The Protein Folding Problem as Type Inference: Why AlphaFold Works and What It Teaches Us About Machine Learning

Matthew Long
Independent Researcher
matthew@yonedaai.com

December 2025

Abstract

We propose a novel theoretical framework for understanding the success of AlphaFold2 in solving the protein structure prediction problem. We argue that protein folding is fundamentally a *type-safe* operation: Anfinsen's thermodynamic hypothesis guarantees that amino acid sequences deterministically map to native structures, analogous to how well-typed programs in dependently typed programming languages have unique normal forms. This framing explains why deep learning succeeds dramatically at structure prediction while struggling with genuinely open-ended reasoning tasks. The key insight is that AlphaFold does not *discover* the physics of protein folding—it learns efficient *proof search heuristics* within a constraint space already defined by thermodynamics. We develop this analogy formally, examine its implications for understanding AI capabilities and limitations, and propose that the distinction between type-safe and type-unsafe problem domains may be a fundamental axis for predicting machine learning tractability. Our framework provides both explanatory power for existing results and predictive power for identifying new domains amenable to deep learning approaches.

Keywords: protein structure prediction, type theory, dependent types, AlphaFold, constraint satisfaction, machine learning theory, Anfinsen's dogma, proof search, computational biology

Contents

1 Introduction **4**
1.1 The Central Question 4
1.2 Preview of the Argument 4
1.3 Contributions 5
1.4 Organization 5

2 Background **5**
2.1 The Protein Folding Problem 5
2.1.1 Anfinsen's Thermodynamic Hypothesis 6
2.1.2 The Folding Funnel 6
2.2 Type Theory and Dependent Types 7
2.2.1 Basic Type Theory 7
2.2.2 Dependent Types 7
2.2.3 The Curry-Howard Correspondence 7
2.2.4 Normal Forms and Canonicity 8

2.3 The AlphaFold2 Architecture . . . 8
2.3.1 Inputs . . . 8
2.3.2 The Evoformer . . . 8
2.3.3 The Structure Module . . . 9
2.3.4 Recycling . . . 9
2.3.5 Training and Loss . . . 9

3 The Type-Theoretic Framework 9
3.1 The Fundamental Correspondence . . . 9
3.2 Formal Definitions . . . 10
3.3 Folding as Proof Search . . . 10
3.4 Constraint Propagation . . . 11
3.5 Chaperones as Type Checkers . . . 11

4 AlphaFold as Learned Proof Search 12
4.1 The Core Insight . . . 12
4.2 MSAs as Type Annotations . . . 12
4.2.1 Co-evolution as Dependent Constraints . . . 12
4.2.2 MSA Depth as Annotation Density . . . 12
4.3 Triangle Attention as Constraint Propagation . . . 12
4.3.1 The Triangle Inequality as a Refinement Type . . . 13
4.3.2 Triangle Updates as Propagation . . . 13
4.4 Recycling as Iterative Proof Search . . . 13
4.5 Empirical Evidence from Trajectory Analysis . . . 14

5 Implications for Machine Learning Theory 14
5.1 Type Safety as a Predictor of ML Tractability . . . 14
5.2 Case Studies . . . 14
5.2.1 Type-Safe Domains (ML Success) . . . 14
5.2.2 Type-Unsafe Domains (ML Struggles) . . . 15
5.2.3 Partially Type-Safe Domains . . . 15
5.3 Quantifying Type Safety . . . 15

6 Where the Analogy Holds and Fails 15
6.1 Success Cases . . . 15
6.2 Predicted Failure Modes . . . 15
6.2.1 Intrinsically Disordered Proteins (IDPs) . . . 16
6.2.2 Metamorphic/Fold-Switching Proteins . . . 16
6.2.3 Prion Diseases . . . 16
6.2.4 Orphan Proteins . . . 16
6.2.5 Large Hetero-Complexes . . . 16
6.3 Boundary Cases . . . 16
6.3.1 Temperature Dependence . . . 16
6.3.2 Post-Translational Modifications . . . 16
6.3.3 Chaperone-Dependent Folding . . . 17

7 Connections to Computational Complexity 17
7.1 NP-Hardness of Protein Folding . . . 17
7.2 Resolution: Natural Proteins Are Not Worst-Case . . . 17
7.3 The Funnel as Polynomial-Time Structure . . . 17

8 Evolution and Type Safety **17**
8.1 Multiple Levels of Type Checking . . . 17
8.2 Meta-Type-Systems: Controlled Unsafety . . . 18
8.2.1 Immune System . . . 18
8.2.2 Neural Plasticity . . . 18
8.2.3 Epigenetics . . . 18
8.2.4 Stress Responses . . . 18
8.3 Gradual Typing in Biology . . . 18

9 Future Directions **18**
9.1 Formal Verification of Predictions . . . 19
9.2 Type-Guided Protein Design . . . 19
9.3 Measuring Problem Type-Safety . . . 19
9.4 Hybrid Systems . . . 19
9.5 Biological Type Inference . . . 19

10 Related Work **20**
10.1 Computational Approaches to Protein Folding . . . 20
10.2 Constraint Satisfaction in Protein Folding . . . 20
10.3 Type Theory and Machine Learning . . . 20
10.4 AlphaFold Architecture Analysis . . . 20

11 Conclusion **20**
11.1 Summary . . . 20
11.2 The Deeper Insight . . . 21
11.3 Implications . . . 21
11.4 Final Remarks . . . 21

A Detailed Correspondence Table **24**

B Pseudocode for Type-Theoretic Folding **24**

C Mathematical Formalization **25**

1 Introduction

The November 2020 announcement that DeepMind's AlphaFold2 had achieved near-experimental accuracy in protein structure prediction marked one of the most significant advances in computational biology [Jumper et al., 2021]. The system achieved a median backbone accuracy of 0.96 Å RMSD on the CASP14 benchmark, compared to 2.8 Å for the next-best method—a dramatic improvement that effectively solved a 50-year grand challenge in molecular biology.

The magnitude of this achievement is difficult to overstate. The protein folding problem had been considered one of the most important unsolved problems in biology since Anfinsen's Nobel Prize-winning work in the 1970s [Anfinsen, 1973]. Previous computational approaches, whether based on molecular dynamics, homology modeling, or earlier machine learning methods, had made steady but incremental progress over decades. AlphaFold2 represented a discontinuous leap that brought computational predictions to the level of experimental techniques like X-ray crystallography and cryo-EM.

Yet the theoretical foundations of this success remain underexplored. Why did deep learning work so spectacularly for this particular problem? AlphaFold's Evoformer architecture is sophisticated, incorporating novel attention mechanisms and geometric reasoning [Jumper et al., 2021], but transformer-based neural networks have been applied to many domains with far less success. What makes protein folding special?

