Self-Indifferentiability of C(X) := H(X) from a RO: n ⟶ n where H is RO: n⟶ n.

G0	**G1**	**G2**	**G3**	**G4**	**G5**	**G6**	**G7**

C(X):
Y = H(X)
Return Y
	C(X):
	If (X,Y) ∉ LC
	Y = H(X)
	LC ← (X,Y) : LC
	Else Find Y
	Return Y
		C(X):
		If (X,Y) ∉ LC
		If (X,Y) ∉ L
		Y ← {0,1}^{n}
		L ← (X,Y) : L
		Else Find Y
		LC ← (X,Y) : LC
		Else Find Y
		Return Y
			Unmodified	C(X):
				If (X,Y) ∉ LC
				If (X,Y) ∉ L
				Y ← {0,1}^{n}
				// L ← (X,Y) : L
				Else Find Y
				LC ← (X,Y) : LC
				Else Find Y
				Return Y
					Unmodified	C(X):
						If (X,Y) ∉ LC
						// If (X,Y) ∉ L
						Y ← {0,1}^{n}
						// Else Find Y
						LC ← (X,Y) : LC
						Else Find Y
						Return Y
							C(X):
							If (X,Y) ∉ LC
							Y ← {0,1}^{n}
							LC ← (X,Y) : LC
							Else Find Y
							Return Y

H(X):
If (X,Y) ∉ L
Y ← {0,1}^{n}
L ← (X,Y) : L
Else Find Y
Return Y
	Unmodified	Unmodified	H(X):
			If (X,Y) ∉ L
			If (X,Y) ∈ LC
			L ← (X,Y) : L
			Else Y ← {0,1}^{n}
			L ← (X,Y) : L
			Else Find Y
			Return Y
				Unmodified	H(X):
					If (X,Y) ∉ L
					If (X,Y) ∈ LC
					L ← (X,Y) : L
					Else Y ← {0,1}^{n}
					L ← (X,Y) : L
					LC ← (X,Y) : LC
					Else Find Y
					Return Y
						Unmodified	H(X):
							If (X,Y) ∉ L
							Y ← C(X)
							L ← (X,Y) : L
							Else Find Y
							Return Y

G0: The real game.

G1: Keep track of C queries.

G2: Inline the query to H within C.

G3: Prepare for moving handling of list L from C to H.

Add all the relevant entries in LC to L. (At this point this change won't add anything yet.)

G4: Remove handling of L from C.

An inconsistency could arise if X was queried to C (and was not added to L under C) and then later queried to H. But in this case H2 updates L with entries in LC.

G5: Prepare for moving checking list L from C.

Add all entries in L to LC. Preemptively complete chains.

G6: Anything in L is on LC so the check in the If statement always passed (and Else is never executed).

G7: Instead of LC entries use oracle C.

If the entry was on LC, C would return the same entry as in **G6**.

If not, in both **G6 and G7** a independent entry is sampelled and added to LC. In both games this new sample is returned under H.

The resulting simulator is:

Sim^C(X):
Return C(X)

Range Extension: Indifferentiability of C(X) := H(0|X)|H(1|X) from a RO: n-1 ⟶ 2n where H is RO: n⟶ n.

G0	G1	G2	G3	G4	G5	G6	G7	G8	G9

C(X):
Y0 = H(0|X)
Y1 = H(1|X)
Return Y0|Y1
	C(X):
	If (X,Y0|Y1) ∉ LC
	  Y0 = H(0|X)
	  Y1 = H(1|X)
	  LC ← (X,Y1|Y2):LC
	Else Find Y0|Y1 in LC
	Return Y0|Y1
		Unmodified	C(X):
			If (X,Y0|Y1) ∉ LC
			  If (0|X,Y) ∉ L
			  Y0,Y1 ← {0,1}^n
			  L ← (0|X,Y0) : L
			  L ← (1|X,Y1) : L
			  Else Find Y0
			  If (1|X,Y) ∉ L
			  Y1 ← {0,1}^n
			  L ← (1|X,Y1) : L
			  Else Find Y1
			  LC ← (X,Y0|Y1):LC
			Else Find Y0|Y1 in LC
			Return Y0|Y1
				C(X):
				If (X,Y0|Y1) ∉ LC
				  If (0|X,Y) ∉ L
				  Y0,Y1 ← {0,1}^n
				  L ← (0|X,Y0) : L
				  L ← (1|X,Y1) : L
				  Else Find Y0,Y1
				  LC ← (X,Y0|Y1):LC
				Else Find Y0|Y1 in LC
				Return Y0|Y1
					Unmodified	C(X):
						If (X,Y0|Y1) ∉ LC
						  If (0|X,Y) ∉ L
						  Y0,Y1 ← {0,1}^n
						// L ← (0|X,Y0) : L
						// L ← (1|X,Y1) : L
						  Else Find Y0,Y1
						  LC ← (X,Y0|Y1):LC
						Else Find Y0|Y1 in LC
						Return Y0|Y1
							Unmodified	C(X):
								If (X,Y0|Y1) ∉ LC
								// If (0|X,Y) ∉ L
								  Y0,Y1 ← {0,1}^n
								// Else Find Y0,Y1
								  LC ← (X,Y0|Y1) : LC
								Else Find Y0|Y1 in LC
								Return Y0|Y1
									Unmodified

