)
- Pattern: alternates 1, 0, 1, 0, ...

Step 2: Use negated sign to flip First define sḡ (negated sign function):

```
sḡ(0) = 1                      [base case]
sḡ(y+1) = 0                    [recursive case]
```

Step 3: Build χ_P using alternation

```
χ_P(0) = 1                      [0 is even]
χ_P(y+1) = sḡ(χ_P(y))          [flip previous result]
```

Step 4: Verify the pattern

- χ_P(0) = 1 ✓
- χ_P(1) = sḡ(χ_P(0)) = sḡ(1) = 0 ✓
- χ_P(2) = sḡ(χ_P(1)) = sḡ(0) = 1 ✓
- Pattern continues correctly

Why this works: Each successor flips even/odd status, and sḡ flips 1↔0.

OFFICIAL EXERCISE 2.4: Half Function (Integer Division)

Exact Exercise Text: "Prove the function half : ℕ → ℕ, defined by half(x) = ⌊x/2⌋, is primitive recursive."

Complete Step-by-Step Solution:

Step 1: Build remainder function rm₂

```
rm₂(0) = 0                      [0 mod 2 = 0]
rm₂(x+1) = sḡ(rm₂(x))          [alternates 0,1,0,1,...]
```

Step 2: Build half function

```
half(0) = 0                     [⌊0/2⌋ = 0]
half(x+1) = half(x) + rm₂(x)    [add 1 every two steps]
```

Step 3: Verify the logic

- When x is even: $rm_2(x) = 0$, so half(x+1) = half(x) + 0 = half(x)
- When x is odd: $rm_2(x) = 1$, so half(x+1) = half(x) + 1
- This gives us: half(0)=0, half(1)=0, half(2)=1, half(3)=1, half(4)=2, ...
- Pattern: $\lfloor 0/2 \rfloor=0$, $\lfloor 1/2 \rfloor=0$, $\lfloor 2/2 \rfloor=1$, $\lfloor 3/2 \rfloor=1$, $\lfloor 4/2 \rfloor=2$ ✓

Step 4: Verify primitive recursion

- Uses $\bar{sg}$ (which we built) and addition (basic PR function)
- Follows proper recursive pattern ✓

OFFICIAL EXERCISE: Sum Function t(x) = 0+1+2+...+x

Exact Exercise Text: "Show that the function $t : \mathbb{N} \to \mathbb{N}$ defined by $t(x) = \Sigma(y=0 \text{ to } x)\ y = 0 + 1 + 2 + \ldots + x$ is in PR."

Complete Step-by-Step Solution:

Step 1: Set up primitive recursion

```
t(0) = 0                          [sum from 0 to 0 is 0]
t(y+1) = t(y) + (y+1)            [add next term]
```

Step 2: Use known PR function sum Since sum(x,y) = x + y is primitive recursive:

```
t(0) = 0                          [f() = 0]
t(y+1) = sum(t(y), y+1)          [g(y,z) = sum(z, s(y))]
```

Step 3: Verify correctness

- t(0) = 0 ✓
- t(1) = t(0) + 1 = 0 + 1 = 1 ✓
- t(2) = t(1) + 2 = 1 + 2 = 3 ✓
- t(3) = t(2) + 3 = 3 + 3 = 6 ✓
- Formula: t(n) = n(n+1)/2 ✓

PRACTICAL TUTORING EXAMPLE: Counting Primes

Exercise: "Prove that $cpr(x,y) = |\{p \mid x \le p < y \wedge p \text{ prime}\}|$ is primitive recursive"

Step-by-Step Construction:

Step 1: Use helper function cpr'(x,k)

```
cpr'(x, 0) = 0                     [no primes in empty interval]
cpr'(x, k+1) = cpr'(x, k) + χ_Prime(x+k)  [add 1 if x+k is prime]
```

Step 2: Relate to original function

```
cpr(x, y) = cpr'(x, y ∸ x)     [count primes in [x, x+(y-x)) = [x, y)]
```

Step 3: Verify with example

- cpr(10, 15) counts primes in [10, 15) = {11, 13}
- cpr'(10, 5) counts primes in {10, 11, 12, 13, 14}
- χ_Prime(10)=0, χ_Prime(11)=1, χ_Prime(12)=0, χ_Prime(13)=1, χ_Prime(14)=0
- Sum = 0+1+0+1+0 = 2 ✓

Key insight: Use bounded counting with characteristic functions.

COMPLETE SOLUTION LIBRARY

Basic Building Blocks (Always Available):

```
0(x) = 0                           [zero function]
s(x) = x + 1                       [successor]
U_j^k(x₁,...,xₖ) = xⱼ              [projections]

sḡ(0) = 1, sḡ(x+1) = 0           [negated sign]
sum(x,0) = x, sum(x,y+1) = sum(x,y)+1
mult(x,0) = 0, mult(x,y+1) = mult(x,y)+x
```

Advanced Functions:

```
eq(x,y) = sḡ(|x-y|)             [equality test]
leq(x,y) = sḡ(x ∸ y)            [less-or-equal test]
max(x,y) = x + (y ∸ x)           [maximum]
min(x,y) = x ∸ (x ∸ y)           [minimum]
```

Set Operations:

```
χ_A∪B(x) = sg(χ_A(x) + χ_B(x))  [union]
χ_A∩B(x) = χ_A(x) · χ_B(x)      [intersection]
χ_Ā(x) = sḡ(χ_A(x))            [complement]
```

This methodology covers EVERY type of primitive recursive exercise by providing:

1. **Pattern recognition** for exercise types
2. **Step-by-step construction** strategies
3. **Complete verification** methods
4. **Common mistake** avoidance
5. **Reusable building blocks** for complex constructions

You can now approach any PR exercise with confidence using these mechanical techniques!