1.1 The Central Question

The standard narrative attributes AlphaFold's success to a combination of factors: the availability of large training datasets (the Protein Data Bank), the incorporation of evolutionary information through multiple sequence alignments, and architectural innovations like triangle attention. While these factors are certainly necessary, they do not constitute a sufficient explanation. Large datasets exist for many problems where deep learning fails. Evolutionary information is available for many biological prediction tasks that remain unsolved. And architectural innovations, while important, seem to provide incremental rather than transformative improvements in most domains.

We propose that the answer lies in a fundamental property of the problem itself: **protein folding is type-safe**.

1.2 Preview of the Argument

The amino acid sequence of a protein uniquely determines its native three-dimensional structure under physiological conditions—a principle known as Anfinsen's dogma or the thermodynamic hypothesis [Anfinsen, 1973]. This deterministic relationship between sequence and structure is precisely analogous to the relationship between well-typed terms and their normal forms in type theory.

In this paper, we develop this analogy in depth. We show that:

(1) Protein folding can be formalized as a type inference problem, where sequences are types and structures are canonical inhabitants.

(2) Anfinsen's dogma provides the *type safety* guarantee that makes the problem well-posed.

(3) AlphaFold succeeds by learning efficient *proof search heuristics* within a constraint space already defined by thermodynamics.

(4) The framework explains both AlphaFold's successes and its failure modes.

(5) The type-safe/type-unsafe distinction may be a fundamental predictor of machine learning tractability.

1.3 Contributions

This paper makes the following contributions:

- **A novel theoretical framework** connecting protein folding to type theory, providing a principled explanation for AlphaFold's success.
- **Formal definitions** of type safety in the protein folding context, with precise correspondences to type-theoretic concepts.
- **Explanatory power** for observed phenomena including AlphaFold's limitations with intrinsically disordered proteins and fold-switching proteins.
- **Predictive framework** for identifying other domains where similar approaches might succeed.
- **Connections** to computational complexity theory, showing how type safety relates to the tractability of NP-hard problems.

1.4 Organization

The remainder of this paper is organized as follows. Section 2 provides necessary background on Anfinsen's dogma, type theory, and the AlphaFold architecture. Section 3 develops the formal correspondence between protein folding and type inference. Section 4 analyzes AlphaFold through this lens, explaining how its components implement proof search. Section 5 explores implications for machine learning theory. Section 6 examines where the analogy holds and where it breaks down. Section 7 connects to computational complexity. Section 8 addresses the apparent tension with evolutionary type unsafety. Section 9 outlines future directions. Section 10 discusses related work, and Section 11 concludes.

2 Background

We begin by reviewing the necessary background in protein biochemistry, type theory, and the AlphaFold architecture. Readers familiar with these topics may wish to skim this section.

2.1 The Protein Folding Problem

Proteins are the molecular machines of life, performing virtually all cellular functions from catalysis to signaling to structural support. A protein is a linear polymer of amino acids, typically ranging from 50 to several thousand residues in length. There are 20 standard amino acids, each with distinct chemical properties determined by their side chains.

The *primary structure* of a protein is its amino acid sequence. The *secondary structure* refers to local folding patterns—primarily α-helices and β-sheets—stabilized by hydrogen bonding. The *tertiary structure* is the full three-dimensional arrangement of atoms, stabilized by a complex interplay of forces including hydrogen bonds, electrostatic interactions, van der Waals forces, and the hydrophobic effect. Some proteins also have *quaternary structure*, involving multiple polypeptide chains.

The protein folding problem asks: given the amino acid sequence (primary structure), can we predict the three-dimensional structure (tertiary structure)?

2.1.1 Anfinsen's Thermodynamic Hypothesis

In 1961, Christian Anfinsen performed a critical experiment with the enzyme ribonuclease A (RNase A) [Anfinsen, 1973]. He showed that when the enzyme was denatured (unfolded) by disrupting its disulfide bonds and then returned to physiological conditions, it spontaneously refolded into its active native conformation. Crucially, this refolding occurred *without external assistance*—the information to achieve the correct structure was contained entirely within the sequence.

This led to what became known as **Anfinsen's dogma** or the **thermodynamic hypothesis**:

> **Anfinsen's Dogma**
>
> The native structure of a protein is uniquely determined by its amino acid sequence and represents the global minimum of the Gibbs free energy under physiological conditions.

More formally, the thermodynamic hypothesis asserts three properties of the native state:

Definition 2.1 (Thermodynamic Hypothesis)**.** *For a protein with sequence s in its standard physiological environment, the native structure $\sigma^*(s)$ satisfies:*

(i) ***Uniqueness****: The sequence has no other configuration with comparable free energy. Formally, if $G(\sigma)$ denotes the Gibbs free energy of structure σ, then there exists a gap $\Delta > 0$ such that for all alternative structures $\sigma' \neq \sigma^*(s)$, we have $G(\sigma') - G(\sigma^*(s)) > \Delta$.*

(ii) ***Stability****: Small perturbations to environmental conditions do not change the minimum. The native state is a* stable *equilibrium.*

(iii) ***Kinetic Accessibility****: The native state can be reached from the unfolded ensemble in biologically relevant timescales, despite the astronomical number of possible conformations.*

The third condition addresses **Levinthal's paradox** [Levinthal, 1969]: a 100-residue protein has approximately 10^{100} possible conformations (assuming only 3 rotational states per bond), yet proteins fold in milliseconds to seconds. The resolution lies in the structure of the energy landscape.

2.1.2 The Folding Funnel

The modern understanding of protein folding invokes the concept of a **folding funnel** [Dill & MacCallum, 2012]. The energy landscape is not flat with a single global minimum—rather, it is funnel-shaped, with the native state at the bottom. As a protein folds, it does not need to search all conformations; instead, it follows the gradient down the funnel, progressively restricting its conformational space.

The funnel has the following properties:

- **High entropy, high energy** at the rim: the unfolded ensemble has many configurations but high average energy.
- **Decreasing entropy, decreasing energy** along the funnel walls: as the protein folds, both configurational options and energy decrease.
- **Single basin** at the bottom: the native state occupies a unique or small cluster of closely related configurations.

This funnel structure is what makes protein folding *tractable* despite the exponential search space. It is also, as we shall argue, what makes it *type-safe*.

2.2 Type Theory and Dependent Types

Type theory, originating with Bertrand Russell's resolution of paradoxes in set theory and formalized by Church, Martin-Löf, and others [Martin-Löf, 1984, Pierce, 2002], provides a foundation for both mathematics and computer science.

