1. Foundation: H and S Predicates

Core Definitions

H Predicate (Halting)

```
H(e,x,t) = "Program Pe(x) halts in t or fewer steps"
```

- **Decidable** - can always determine true/false in finite time
- **Used for**: Domain concerns (does program halt?)
- **Characteristic function**: $\chi H(e,x,t)$

S Predicate (Step)

```
S(e,x,y,t) = "Program Pe(x) halts with output y in t or fewer steps"
```

- **Decidable** - can always determine true/false in finite time
- **Used for**: Domain + codomain concerns (does program halt with specific output?)
- **Characteristic function**: $\chi S(e,x,y,t)$

Function Selection Logic

```
Use H when: Only care about halting (domain properties)
Use S when: Care about halting AND specific output (domain + codomain)
Use H ∧ S: Complex relationships requiring both
```

2. Minimalization Structure

General Template

```
sc_A(x) = 1(μw. [condition using H/S functions])
```

Step-by-Step Construction

1. **Identify the property**: What makes $x \in A$?
2. **Determine variables needed**: inputs, outputs, steps, multiplicative factors
3. **Choose H or S**: Based on whether you need specific outputs
4. **Bundle variables into w**: $w = \langle var_1, var_2, var_3, ...\rangle$
5. **Replace variables with projections**: $var_1 \to (w)_1$, $var_2 \to (w)_2$, etc.

3. Tuple Encoding Patterns

W-Encoding (95% of cases)

```
w = ⟨var₁, var₂, var₃, ...⟩
Access: (w)₁, (w)₂, (w)₃, ...
```

Important:

- Each variable appears only once in w
- Don't include x (function parameter) in w
- Not injective, but we only care about finite description

π-Encoding (when specified)

```
Only use when exercise explicitly provides π(x,y)
Example: π instead of w for pair encoding
```

Variable Bundling Rules

```
✓ Correct: w = ⟨y, t, k⟩ where y, t, k are distinct
✗ Wrong: w = ⟨y, t, y⟩ (duplicate variable)
✗ Wrong: w = ⟨x, y, t⟩ (x is function parameter)
```

4. Exercise Type 3 Patterns (Standard)

Template Structure

```
sc_A(x) = 1(μw. H(e, x, (w)₁) ∧ S(e, x, y, (w)₂))
```

Example 1: Domain-Codomain Intersection

Set: A = {x | Ex ∩ B ≠ ∅} where B finite

Construction:

```
Property: ∃y ∈ Ex such that y ∈ B
Variables needed: y (output), t (steps)
H/S choice: S (need specific output y)
w encoding: w = ⟨y, t⟩

sc_A(x) = 1(μw. S(x, (w)₁, (w)₂, (w)₃) ∧ (y ∈ B))
```

```
Since B finite: (y ∈ B) = (y = b₁) ∨ (y = b₂) ∨ ... ∨ (y = bₙ)
```

Example 2: Basic Membership

Set: A = {x | x ∈ Wx}

Construction:

```
Property: Program x halts on input x
Variables needed: t (steps)
H/S choice: H (only need halting)
w encoding: w = ⟨t⟩

sc_A(x) = 1(μw. H(x, x, (w)₁))
```

5. Exercise Type 4 Patterns (Complex)

Template Structure

```
sc_A(x) = 1(μw. H(x, (w)₁, (w)₃) ∧ S(x, (w)₁, (w)₂, (w)₃) ∧ [constraints])
```

Example 1: Multiplicative Relations

Set: B = {x | ∃k ∈ ℕ. k·x ∈ Wx}

Construction:

```
Property: ∃k such that program x halts on input k·x
Variables needed: k (multiplier), t (steps)
H/S choice: H (only need halting)
w encoding: w = ⟨k, t⟩

sc_B(x) = 1(μw. H(x, (w)₁ · x, (w)₂))
```

Example 2: Universal Quantification

Set: B = {x | ∀k ∈ ℕ. k+x ∈ Wx}

Construction:

```
Property: For all k, program x halts on input k+x
This is complement of ∃k such that ¬H(x, k+x, t)
```

```
Use reduction: K̄ ≤m B

Reduction function:
g(x,y) = {0 if ¬H(x,x,y)
         {↑ otherwise

sc_B(x) shows B is not r.e.
```

Example 3: Self-Reference with Output

Set: A = {x | x ∈ Wx ∩ Ex}

Construction:

```
Property: x ∈ Wx AND x ∈ Ex
Variables needed: z (input for Ex), t₁, t₂ (steps)
H/S choice: Both H and S
w encoding: w = ⟨z, t₁, t₂⟩

sc_A(x) = 1(μw. H(x, x, (w)₂) ∧ S(x, (w)₁, x, (w)₃))
```