H(b|X):
If (b|X,Yb) ∉ L
  Yb ← {0,1}^n
  L ← (b|X,Yb) : L
Else Find Yb in L
Return Yb
	Unmodified	H(b|X):
		If (b|X,Yb) ∉ L
		  Y0,Y1 ← {0,1}^n
		  L ← (0|X,Y0) :L
		  L ← (1|X,Y1) :L
		Else Find Yb in L
		Return Yb
			Unmodified	Unmodified	H(b|X):
					If (b|X,Yb) ∉ L
					  If (X,Y0|Y1) ∈ LC
					  L ← (0|X,Y0) : L
					  L ← (1|X,Y1) : L
					  Else
					  Y0,Y1 ← {0,1}^n
					  L ← (0|X,Y0) :L
					  L ← (1|X,Y1) :L
					Else Find Yb in L
					Return Yb
						Unmodified	H(b|X):
							If (b|X,Yb) ∉ L
							  If (X,Y0|Y1) ∈ LC
							  L ← (0|X,Y0) :L
							  L ← (1|X,Y1) :L
							  Else
							  Y0,Y1 ← {0,1}^n
							  L ← (0|X,Y0) : L
							  L ← (1|X,Y1) : L
							  LC ← (X,Y0|Y1):LC
							Else Find Yb in L
							Return Yb
								Unmodified	H(b|X):
									If (b|X,Yb) ∉ L
									  Y0|Y1 ← C(X)
									  L ← (0|X,Y0) : L
									  L ← (1|X,Y1) : L
									Else Find Yb in L
									Return Yb

G0: The real game.
G1: Keep track of C queries.
G2: Add entries for b = 0 and b = 1 in tandem: For each b we have that (b|X,Yb) ∉ L iff (0|X,Y0) ∉ L and (1|X,Y1) ∉ L.
G3: Inline the two H queries within C.
G4: Simplify due to change in **G2**.
G5: Prepare for moving handling of list L from C to H. For all entries X in LC add 0|X and 1|X to L. (This does nothing at this step since for X on LC has 0|X and 1|X are both in L.)
G6: Remove handling of L from C. H will handle the entries consistently since it looks at entries on LC.
G7: Always add the L entry to LC. Prepares us to stop relying on LC and use C instead.
G8: If some entry is in L it will be on LC too (due to **G7**). So remove If-Else.
G9: Use C instead of LC entries irrespectively of whether or not (X,Y) is on LC.

Simplifying H, we get the simulator below, which makes 1 C-query per invocation:

Sim^C(b|X):
Y0|Y1 ← C(X)
Return Yb

Question: Is a this range extender *multi-stage* indifferentiable?
Answer: Yes, because the simulator is stateless!

Domain Extension: Indifferentiability of C(X) := H2(H1(X1,X2),X3) from a RO: 3n ⟶ n where H1,H2 are independent RO: 2n ⟶ n.