2.2.1 Basic Type Theory

In programming language theory, **types** classify programs by the kinds of values they compute. A type system assigns types to terms (programs), enabling detection of certain classes of errors before runtime.

Example 2.2 (Simple Types)**.** *In a simply-typed lambda calculus:*

- $3 : \mathsf{Int}$ *(the term* 3 *has type integer)*
- $\mathsf{true} : \mathsf{Bool}$ *(the term* true *has type boolean)*
- $(\lambda x.x + 1) : \mathsf{Int} \to \mathsf{Int}$ *(a function from integers to integers)*

The key property of type systems is **type safety**, often stated as:

Theorem 2.3 (Type Safety)**.** *Well-typed programs don't go wrong.*

More precisely, type safety comprises two properties:

- **Progress**: A well-typed term is either a value or can take a step.
- **Preservation**: If a well-typed term takes a step, the result is also well-typed.

2.2.2 Dependent Types

Dependent types extend simple types by allowing types to depend on values. This is a powerful generalization that enables types to encode invariants and specifications.

Example 2.4 (Vectors)**.** *Consider the type of vectors (lists of fixed length):*

$$\mathsf{Vec} : \mathsf{Type} \to \mathbb{N} \to \mathsf{Type}$$

Here $\mathsf{Vec}\ A\ n$ *is the type of vectors containing* n *elements of type* A*. The length* n *is a* value *that the type depends on. This enables:*

$$\mathsf{head} : \forall A.\, \mathsf{Vec}\ A\ (n+1) \to A$$

The type of head *guarantees it can only be called on non-empty vectors—a static guarantee impossible with simple types.*

2.2.3 The Curry-Howard Correspondence

A fundamental insight connecting type theory and logic is the **Curry-Howard correspondence** (also called "propositions as types"):

Logic	Type Theory
Proposition	Type
Proof	Term (inhabitant of type)
Implication $A \Rightarrow B$	Function type $A \to B$
Conjunction $A \wedge B$	Product type $A \times B$
Universal $\forall x.P(x)$	Dependent function $\Pi x.P(x)$

Under this correspondence, type checking becomes **proof verification**, and term construction becomes **proof search**. A well-typed term is a proof of the proposition corresponding to its type.

2.2.4 Normal Forms and Canonicity

In type theory, the **normal form** of a term is its fully reduced (simplified) version. A key property of many type systems is:

Theorem 2.5 (Strong Normalization)**.** *Every well-typed term has a unique normal form, and every reduction sequence terminates.*

For closed terms of ground type, there is often a unique **canonical form**—a "simplest" inhabitant. For instance, every closed term of type Bool reduces to either true or false.

This property—that types constrain their inhabitants severely enough that there is a unique (or small number of) canonical inhabitant(s)—is central to our analogy with protein folding.

2.3 The AlphaFold2 Architecture

AlphaFold2 [Jumper et al., 2021] is a neural network architecture for protein structure prediction. We provide a high-level overview of its components.

2.3.1 Inputs

AlphaFold2 takes as input:

1. **Primary sequence**: The amino acid sequence of the target protein.
2. **Multiple Sequence Alignment (MSA)**: A collection of evolutionarily related sequences, aligned to highlight conserved and co-evolving positions.
3. **Templates** (optional): Experimentally determined structures of homologous proteins.

The MSA is crucial: it provides evolutionary information that constrains the structure prediction. Co-evolving residue pairs often indicate spatial proximity in the folded structure.

2.3.2 The Evoformer

The **Evoformer** is the main processing component, consisting of 48 transformer-like blocks. It operates on two representations:

1. **MSA representation**: An $N_{\text{seq}} \times N_{\text{res}}$ array encoding information about each residue across all sequences in the alignment.
2. **Pair representation**: An $N_{\text{res}} \times N_{\text{res}}$ array encoding information about pairs of residues.

The key innovations in the Evoformer include:

- **Axial attention**: Attention computed separately over rows (sequences) and columns (residues) of the MSA.
- **Triangle attention**: Attention over residue triplets, enforcing the constraint that distances must satisfy the triangle inequality.
- **Triangle multiplicative updates**: Non-attention updates where edge ij is updated based on edges ik and kj, implementing a form of constraint propagation.
- **Outer product mean**: Communication from MSA to pair representation via outer products.

2.3.3 The Structure Module

The **Structure Module** converts the pair and MSA representations into 3D atomic coordinates. It represents each residue as a rigid body (rotation and translation) and uses:

- **Invariant Point Attention (IPA)**: An attention mechanism that operates on 3D points in a way that is invariant to global rotations and translations.
- **Iterative refinement**: The structure is refined over multiple layers, with each layer producing an improved prediction.

2.3.4 Recycling

AlphaFold2 uses **recycling**: the predicted structure is fed back through the network for multiple iterations, allowing progressive refinement. This is analogous to iterative proof search.

2.3.5 Training and Loss

The network is trained end-to-end on structures from the Protein Data Bank. Key loss components include:

- **FAPE (Frame Aligned Point Error)**: Measures the accuracy of predicted atom positions under many different alignments.
- **Auxiliary losses**: Including distogram prediction, masked MSA prediction, and side-chain accuracy.

3 The Type-Theoretic Framework

We now develop the formal correspondence between protein folding and type inference.

3.1 The Fundamental Correspondence

We propose the following mapping between the protein folding domain and type theory:

Protein Folding Domain	**Type Theory Domain**
Amino acid sequence	Type signature
Native 3D structure	Canonical term / normal form
Folding process	Term normalization / proof search
Thermodynamic constraints	Typing rules / constraints
Free energy landscape	Type-theoretic proof search space
Energy minimum	Normal form
Chaperone proteins	Type checker / proof assistant
Misfolded protein (prion)	Type error / stuck term
Multiple sequence alignment	Type annotations / hints
Co-evolution signal	Dependent type constraints

Table 1: Correspondence between protein folding and type theory

3.2 Formal Definitions

We can make this correspondence precise with the following definitions.

Definition 3.1 (Protein Type System)**.** *A* protein type system *consists of:*

(i) A set $\mathcal{A} = \{A_1, \ldots, A_{20}\}$ *of amino acids.*

(ii) A set $\mathcal{S} = \mathcal{A}^*$ *of sequences (finite strings over* $\mathcal{A}$*).*

(iii) A set $\mathcal{C}$ *of 3D conformations (assignments of coordinates to all atoms).*

(iv) An energy function $G : \mathcal{S} \times \mathcal{C} \to \mathbb{R}$ *giving the Gibbs free energy.*

(v) Environmental parameters E *(temperature, pH, ionic strength, etc.).*

Definition 3.2 (Structure Type)**.** *For a sequence* $s \in \mathcal{S}$ *and environment* E*, the* structure type $\mathsf{Structure}(s, E)$ *is defined as:*

$$\mathsf{Structure}(s, E) \triangleq \{\sigma \in \mathcal{C} \mid G(s, \sigma) = \min_{\sigma'} G(s, \sigma') \text{ and } \sigma \text{ is kinetically accessible from unfolded state}\}$$

This is the set of conformations that achieve the global free energy minimum and can be reached by the folding process.