6. Converting Predicates to Functions: The Mechanical Process

Core Transformation Rules

Step 1: Replace predicates with characteristic functions

```
H(e,x,t) → χH(e,x,t)     (1 if program halts, 0 otherwise)
S(e,x,y,t) → χS(e,x,y,t) (1 if program halts with output y, 0 otherwise)
```

Step 2: Handle equality and comparisons

```
a = b → sḡ(|a - b|)      (0 if equal, 1 if different)
a ≠ b → sg(|a - b|)      (1 if different, 0 if equal)
a > b → sg(a ∸ b)        (1 if a > b, 0 otherwise)
a ≥ b → sg(a + 1 ∸ b)    (1 if a ≥ b, 0 otherwise)
a < b → sg(b ∸ a)        (1 if a < b, 0 otherwise)
```

Note: ∸ denotes cut-off subtraction (a ∸ b = max(0, a-b))

Step 3: Transform logical operations

```
A ∧ B → χA + χB          (AND becomes addition)
A ∨ B → χA · χB          (OR becomes multiplication)
¬A → sḡ(χA)             (NOT becomes negated sign)
```

Step 4: Goal for minimization

- We want the expression to equal **0 when the condition is TRUE**
- We want the expression to equal **1 when the condition is FALSE**
- This makes μw stop when we find what we're looking for

Complete Transformation Example

Example: A = {x | Wx ∩ Ex ≠ ∅}

Step 1: Write in predicate form

```
x ∈ A ⟺ ∃y ∃t₁ ∃t₂ (H(x, y, t₁) ∧ S(x, z, y, t₂))
```

Meaning: ∃y that's in both domain and codomain

Step 2: Identify variables for w-encoding

```
Variables: y, z, t₁, t₂
w = ⟨y, z, t₁, t₂⟩
```

Step 3: Replace variables with projections

```
H(x, (w)₁, (w)₃) ∧ S(x, (w)₂, (w)₁, (w)₄)
```

Step 4: Convert to characteristic functions

```
χH(x, (w)₁, (w)₃) ∧ χS(x, (w)₂, (w)₁, (w)₄)
```

Step 5: Apply logical transformation (∧ → +)

```
χH(x, (w)₁, (w)₃) + χS(x, (w)₂, (w)₁, (w)₄)
```

Step 6: Apply negated sign (want 0 when true)

```
sḡ(χH(x, (w)₁, (w)₃) + χS(x, (w)₂, (w)₁, (w)₄))
```

Step 7: Complete semicharacteristic function

```
sc_A(x) = 1(μw. sḡ(χH(x, (w)₁, (w)₃) + χS(x, (w)₂, (w)₁, (w)₄)))
```

More Complex Example: Finite Set Intersection

Example: $A = \{x \mid Ex \cap B \neq \emptyset\}$ where $B = \{b_1, b_2, b_3\}$ is finite

Step 1: Write condition in predicate form

```
x ∈ A ⇔ ∃y ∃z ∃t (S(x, z, y, t) ∧ (y ∈ B))
```

Step 2: Handle finite set membership

```
y ∈ B ⇔ (y = b₁) ∨ (y = b₂) ∨ (y = b₃)
```

Step 3: Convert equalities

```
(y = b₁) → sḡ(|y − b₁|)
(y = b₂) → sḡ(|y − b₂|)
(y = b₃) → sḡ(|y − b₃|)
```

Step 4: Convert OR operation ($\vee \to \cdot$)

```
(y ∈ B) → sḡ(|y − b₁|) · sḡ(|y − b₂|) · sḡ(|y − b₃|)
```

Step 5: Complete condition with AND ($\wedge \to +$)

```
χS(x, z, y, t) + sḡ(|y − b₁|) · sḡ(|y − b₂|) · sḡ(|y − b₃|)
```

Step 6: Apply negated sign and minimization

```
sc_A(x) = 1(μw. sḡ(χS(x, (w)₂, (w)₁, (w)₃) + sḡ(|(w)₁ − b₁|) · sḡ(|(w)₁ − b₂|) · sḡ(|(w)₁ − b₃|)))
```

Advanced Example: Multiple Conditions

Example: $B = \{x \mid \exists k \in \mathbb{N}.\ k \cdot x \in Wx\}$

Step 1: Predicate form

```
x ∈ B ⇔ ∃k ∃t (H(x, k·x, t))
```

Step 2: Variable identification

```
Variables: k, t
w = ⟨k, t⟩
```

Step 3: Direct transformation

```
sc_B(x) = 1(μw. sg̅(χH(x, (w)₁ · x, (w)₂)))
```

Universal Quantification Example

Example: Set with ∀k condition (typically not r.e.)