G0	**G1**	**G2**	**G3**	**G4**	**G5**

C(X1,X2,X3):
Y1 = H1(X1,X2)
Y2 = H2(Y1,X3)
Return Y2
	C(X1,X2,X3):
	If ((X1,X2,X3),Y2) ∉ LC
	  Y1 = H1(X1,X2)
	  Y2 = H2(Y1,X3)
	  LC ← ((X1,X2,X3),Y2) : LC
	Else Find ((X1,X2,X3),Y2) ∈ LC
	Return Y2
		C(X1,X2,X3):
		If ((X1,X2,X3),Y2) ∉ LC
		  Y1 = H1(X1,X2)
		  If ((Y1,X3),Y2) ∉ L2
		  Y2 ← {0,1}^{n}
		  L2 ← ((Y1,X3),Y2) : L2
		  Else Find Y2 in L2
		  LC ← ((X1,X2,X3),Y2) : LC
		Else Find ((X1,X2,X3),Y2) ∈ LC
		Return Y2
			Unmodified	C(X1,X2,X3):
				If ((X1,X2,X3),Y2) ∉ LC
				  Y1 = H1(X1,X2)
				  If ((Y1,X3),Y2) ∉ L2
				  Y2 ← {0,1}^{n}
				// L2 ← ((Y1,X3),Y2) : L2
				  Else Find Y2 in L2
				  LC ← ((X1,X2,X3),Y2) : LC
				Else Find ((X1,X2,X3),Y2) ∈ LC
				Return Y2
					Unmodified

H1(X1,X2):
If ((X1,X2),Y1) ∉ L1
  Y1 ← {0,1}^{n}
  L1 ← ((X1,X2),Y1) : L1
Else Find Y1 in L1
Return Y1
	Unmodified	Unmodified	Unmodified	Unmodified	Unmodified

H2(Y1,X3):
If ((Y1,X3),Y2) ∉ L2
  Y2 ← {0,1}^{n}
  L2 ← ((Y1,X3),Y2) : L2
Else Find Y2 in L2
Return Y2
	Unmodified	Unmodified	H2(Y1,X3):
			If ((Y1,X3),Y2) ∉ L2:
			  If ((X1,X2,X3),Y2) ∈ LC & ((X1,X2),Y1) ∈ L1
			  L2 ← ((Y1,X3),Y2) : L2
			  Else Y2 ← {0,1}^{n}
			  L2 ← ((Y1,X3),Y2) : L2
			Else Find Y2 in L2
			Return Y2
				Unmodified	H2(Y1,X3):
					If ((Y1,X3),Y2) ∉ L2:
					  If ((X1,X2,X3),Y2) ∈ LC & ((X1,X2),Y1) ∈ L1
					  L2 ← ((Y1,X3),Y2) : L2
					  Else
					  Y2 ← {0,1}^{n}
					  If ((X1,X2),Y1) ∈ L1
					  LC ← ((X1,X2,X3),Y2) : LC
					  L2 ← ((Y1,X3),Y2) : L2
					Else Find Y2 in L2
					Return Y2

G0: The real game.
G1: Keep track of C queries.
G2: Inline H2.
G3: Add C queries relevant to the current H2 query to L2. (Does not execute at this point.)
G4: Remove handling of L2 from C. OK due to change in **G3** and keeping track of C queries introduced in **G1**.
G5: **Preemptively** create LC entry for H1 and H2 entries that form a chain.

G6	G7	G8	G9	G10	G11

Unmodified	C(X1,X2,X3):
	If ((X1,X2,X3),Y2) ∉ LC
	Y1 = H1(X1,X2)
	// If ((Y1,X3),Y2) ∉ L2
	Y2 ← {0,1}^{n}
	// Else Find Y2 in L2
	LC ← ((X1,X2,X3),Y2) : LC
	Else Find ((X1,X2,X3),Y2) ∈ LC
	Return Y2
		C(X1,X2,X3):
		If ((X1,X2,X3),Y2) ∉ LC
		// Y1 = H1(X1,X2)
		Y2 ← {0,1}^{n}
		LC ← ((X1,X2,X3),Y2) : LC
		Else Find ((X1,X2,X3),Y2) ∈ LC
		Return Y2
			Unmodified	Unmodified	Unmodified

H1(X1,X2):
If ((X1,X2),Y1) ∉ L1
Y1 ← {0,1}^{n}
If ((Y1,*),*) ∈ L2 SetFlag
L1 ← ((X1,X2),Y1) : L1
Else Find Y1 in L1
Return Y1
	Unmodified	Unmodified	H1(X1,X2):
			If ((X1,X2),Y1) ∉ L1
			Y1 ← {0,1}^{n}
			If ((*,*),Y1) ∈ L1 SetFlag
			If ((Y1,*),*) ∈ L2 SetFlag
			L1 ← ((X1,X2),Y1) : L1
			Else Find Y1 in L1
			Return Y1
				Unmodified	Unmodified