Definition 3.3 (Type Safety (Anfinsen))**.** *A sequence* s *is* type-safe *under environment* E *if* $|\mathsf{Structure}(s, E)| = 1$*—that is, the structure type has exactly one inhabitant.*

Anfinsen's dogma can now be stated type-theoretically:

Theorem 3.4 (Anfinsen as Type Safety)**.** *For most naturally occurring protein sequences* s *under physiological conditions* E_{phys}*, the sequence is type-safe:* $|\mathsf{Structure}(s, E_{phys})| = 1$*.*

This is not a mathematical theorem to be proved, but rather an empirical claim about the distribution of natural proteins.

3.3 Folding as Proof Search

Under our framework, protein folding is proof search: the process of constructing the unique (or canonical) inhabitant of a type.

Definition 3.5 (Folding Function)**.** *The* folding function *is:*

$$\mathsf{fold} : (s : \mathcal{S}) \to E \to \mathsf{Maybe}(\mathsf{Structure}(s, E))$$

This is a dependent function: its return type depends on the input sequence. The Maybe *accounts for the possibility of failed folding.*

In Haskell-like pseudocode:

```
-- Amino acid sequence (the "type signature")
data Sequence = List AminoAcid

-- Folded structure as a dependent type
-- The structure depends on and is determined by the sequence
data Structure (s :: Sequence) where
  -- This is NOT a function -- it's a *proof*
  -- that the structure is consistent with sequence constraints

-- Folding as proof search
fold :: (s :: Sequence) -> Maybe (Structure s)
```

```
fold s = search (constraints s) (searchSpace s)

-- The key insight: for type-safe sequences,
-- there's exactly one valid Structure s
```

Listing 1: Protein folding as type inference

3.4 Constraint Propagation

The energy function G encodes a complex set of constraints. We can decompose it:

$$G(s, \sigma) = \sum_{\text{local}} E_{\text{bond}} + \sum_{\text{pairs}} E_{\text{nonbond}} + E_{\text{solvation}} + \cdots \tag{1}$$

Each term contributes constraints:

- **Bond terms**: Constraints on covalent geometry (bond lengths, angles).
- **Non-bonded terms**: Constraints on inter-residue distances (van der Waals, electrostatics).
- **Solvation**: Constraints from the hydrophobic effect.

The folding process is **constraint propagation**: as some constraints are satisfied, they restrict the space of possibilities for others, progressively narrowing to the unique solution.

Proposition 3.6 (Constraint Propagation)**.** *The folding funnel can be understood as the progressive satisfaction of constraints, where satisfying local constraints (secondary structure formation) enables and constrains the satisfaction of global constraints (tertiary packing).*

This hierarchical structure is analogous to type inference in dependently typed languages, where local type constraints propagate to determine global structure.

3.5 Chaperones as Type Checkers

Molecular chaperones like GroEL/GroES play a crucial role in protein folding [Hartl et al., 2011]. In our framework:

Definition 3.7 (Chaperone Function)**.** *A* chaperone *is a function:*

$$\mathsf{chaperone} : \mathsf{PartialStructure}(s) \to \mathsf{Either}(\mathsf{Error}, \mathsf{PartialStructure}(s))$$

that validates partial folds and provides guidance when automatic inference fails.

Chaperones do not change the target structure—they assist the search process. Key chaperone functions include:

1. **Isolation**: Preventing ill-typed interactions (aggregation) during folding.
2. **Validation**: Checking that partial folds satisfy constraints.
3. **Guidance**: Providing "type hints" that narrow the search space.
4. **Reset**: Allowing backtracking from kinetically trapped states.

This is precisely the role of a type checker or proof assistant in dependent type theory.

4 AlphaFold as Learned Proof Search

With our framework established, we can now analyze AlphaFold through this lens.

4.1 The Core Insight

Central Claim

AlphaFold is not discovering protein physics—it is learning efficient **proof search heuristics** for a constraint satisfaction problem whose solution space is already defined by thermodynamics.

The network does not need to learn the laws of physics from scratch. The physics is already encoded in the training data: each (s, σ) pair in the PDB is a solved instance of the constraint satisfaction problem. AlphaFold learns patterns in *how* these constraints are satisfied, not *what* the constraints are.

4.2 MSAs as Type Annotations

AlphaFold's heavy reliance on Multiple Sequence Alignments (MSAs) makes perfect sense in our framework. Evolution provides **additional type constraints** that narrow the proof search space.

4.2.1 Co-evolution as Dependent Constraints

When two residue positions co-evolve (mutate in correlated ways), this indicates they are in spatial proximity in the folded structure. In type-theoretic terms, co-evolution reveals **dependent constraints**—constraints on pairs of variables that must be satisfied together.

Example 4.1 (Co-evolution as Type Constraint)**.** *If positions i and j show strong co-evolution, this can be formalized as a constraint:*

$$\mathsf{distance}(\sigma[i], \sigma[j]) < \delta \quad \textit{for some threshold } \delta$$

This is a refinement type: the structure type is refined to only include conformations satisfying this distance constraint.

4.2.2 MSA Depth as Annotation Density

AlphaFold's accuracy degrades dramatically when MSA depth falls below approximately 30 sequences [Jumper et al., 2021]. This makes sense: without sufficient type annotations, the constraint space is too weakly specified for efficient proof search.

Proposition 4.2 (MSA Depth Threshold)**.** *There exists a threshold $N_{min} \approx 30$ such that for MSA depth $N < N_{min}$, the type constraints from co-evolution are insufficient to uniquely determine the structure, leading to increased prediction error.*

Conversely, very deep MSAs (thousands of sequences) provide diminishing returns—once the type is sufficiently constrained, additional annotations are redundant.

4.3 Triangle Attention as Constraint Propagation

The Evoformer's triangle attention mechanism directly implements constraint propagation for geometric consistency.

4.3.1 The Triangle Inequality as a Refinement Type

Any valid 3D structure must satisfy the triangle inequality: for any three residues i, j, k,

$$|d_{ij} - d_{jk}| \leq d_{ik} \leq d_{ij} + d_{jk}$$

where d_{xy} denotes the distance between residues x and y.