Original: C = {x | ∀k ∈ ℕ. k+x ∈ Wx}

This is equivalent to: ¬∃k such that ¬H(x, k+x, t)

For reduction K̄ ≤m C:

```
g(x,y) = {0    if ¬H(x,x,y)
         {↑    otherwise
```

Transformation:

```
g(x,y) = μz. χH(x,x,y)
```

SMN application: ∃s total computable s.t. φs(x)(y) = g(x,y)

9. Step-by-Step Transformation Walkthrough

Complete Example: A = {x | x ∈ Wx ∪ Ex}

Step 1: Parse the condition

```
x ∈ Wx ∪ Ex means: x ∈ Wx OR x ∈ Ex
```

Step 2: Express as predicates

```
x ∈ Wx: ∃t₁ H(x, x, t₁)
x ∈ Ex: ∃z ∃t₂ S(x, z, x, t₂)
```

Step 3: Combine with OR

```
(∃t₁ H(x, x, t₁)) ∨ (∃z ∃t₂ S(x, z, x, t₂))
```

Step 4: Identify all variables

```
Variables: t₁, z, t₂
w = ⟨t₁, z, t₂⟩
```

Step 5: Replace with projections

```
H(x, x, (w)₁) ∨ S(x, (w)₂, x, (w)₃)
```

Step 6: Convert to characteristic functions

```
χH(x, x, (w)₁) ∨ χS(x, (w)₂, x, (w)₃)
```

Step 7: Apply OR transformation (∨ → ·)

```
χH(x, x, (w)₁) · χS(x, (w)₂, x, (w)₃)
```

Step 8: Apply negated sign (want 0 when true)

```
sḡ(χH(x, x, (w)₁) · χS(x, (w)₂, x, (w)₃))
```

Step 9: Complete function

```
sc_A(x) = 1(μw. sḡ(χH(x, x, (w)₁) · χS(x, (w)₂, x, (w)₃)))
```

Verification of Logic

Check OR operation (A ∨ B → A·B with negation):

- If A true, B false: $\chi A = 1, \chi B = 0 \rightarrow \chi A \cdot \chi B = 0 \rightarrow \overline{sg}(0) = 1$ ❌
- If A false, B true: $\chi A = 0, \chi B = 1 \rightarrow \chi A \cdot \chi B = 0 \rightarrow \overline{sg}(0) = 1$ ❌
- If both false: $\chi A = 0, \chi B = 0 \rightarrow \chi A \cdot \chi B = 0 \rightarrow \overline{sg}(0) = 1$ ✓
- If both true: $\chi A = 1, \chi B = 1 \rightarrow \chi A \cdot \chi B = 1 \rightarrow \overline{sg}(1) = 0$ ✓

Correct OR with characteristic functions:

```
A ∨ B should be: sḡ(sḡ(χA) · sḡ(χB))
```

Simplified OR for minimization:

```
A ∨ B as: sḡ(χA) + sḡ(χB) − sḡ(χA) · sḡ(χB)
```

10. Common Mistakes and Debugging

Mistake 1: Wrong Logical Operations

❌ **Wrong**: $A \wedge B \rightarrow \chi A \cdot \chi B$ (multiplication for AND) ✅ **Correct**: $A \wedge B \rightarrow \chi A + \chi B$ (addition for AND)

❌ **Wrong**: $A \vee B \rightarrow \chi A + \chi B$ (addition for OR)
✅ **Correct**: $A \vee B \rightarrow \chi A \cdot \chi B$ (multiplication for OR)

Mistake 2: Variable Duplication in w

❌ **Wrong**: $w = \langle y, t, y \rangle$ (y appears twice) ✅ **Correct**: $w = \langle y, t \rangle$ (each variable once)

Mistake 3: Including Function Parameters

❌ **Wrong**: $w = \langle x, y, t \rangle$ (x is the function parameter) ✅ **Correct**: $w = \langle y, t \rangle$ (exclude x)

Mistake 4: Forgetting Negated Sign

❌ **Wrong**: $\mu w.\ \chi H(x, x, (w)_1)$ (stops when $\chi H = 1$) ✅ **Correct**: $\mu w.\ \overline{sg}(\chi H(x, x, (w)_1))$ (stops when $\chi H = 1$, returns 0)

Mistake 5: Wrong Equality Conversion

❌ **Wrong**: $a = b \rightarrow sg(|a - b|)$ (1 when equal) ✅ **Correct**: $a = b \rightarrow \overline{sg}(|a - b|)$ (0 when equal)