Unmodified	Unmodified	Unmodified	Unmodified	H2(Y1,X3):
				If ((Y1,X3),Y2) ∉ L2:
				If ((X1,X2,X3),Y2) ∈ LC & ((X1,X2),Y1) ∈ L1
				L2 ← ((Y1,X3),Y2) : L2
				Else
				If ((X1,X2),Y1) ∈ L1
				Y2 ← {0,1}^{n}
				LC ← ((X1,X2,X3),Y2) : LC
				Else Y2 ← {0,1}^{n}
				L2 ← ((Y1,X3),Y2) : L2
				Else Find Y2 in L2
				Return Y2
					H2(Y1,X3):
					If ((Y1,X3),Y2) ∉ L2:
					If ((X1,X2),Y1) ∈ L1
					Y2 ← C(X1,X2,X3)
					L2 ← ((Y1,X3),Y2) : L2
					Else Y2 ← {0,1}^{n}
					L2 ← ((Y1,X3),Y2) : L2
					Else Find Y2 in L2
					Return Y2

G6: Avoid **accidental chains**: Chains with where H1 query comes **after** a matching H2 query are excluded. This introduces a term: (qC+q1)(qC + q2) / 2^n

G7: Assume ((Y1,X3),Y2) is not in L2 (in which case Y1 is not needed in C). Suppose not, i.e., that ((Y1,X3),Y2) is in L2. We have two cases:

((X1,X2),Y1) was on L1 when ((Y1,X3),Y2) was added to L2. Then (X1,X2,X3,Y2) would be on LC due to the change in **G5**.

((X1,X2),Y1) was not on L1 when ((Y1,X3),Y2) was added to L2. Then it will be added later on since the H1 query under C will certainly add it. This however would set Flag.

G8: Remove the H1 call (Y1 is not used in C). The H1 entry will be created when needed. Difference: (qC+q1)(qC + q2)/2^n (flag is set more often in **G7** than in **G8**).

G9: Avoid collisions in H1. This means two different chains with different Y1 values would have the same C values iff they have the same H2 values. Difference: (qC+q1)(qC + q1)/2^n

G10: Use **different** Y2 for all entries ((X1,X2),Y1) was on L1. No change due to avoiding collisions in **G8** (entry on L1 is unique).

G11: Use C instead of LC. When ((X1,X2),Y1) is on L1 we either retrieve Y2 from LC or create a new one and added to LC. This is equivalent to a C call.

The resulting simulator, which makes at most 1 C-query per H2 call, is:

Sim1(X1,X2):	Sim2^C(Y1,X3):

If ((X1,X2),Y1) ∉ L1
Y1 ← {0,1}^{n}
If ((*,*),Y1) ∈ L1 SetFlag
If ((Y1,*),*) ∈ L2 SetFlag
L1 ← ((X1,X2),Y1) : L1
Else Find Y1 in L1
Return Y1
	If ((Y1,X3),Y2) ∉ L2:
	If ((X1,X2),Y1) ∈ L1
	Y2 ← C(X1,X2,X3)
	L2 ← ((Y1,X3),Y2) : L2
	Else Y2 ← {0,1}^{n}
	L2 ← ((Y1,X3),Y2) : L2
	Else Find Y2 in L2
	Return Y2

Extra Example: Indifferentiability of C(X) := H1(X) + H2(X) from a RO : n⟶ n where H1 and H2 are independent RO: n⟶ n.