In type-theoretic terms, this is a **refinement type**: the distance matrix type is refined to only admit physically realizable configurations.

```
-- Distance matrix with triangle inequality refinement
data DistanceMatrix where
  MkDM :: (dm :: Matrix Double)
       -> (forall i j k. |dm[i,j] - dm[j,k]| <= dm[i,k] <= dm[i,j] + dm
          [j,k])
       -> DistanceMatrix
```

Listing 2: Triangle inequality as refinement type

4.3.2 Triangle Updates as Propagation

The triangle multiplicative updates implement constraint propagation: when edge ij is updated based on edges ik and kj, this propagates type information through the constraint graph.

$$\mathbf{z}_{ij} \leftarrow \mathbf{z}_{ij} + \sum_k f(\mathbf{z}_{ik}, \mathbf{z}_{kj}) \tag{2}$$

The network learns which propagation patterns most efficiently eliminate invalid configurations—in other words, it learns efficient proof search strategies.

4.4 Recycling as Iterative Proof Search

AlphaFold's recycling mechanism—feeding predictions back through the network for iterative refinement—directly implements proof search.

Algorithm 1 AlphaFold as Iterative Proof Search

Input: Sequence s, MSA M, templates T
Initialize: Structure $\sigma_0 \leftarrow$ identity
for $r = 1$ to R **do** ▷ Recycling iterations
    Compute MSA representation from M
    Compute pair representation from M, T, σ_{r-1}
    for $b = 1$ to 48 **do** ▷ Evoformer blocks
        Propagate constraints via triangle attention
        Update representations
    end for
    $\sigma_r \leftarrow \mathsf{StructureModule}(\mathsf{pair}, \mathsf{single})$
end for
Return: σ_R

Each iteration produces a candidate proof (structure) which is then refined by the constraint-encoding layers. The trajectory of intermediate structures shows the network progressively eliminating invalid configurations until it converges on the unique valid proof.

4.5 Empirical Evidence from Trajectory Analysis

Jumper et al. [Jumper et al., 2021] trained auxiliary structure modules to decode intermediate representations, revealing the trajectory of structural beliefs through the network:

- **Easy proteins** (e.g., LmrP, T1024): The correct structure emerges within the first few layers. The type constraints are sufficient to rapidly eliminate alternatives.
- **Hard proteins** (e.g., SARS-CoV-2 ORF8, T1064): The network searches and rearranges secondary structure elements for many layers before settling on a solution. The proof search requires more backtracking.

This is exactly what our framework predicts: the difficulty of proof search depends on how constraining the type is.

5 Implications for Machine Learning Theory

Our framework has broader implications for understanding when machine learning will succeed.

5.1 Type Safety as a Predictor of ML Tractability

We propose that **type safety** is a fundamental axis for predicting machine learning tractability:

Definition 5.1 (Type-Safe Problem Domain)**.** *A problem domain is* type-safe *if:*

(i) Inputs deterministically specify outputs (up to equivalence).

(ii) The mapping has sufficient structure that verification is tractable.

(iii) The constraint space has a "funnel" structure guiding search to solutions.

Definition 5.2 (Type-Unsafe Problem Domain)**.** *A problem domain is* type-unsafe *if:*

(i) Inputs underdetermine outputs (many valid responses).

(ii) Verification is as hard as generation.

(iii) The "solution" space lacks structure to guide search.

5.2 Case Studies

5.2.1 Type-Safe Domains (ML Success)

1. **Protein structure prediction**: Type-safe by Anfinsen's dogma. AlphaFold succeeded spectacularly.
2. **Image classification**: The mapping from image to (ground truth) label is deterministic. CNNs achieve superhuman accuracy.
3. **Game playing (Go, Chess)**: Perfect information games have deterministic optimal play. AlphaGo/AlphaZero achieved superhuman performance.
4. **Speech recognition**: Acoustic signals (approximately) determine phonemes. Modern ASR achieves human-level accuracy.

5.2.2 Type-Unsafe Domains (ML Struggles)

1. **Open-ended reasoning**: No unique correct answer to "What should I do?" or "Is this argument valid?" Current LLMs struggle with reliability.
2. **Creative writing**: Many valid poems about love. Generation is easy but quality is undefined.
3. **Theorem proving**: The search space is enormous and lacks the funnel structure of protein folding. Current systems can verify but struggle to discover proofs.
4. **Real-world robotics**: The environment is open-ended and partially observable. Current systems remain brittle.

5.2.3 Partially Type-Safe Domains

1. **Machine translation**: Approximately type-safe (meaning is preserved) but admits variation (multiple valid translations). Neural MT works well but not perfectly.
2. **Code generation**: Specification constrains output, but not uniquely. Copilot helps but doesn't replace programmers.

5.3 Quantifying Type Safety

Can we measure the "type safety" of a domain to predict ML tractability?

Conjecture 5.3 (Type Safety Metric)**.** *For a problem domain with input space $\mathcal{X}$ and output space $\mathcal{Y}$, define:*

$$\tau = \mathbb{E}_{x \sim \mathcal{X}} \left[\frac{1}{\log |\{y : y \text{ is valid for } x\}|} \right]$$

Higher τ (fewer valid outputs per input) predicts better ML performance.

For protein folding, τ is approximately 1 (unique structure per sequence), explaining AlphaFold's success. For creative writing, $\tau \to 0$ (infinitely many valid outputs).

6 Where the Analogy Holds and Fails

No analogy is perfect. We examine the boundaries of our framework.

6.1 Success Cases

The type-theoretic framework successfully explains:

1. **Why MSAs matter**: They provide type annotations narrowing search.
2. **Why triangle attention works**: It enforces geometric refinement types.
3. **Why recycling helps**: It implements iterative proof search.
4. **Why accuracy correlates with MSA depth**: More annotations enable better inference.
5. **Why some proteins are "hard"**: Their types are less constraining.

6.2 Predicted Failure Modes

Our framework predicts several failure modes, all observed empirically:

6.2.1 Intrinsically Disordered Proteins (IDPs)

Many proteins or protein regions lack a unique native structure, existing as dynamic ensembles [Dyson & Wright, 2005]. In our framework, these are **type-underdetermined sequences**—the type admits multiple inhabitants.

Proposition 6.1 (IDP Failure)**.** *For intrinsically disordered sequences s, $|\mathsf{Structure}(s, E)| > 1$ (possibly infinite). AlphaFold's assumption of a unique target structure fails, leading to low confidence scores and inaccurate predictions.*

Indeed, AlphaFold's pLDDT (predicted local distance difference test) scores are systematically low for disordered regions, and the predicted structures are unreliable.