Debugging Checklist

1. Check logical operations:

```
AND → Addition (+)
OR → Multiplication (·)
NOT → Negated sign (sḡ)
```

2. Verify minimization goal:

```
Want 0 when condition is TRUE (so μw stops)
Want 1 when condition is FALSE
```

3. Test simple cases:

```
If H(x,x,t) is true, χH = 1, sḡ(χH) = 0 ✓
If H(x,x,t) is false, χH = 0, sḡ(χH) = 1 ✓
```

4. Check variable encoding order:

```
w = ⟨var1, var2, var3⟩
var1 → (w)₁, var2 → (w)₂, var3 → (w)₃
```

11. Quick Reference Tables

Function Selection Guide

Concern	Use	Example
Only halting	H(e,x,t)	x ∈ Wx
Halting + output	S(e,x,y,t)	x ∈ Ex
Both	H ∧ S	x ∈ Wx ∩ Ex

Logical Conversion Table

Predicate	Characteristic	Goal (for μw)
A ∧ B	χA + χB	$\overline{sg}$(χA + χB)
A ∨ B	χA · χB	$\overline{sg}$(χA · χB)
¬A	$\overline{sg}$(χA)	χA
a = b	$\overline{sg}$(|a - b|)	|a - b|
a ≠ b	sg(|a - b|)	$\overline{sg}$(|a - b|)

Exercise Type Recognition

Pattern	Type	H/S Choice	Complexity
Ex ∩ B ≠ ∅	3	S	Simple
x ∈ Wx	3	H	Simple
∃k. k·x ∈ Wx	4	H	Medium
∀k. k+x ∈ Wx	4	Not r.e.	Complex
x ∈ Wx ∩ Ex	4	H ∧ S	Complex

This comprehensive guide provides the systematic approach needed to master semicharacteristic function construction with proper H/S function usage, tuple encoding, and predicate-to-function transformations for computability exercises.

7. Complete Examples from Exams

Exercise Type 3: Ex ∩ B ≠ ∅

```
A = {x | Ex ∩ B ≠ ∅}, B finite = {b₁, b₂, ..., bₙ}

Step 1: Property analysis
∃y ∈ Ex such that y ∈ B

Step 2: Variables
y (output), t (steps)

Step 3: Encoding
w = ⟨y, t⟩

Step 4: Construction
sc_A(x) = 1(μw. S(x, (w)₁, (w)₂, (w)₃) ∧ ((w)₂ ∈ B))

Step 5: Finite set handling
(w)₂ ∈ B = ((w)₂ = b₁) ∨ ... ∨ ((w)₂ = bₙ)
= sḡ(|(w)₂ - b₁|) · ... · sḡ(|(w)₂ - bₙ|)
```

Exercise Type 4: k·x ∈ Wx

```
B = {x | ∃k ∈ ℕ. k·x ∈ Wx}

Step 1: Property analysis
∃k such that ϕx(k·x) halts

Step 2: Variables
k (multiplier), t (steps)

Step 3: Encoding
w = ⟨k, t⟩

Step 4: Construction
sc_B(x) = 1(μw. H(x, (w)₁ · x, (w)₂))
```

Exercise Type 4: x ∈ Wx ∪ Ex

```
A = {x | x ∈ Wx ∪ Ex}

Step 1: Property analysis
x ∈ Wx OR x ∈ Ex

Step 2: Variables
t₁ (steps for Wx), z (input for Ex), t₂ (steps for Ex)

Step 3: Encoding
w = ⟨t₁, z, t₂⟩

Step 4: Construction
sc_A(x) = 1(μw. H(x, x, (w)₁) ∨ S(x, (w)₂, x, (w)₃))
= 1(μw. χH(x, x, (w)₁) · χS(x, (w)₂, x, (w)₃))
```

8. Common Patterns Summary

Recognition Checklist

Exercise Type 3 Indicators:

- Direct domain/codomain operations (Wx, Ex)
- Finite set intersections
- Simple membership testing
- Single quantification level

Exercise Type 4 Indicators:

- Self-reference (x appears multiple times)
- Universal quantification (∀k)
- Multiplicative/additive relations (k·x, k+x)
- Complex logical combinations

Encoding Decision Tree

```
1. Count distinct variables (excluding function parameters)
2. Variables = {v₁, v₂, ..., vₙ} → w = ⟨v₁, v₂, ..., vₙ⟩
3. Replace vᵢ with (w)ᵢ throughout
4. If π provided explicitly → use π instead of w
```

Function Choice Strategy

```
Domain only? → H(program, input, steps)
Domain + specific output? → S(program, input, output, steps)
Both needed? → H(...) ∧ S(...)
```

This systematic approach covers 95% of computability exercise patterns and provides the mechanical process for constructing semicharacteristic functions using H and S predicates with proper tuple encoding.