G0	**G1**	**G2**	**G3**	**G4**	**G5**	**G6**	**G7**	**G8**	**G9**

C(X):
Y1 = H1(X)
Y2 = H2(X)
Z = Y1 + Y2
Return Z
	C(X):
	If (X,Z) ∉ LC
	  Y1 = H1(X)
	  Y2 = H2(X)
	  Z = Y1 + Y2
	  LC ← (X,Z) : LC
	Else Find Z in LC
	Return Z
		C(X):
		If (X,Z) ∉ LC
		  If (X,Y1) ∉ L1
		  Y1 ← {0,1}^{n}
		  L1 ← (X,Y1) : L1
		  Else Find Y1
		  Y2 = H2(X)
		  Z = Y1 + Y2
		  LC ← (X,Z) : LC
		Else Find Z in LC
		Return Z
			C(X):
			If (X,Z) ∉ LC
			  Y2 = H2(X)
			  If (X,Y1) ∉ L1
			  Z ← {0,1}^{n}
			  Y1 = Z + Y2
			  L1 ← (X,Y1) : L1
			  Else Find Y1
			  Z = Y1 + Y2
			  LC ← (X,Z) : LC
			Else Find Z in LC
			Return Z
				Unmodified	Unmodified	C(X):
						If (X,Z) ∉ LC
						  Y2 = H2(X)
						  If (X,Y1) ∉ L1
						  Z ← {0,1}^{n}
						  Y1 = Z + Y2
						// L1 ← (X,Y1) : L1
						  Else Find Y1
						  Z = Y1 + Y2
						  LC ← (X,Z) : LC
						Else Find Z in LC
						Return Z
							C(X):
							If (X,Z) ∉ LC
							  Y2 = H2(X)
							// If (X,Y1) ∉ L1
							  Z ← {0,1}^{n}
							  Y1 = Z + Y2
							// Else Find Y1
							// Z = Y1 + Y2
							  LC ← (X,Z) : LC
							Else Find Z in LC
							Return Z
								C(X):
								If (X,Z) ∉ LC
								// Y2 = H2(X)
								  Z ← {0,1}^{n}
								// Y1 = Z + Y2
								  LC ← (X,Z) : LC
								Else Find Z in LC
								Return Z
									Unmodified

H1(X):
If (X,Y1) ∉ L1
  Y1 ← {0,1}^{n}
  L1 ← (X,Y1) : L1
Else Find Y1 on L1
Return Y1
	Unmodified	Unmodified	Unmodified	H1(X):
				If (X,Y1) ∉ L1
				  If (X,Z) ∈ LC Find Z
				  Y1 = Z + H2(X)
				  Else
				  Y1 ← {0,1}^{n}
				  Z = Y1 + H2(X)
				  LC ← (X,Z) : LC
				  L1 ← (X,Y1) : L1
				Else Find Y1 in L1
				Return Y1
					H1(X):
					If (X,Y1) ∉ L1
					  If (X,Z) ∈ LC Find Z
					// Y1 = Z + H2(X)
					  Else
					  Z ← {0,1}^{n}
					  LC ← (X,Z) : LC
					  Y1 = Z + H2(X)
					  L1 ← (X,Y1) : L1
					Else Find Y1 in L1
					Return Y1
						Unmodified	Unmodified	Unmodified	H1(X):
									Return C(X) + H2(X)

H2(X):
If (X,Y2) ∉ L2
  Y2 ← {0,1}^{n}
  L2 ← (X,Y2) : L2
Else Find Y2 on L2
Return Y2
	Unmodified	Unmodified	Unmodified	Unmodified	Unmodified	Unmodified	Unmodified	Unmodified	Unmodified

G0: The real game.
G1: Keep track of the C queries.
G2: Inline H1.
G3: Change of random variable. Y1 remain iid and random for different X on L1. No change in output distribution.
G4: Two changes: 1) Check if Y1 could have been set in C. This is never reached (since if X in LC it will be automatically on L1).
2) Always add the current entry in L1 to LC (which is necessarily not in LC).
G5: Change of random variable.
G6: Remove handling of L1 from C. Consistency between LC and L1 is now ensured within H1 (due to the **first** change in **G4**).
G7: If X is in L1 it will be also in LC due to the operation of H1. Put differently, if X is not in LC it won't be in L1 either. So the first If check always holds: Remove if-Else.
G8: Y1 and Y2 are not used in C. Defer handling of Y2 to H2 and remove from C.
G9: H1 uses C(X) + H2(X). Irrespectively of (X,Z) being in LC or not this matches the output on **G8** (due to the **second** change in **G4**).
The resulting simulator,which makes at most 1 C-query per H1 call, is:

Sim1^C(X):	Sim2(X):

Return C(X) + Sim2(X)	If (X,Y2) ∉ L2
	  Y2 ← {0,1}^{n}
	  L2 ← (X,Y2) : L2
	Else Find Y2 in L2
	Return Y2