6.2.2 Metamorphic/Fold-Switching Proteins

Some proteins adopt radically different structures depending on conditions (pH, binding partners, etc.) [Porter & Looger, 2018]. These are **context-dependent types**—the constraints change with environment.

Example 6.2 (RfaH)**.** *The transcription factor RfaH has a C-terminal domain that can exist as either an α-helix bundle or a β-barrel, depending on context. AlphaFold typically predicts only one form, missing the alternative.*

6.2.3 Prion Diseases

Prions are proteins that adopt alternative stable conformations capable of templating their misfolded state [Prusiner, 1998]. These represent **non-unique minima**—violations of the uniqueness condition in Anfinsen's dogma.

In type-theoretic terms, prions correspond to type systems with **non-confluence**, where different reduction paths lead to different normal forms.

6.2.4 Orphan Proteins

Proteins with very few homologs have shallow MSAs, providing insufficient type annotations for efficient proof search. AlphaFold's accuracy degrades significantly for such cases.

6.2.5 Large Hetero-Complexes

When a protein's structure is determined primarily by interactions with other chains not present in the MSA, the type constraints are incomplete. AlphaFold struggles with proteins having many hetero-contacts.

6.3 Boundary Cases

Some phenomena lie at the boundary of our framework:

6.3.1 Temperature Dependence

Protein stability depends on temperature. At sufficiently high temperatures, proteins denature. Our framework handles this through the environmental parameter E: the type $\mathsf{Structure}(s, E)$ is empty for denaturing conditions.

6.3.2 Post-Translational Modifications

Modifications like phosphorylation can alter structure. These change the effective sequence (and thus type) after translation.

6.3.3 Chaperone-Dependent Folding

Some proteins absolutely require chaperones for correct folding in vivo. Our framework accommodates this: the type $\mathsf{Structure}(s, E)$ includes chaperone presence as part of E, or we can view chaperones as providing additional type hints not present in the sequence alone.

7 Connections to Computational Complexity

7.1 NP-Hardness of Protein Folding

Protein structure prediction is NP-hard in general lattice models [Crescenzi et al., 1998]. Finding the global energy minimum is computationally intractable for worst-case instances.

Yet AlphaFold solves the problem in GPU-minutes for most natural proteins. This is not a contradiction.

7.2 Resolution: Natural Proteins Are Not Worst-Case

Theorem 7.1 (Average-Case vs. Worst-Case)**.** *While protein folding is NP-hard in the worst case, natural proteins are not worst-case instances. Evolution has selected for sequences that fold—sequences whose type constraints are strong enough to make proof search tractable.*

Informal Argument. Sequences that fold to unique structures in biologically relevant timescales have been selected by evolution. Sequences with "hard" folding landscapes (many local minima, rough energy surfaces) would fold slowly, misfold, or aggregate—all selected against. The result is that the PDB contains only "easy" instances. □

This parallels type inference in dependent type languages:

Proposition 7.2 (Type Inference Analogy)**.** *While type inference for dependently typed languages is undecidable in general, practical type checkers succeed because real programs have enough type annotations to make inference tractable. Similarly, real proteins have enough thermodynamic constraints to make structure prediction tractable.*

7.3 The Funnel as Polynomial-Time Structure

The folding funnel provides polynomial-time structure to an NP-hard problem:

Proposition 7.3 (Funnel Tractability)**.** *If the energy landscape has a funnel shape with depth polynomial in sequence length and width (number of configurations at each level) bounded by a polynomial, then gradient-based search finds the minimum in polynomial time.*

Natural proteins have evolved such funnel-shaped landscapes. This is the physical analog of having sufficient type structure for efficient proof search.

8 Evolution and Type Safety

An apparent paradox: if protein folding is type-safe, isn't evolution type-unsafe? Mutations are random changes with no regard for type correctness.

8.1 Multiple Levels of Type Checking

The resolution lies in recognizing **multiple levels of type checking**:

1. **Individual level (runtime)**: Protein folding IS type-safe. The ribosome faithfully executes the sequence, physics determines the fold, chaperones validate. No randomness.

2. **Population level (compile time)**: Evolution operates on the "source code" (genome) across generations. This is where type unsafety occurs.

3. **Selection (type checker)**: Organisms that don't "type-check" against physical reality (proteins don't fold, metabolic pathways fail) are selected against. Death is the ultimate type error.

Proposition 8.1 (Two-Level Type System)**.** *Evolution operates as a two-level type system:*

- ***Static typing*** *(within individual): Sequence → Structure is deterministic.*
- ***Dynamic typing*** *(across generations): Mutations can produce any sequence; selection filters for type-correct ones.*

8.2 Meta-Type-Systems: Controlled Unsafety

Evolution has developed mechanisms for controlled type unsafety—ways to relax type checking under specific conditions:

8.2.1 Immune System

V(D)J recombination and somatic hypermutation are **sanctioned randomness**. The type system has a carve-out: "randomness allowed here, but only for antibody genes, only in B cells, only under controlled conditions."

8.2.2 Neural Plasticity

Synaptic weights change, but the basic architecture doesn't. This is analogous to runtime configuration changes within a type-safe framework.

8.2.3 Epigenetics

Gene expression can be modified without changing the sequence—feature flags rather than source code changes.

8.2.4 Stress Responses

Heat shock proteins, the SOS response, and similar mechanisms relax quality control under stress. This is like switching to permissive mode: "We're dying anyway, try things that might not type-check."

8.3 Gradual Typing in Biology

Proposition 8.2 (Biological Gradual Typing)**.** *Life has evolved a form of* ***gradual typing****: mostly type-safe operation with controlled escape hatches for exploration. The type safety isn't violated—it's parameterized by context.*

This may explain why biology achieves robust function despite apparent messiness: the core operations are type-safe, while innovation happens at carefully controlled boundaries.

9 Future Directions

Our framework suggests several research directions:

9.1 Formal Verification of Predictions

Can we develop formal type checkers for predicted protein structures?

Definition 9.1 (Protein Type Checker)**.** *A* protein type checker *would verify that a predicted structure σ satisfies all constraints encoded in sequence s:*

- *Geometric constraints (bond lengths, angles, clashes)*
- *Energy constraints (local minima condition)*
- *Evolutionary constraints (consistency with MSA)*

Such a checker could provide formal guarantees on prediction quality, beyond the current empirical metrics.

9.2 Type-Guided Protein Design

Can we design proteins by specifying desired structural types and searching for inhabitants?

Definition 9.2 (Inverse Folding as Type Inhabitation)**.** *Given a target structure (type) σ, find a sequence s such that* $\mathsf{fold}(s) = \sigma$*. This is the type inhabitation problem.*

Current protein design methods (e.g., ProteinMPNN) can be understood through this lens as solving inverse type inference.

9.3 Measuring Problem Type-Safety

Can we develop metrics for the "type safety" of a domain that predict ML tractability?

Conjecture 9.3 (Type-Safety Metric Research Program)**.** *Develop a rigorous definition of domain type-safety measurable from data, and empirically validate that it predicts ML performance across domains.*

9.4 Hybrid Systems

Can we combine learned proof search heuristics with formal verification for partially type-safe domains?

[Verified Neural Networks] For domains with partial type safety, use neural networks for search and formal methods for verification:

1. Neural network proposes candidate solutions
2. Formal type checker verifies satisfiability of constraints
3. Feedback loop refines search

This hybrid approach could extend ML success to domains currently considered intractable.

9.5 Biological Type Inference

Can we extend the framework to other biological inference problems?

- **RNA structure**: Also type-safe (sequence determines structure), but less constrained than proteins.
- **Protein-protein interactions**: Partially type-safe (binding depends on complementary surfaces).

- **Gene regulatory networks**: Less type-safe (many valid regulatory states for a given genome).
- **Developmental biology**: Complex constraints from morphogen gradients might provide type structure.

10 Related Work

10.1 Computational Approaches to Protein Folding

The history of computational protein structure prediction spans five decades [Dill & MacCallum, 2012]. Early approaches used molecular dynamics [Karplus & McCammon, 2002] and energy minimization. Statistical methods exploited homology [Marti-Renom et al., 2000] and fragment assembly [Simons et al., 1997]. The introduction of co-evolution analysis [Marks et al., 2011] enabled contact prediction, which was later enhanced by deep learning [Senior et al., 2020].

AlphaFold2 [Jumper et al., 2021] represented a discontinuous advance, achieving near-experimental accuracy. Concurrent work by RoseTTAFold [Baek et al., 2021] achieved similar results with a smaller network. More recent work includes ESMFold [Lin et al., 2023], which replaces MSAs with protein language models, and AlphaFold3 [Abramson et al., 2024], which extends to protein-ligand complexes.

Our work provides a theoretical framework for understanding why these approaches succeed.

10.2 Constraint Satisfaction in Protein Folding

The connection between protein folding and constraint satisfaction has been explored previously [Dal Palù et al., 2004, Backofen, 1998]. Our contribution is to formalize this connection through the lens of type theory, enabling new insights about AlphaFold's architecture and ML tractability more broadly.

10.3 Type Theory and Machine Learning

Recent work has explored connections between type theory and machine learning [McLaughlin & Fang, 2016], particularly in the context of probabilistic programming [Goodman et al., 2012]. Our framework extends this direction to analyze the structure of problem domains that determine ML success.

10.4 AlphaFold Architecture Analysis

Several papers have analyzed AlphaFold's architecture [Ahdritz et al., 2022, Wu et al., 2022]. Our work complements these by providing a theoretical interpretation of why the architecture works, rather than describing what it does.

11 Conclusion

11.1 Summary

We have presented a type-theoretic framework for understanding protein folding and AlphaFold's success:

1. **Protein folding is type-safe**: Anfinsen's dogma guarantees that sequences uniquely determine structures, analogous to how well-typed terms have unique normal forms.
2. **AlphaFold learns proof search**: The network doesn't discover physics—it learns efficient heuristics for navigating a constraint space already defined by thermodynamics.

3. **Architecture reflects type structure**: MSAs provide type annotations, triangle attention enforces refinement types, recycling implements iterative proof search.

4. **Failures occur where type safety fails**: IDPs, fold-switchers, and shallow-MSA proteins violate the assumptions of the framework.

5. **Type safety predicts ML tractability**: The framework extends to other domains, suggesting a fundamental axis for understanding when deep learning will succeed.

11.2 The Deeper Insight

The deepest insight is this: **machine learning systems don't need to discover the laws of physics or the nature of protein chemistry. They need to learn efficient navigation of constraint spaces that physics has already defined.**

When those constraint spaces have the right structure—when they're type-safe—learning can succeed spectacularly. When they lack such structure, no amount of data or compute will help.

11.3 Implications

For practitioners, this suggests:

- Before applying ML to a domain, analyze its type structure.
- Look for domains where inputs (approximately) uniquely determine outputs.
- Seek auxiliary information (like MSAs) that provides type annotations.
- Design architectures that encode domain constraints (like triangle inequality).

For theorists, this opens:

- Formalizing type safety for other domains.
- Developing metrics that predict ML tractability.
- Understanding the relationship between computational complexity and type structure.
- Exploring hybrid systems combining learning and formal verification.

11.4 Final Remarks

AlphaFold's success is not merely a triumph of deep learning scale. It represents the identification of a problem with the right mathematical structure for machine learning to succeed. Protein folding is a showcase of what becomes possible when physics guarantees type safety—when the universe provides a proof assistant.

Understanding this may be the key to identifying the next problems where similar triumphs are possible, and to honestly acknowledging the boundaries of what machine learning can achieve.

Acknowledgments

We thank the DeepMind team for making AlphaFold's architecture and training details publicly available, enabling theoretical analysis of this remarkable system. We also thank the structural biology community for decades of painstaking work building the Protein Data Bank, without which none of this would be possible.

References

Abramson, J., et al. (2024). Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature*, 630, 493–500.

Ahdritz, G., et al. (2022). OpenFold: Retraining AlphaFold2 yields new insights into its learning mechanisms and capacity for generalization. *bioRxiv*.

Anfinsen, C.B. (1973). Principles that govern the folding of protein chains. *Science*, 181(4096), 223–230.

Backofen, R. (1998). Using constraint programming for lattice protein folding. *Pacific Symposium on Biocomputing*, 387–398.

Baek, M., et al. (2021). Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 373(6557), 871–876.

Crescenzi, P., Goldman, D., Papadimitriou, C., Piccolboni, A., & Yannakakis, M. (1998). On the complexity of protein folding. *Journal of Computational Biology*, 5(3), 423–465.

Dal Palù, A., Dovier, A., & Fogolari, F. (2004). Constraint logic programming approach to protein structure prediction. *BMC Bioinformatics*, 5, 186.

Dill, K.A. & MacCallum, J.L. (2012). The protein-folding problem, 50 years on. *Science*, 338(6110), 1042–1046.

Dyson, H.J. & Wright, P.E. (2005). Intrinsically unstructured proteins and their functions. *Nature Reviews Molecular Cell Biology*, 6, 197–208.

Goodman, N.D., Mansinghka, V.K., Roy, D.M., Bonawitz, K., & Tenenbaum, J.B. (2012). Church: a language for generative models. *arXiv:1206.3255*.

Hartl, F.U., Bracher, A., & Hayer-Hartl, M. (2011). Molecular chaperones in protein folding and proteostasis. *Nature*, 475, 324–332.

Jumper, J., et al. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, 596, 583–589.

Karplus, M. & McCammon, J.A. (2002). Molecular dynamics simulations of biomolecules. *Nature Structural Biology*, 9, 646–652.

Levinthal, C. (1969). How to fold graciously. *Mossbauer Spectroscopy in Biological Systems*, 67, 22–24.

Lin, Z., et al. (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637), 1123–1130.

Marks, D.S., et al. (2011). Protein 3D structure computed from evolutionary sequence variation. *PLoS ONE*, 6(12), e28766.

Martin-Löf, P. (1984). *Intuitionistic Type Theory*. Bibliopolis.

Marti-Renom, M.A., et al. (2000). Comparative protein structure modeling of genes and genomes. *Annual Review of Biophysics and Biomolecular Structure*, 29, 291–325.

McLaughlin, C. & Fang, D. (2016). Functional types for machine learning. *arXiv:1612.01153*.

Pierce, B.C. (2002). *Types and Programming Languages*. MIT Press.

Porter, L.L. & Looger, L.L. (2018). Extant fold-switching proteins are widespread. *Proceedings of the National Academy of Sciences*, 115(23), 5968–5973.

Prusiner, S.B. (1998). Prions. *Proceedings of the National Academy of Sciences*, 95(23), 13363–13383.

Senior, A.W., et al. (2020). Improved protein structure prediction using potentials from deep learning. *Nature*, 577, 706–710.

Simons, K.T., Kooperberg, C., Huang, E., & Baker, D. (1997). Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions. *Journal of Molecular Biology*, 268(1), 209–225.

Tunyasuvunakool, K., et al. (2021). Highly accurate protein structure prediction for the human proteome. *Nature*, 596, 590–596.

Wu, R., et al. (2022). High-resolution de novo structure prediction from primary sequence. *bioRxiv*.

A Detailed Correspondence Table

Protein Folding	Type Theory	Notes
Amino acid sequence	Type signature	Input specification
Native structure	Normal form/canonical term	Unique output
Folding process	Normalization/proof search	Computation
Energy function	Typing judgment derivation	Validity criterion
Free energy minimum	Normal form property	Uniqueness guarantee
Thermodynamic constraints	Typing rules	Inference rules
Chaperone proteins	Type checker/assistant	Guided verification
MSA (evolutionary data)	Type annotations	Inference hints
Co-evolution	Dependent constraints	Correlated variables
Triangle inequality	Refinement type	Geometric constraint
Misfolded protein	Type error/stuck term	Computation failure
Prion (alternate stable)	Non-confluence	Multiple normal forms
IDP (disordered)	Underdetermined type	No unique inhabitant
Fold-switching	Context-dependent type	Environment matters
Chaperonin cavity	Isolation/sandbox	Controlled evaluation
Heat shock response	Permissive type mode	Relaxed checking

Table 2: Extended correspondence between protein folding and type theory

B Pseudocode for Type-Theoretic Folding

```
-- Basic types
type AminoAcid = Char  -- One of 20 standard amino acids
type Sequence = [AminoAcid]
type Coordinates = [(Float, Float, Float)]

-- Environment includes temperature, pH, etc.
data Environment = Env { temperature :: Float
                       , pH :: Float
                       , ionicStrength :: Float
                       , chaperones :: [Chaperone] }

-- Dependent type: Structure depends on sequence
data Structure :: Sequence -> Type where
  -- A structure is valid only if it satisfies all constraints
  MkStructure :: (s :: Sequence)
              -> Coordinates
              -> ConstraintsSatisfied s coords
              -> Structure s

-- The constraint satisfaction proof
data ConstraintsSatisfied :: Sequence -> Coordinates -> Type where
  AllSat :: BondConstraints s c
         -> NonBondedConstraints s c
         -> SolvationConstraints s c
         -> TriangleInequality c
         -> ConstraintsSatisfied s c

-- Folding function with dependent return type
fold :: (s :: Sequence) -> Environment -> Maybe (Structure s)
fold s env =
```

```
  let initialEnsemble = unfoldedEnsemble s
      searchResult = proofSearch (constraints s env) initialEnsemble
  in case searchResult of
       Found proof -> Just (MkStructure s (extractCoords proof) proof)
       NotFound -> Nothing

-- AlphaFold is learned proof search
alphaFold :: (s :: Sequence) -> MSA -> Maybe Template -> Structure s
alphaFold s msa template =
  let typeHints = extractTypeHints msa
      initialGuess = maybe identity template
      refined = recycleN 3 (evoformer typeHints) initialGuess
  in structureModule refined
```

Listing 3: Extended pseudocode for protein folding types

C Mathematical Formalization

Definition C.1 (Formal Protein Type System). *A formal protein type system $\mathcal{P}$ consists of:*

- *A finite alphabet $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ of amino acids*
- *A set of sequences $\mathcal{S} = \Sigma^+$*
- *A metric space $(\mathcal{C}, d)$ of conformations*
- *An energy functional $G : \mathcal{S} \times \mathcal{C} \to \mathbb{R}$*
- *Environmental parameters $\mathcal{E}$*

Definition C.2 (Type-Safe Sequence). *A sequence $s \in \mathcal{S}$ is* type-safe *under environment $E \in \mathcal{E}$ if:*

$$\exists! \sigma^* \in \mathcal{C} : G_E(s, \sigma^*) = \inf_{\sigma \in \mathcal{C}} G_E(s, \sigma)$$

and σ^* is kinetically accessible from the unfolded ensemble.

Theorem C.3 (Anfinsen's Dogma as Uniqueness). *For the majority of naturally occurring sequences s under physiological conditions E_{phys}, the sequence is type-safe.*

Conjecture C.4 (Type Safety and Learnability). *Let $\mathcal{D}$ be a problem domain with input space $\mathcal{X}$ and output space $\mathcal{Y}$. Define the type-safety measure:*

$$\tau(\mathcal{D}) = \mathbb{E}_{x \sim \mathcal{X}} \left[\frac{1}{H(Y|X = x) + 1} \right]$$

where $H(Y|X = x)$ is the conditional entropy of valid outputs given input x.

Then the sample complexity of learning $\mathcal{D}$ to accuracy ϵ is:

$$N(\epsilon, \mathcal{D}) = O\left(\frac{poly(1/\epsilon)}{\tau(\mathcal{D})} \right)$$