Transdisciplinary Parsing Framework: Integrating Protein Synthesis, Flat Syntax, and Universal Dependencies

Executive Summary

This document presents a comprehensive transdisciplinary framework that unifies three distinct perspectives on linguistic structure:

1. **Biological Parallel**: The protein synthesis pathway (Transcription → Translation → Folding)
2. **Linguistic Theory**: Croft's flat-syntax three-layer annotation scheme
3. **Computational Framework**: Universal Dependencies morphological features as bonding agents

The core innovation is mapping these three perspectives onto a three-stage parsing process where morphological features function analogously to chemical properties in amino acid bonding, guiding the assembly of linguistic structure through local interactions that generate global patterns.

Table of Contents

1. Three-Stage Process Mapping
2. Stage 1: Transcription (Lexical Level)
3. Stage 2: Translation (Phrasal Level)
4. Stage 3: Folding (Sentential Level)
5. Morphological Features as Chemical Properties
6. Feature-Driven Linking Mechanisms
7. Integration with Existing Parser
8. Cross-Linguistic Implications
9. Implementation Strategy
10. Research Questions and Future Directions

---

1. Three-Stage Process Mapping {#three-stage-mapping}

1.1 Biological-Linguistic-Computational Correspondence

Biological Stage	Linguistic Stage (Croft)	Parser Stage	Primary Operations	Output
DNA → mRNA (Transcription)	**Phrasal Level**	**Transcription**	Type resolution, MWE assembly	Lexical units with types
mRNA → Amino Acids (Translation)	**Clausal Level**	**Translation**	Phrase construction, local dependencies	Phrasal constituents
Amino Acids → 3D Protein (Folding)	**Sentential Level**	**Folding**	Phrase integration, global structure	Complete parse graph

1.2 Conceptual Framework

```
INPUT: Linear word sequence
  ↓
STAGE 1: TRANSCRIPTION (Lexical Assembly)
- Resolve ambiguous word types (E/V/A/F classification)
- Assemble multi-word expressions (MWE)
- Apply morphological feature extraction
- Output: Stable lexical units with features
  ↓
STAGE 2: TRANSLATION (Phrasal Construction)
- Build local phrasal constituents
- Establish intra-phrase dependencies
- Feature-driven agreement and government
- Output: Separate phrasal structures
  ↓
STAGE 3: FOLDING (Sentential Integration)
- Integrate phrases into sentence structure
- Establish long-distance dependencies
- Resolve crossing edges and non-projectivity
- Output: Complete dependency graph
```

1.3 Information Flow Parallels

Biological:

```
Genetic Code (DNA) → Intermediate Message (RNA) → Functional Structure (Protein)
Information preserved but transformed at each stage
```

Linguistic:

Surface Forms (Words) → Intermediate Units (Phrases) → Semantic Structure (Parse Graph)
Meaning preserved but representation transformed at each stage

---

2. Stage 1: Transcription (Lexical Level) {#stage-1-transcription}

2.1 Biological Parallel: DNA to mRNA

In Biology:

- DNA double helix unwinds
- RNA polymerase reads template strand
- Creates complementary mRNA sequence
- Resolves potential ambiguity (multiple reading frames)
- Produces stable intermediate for translation

In Parsing:

- Linear word sequence accessed
- Type classifier reads surface forms + morphological features
- Creates typed lexical units with feature bundles
- Resolves word category ambiguity (bank_1 vs bank_2)
- Produces stable intermediate for phrase building

2.2 Croft's Phrasal Level Mapping

Croft's phrasal level consists of:

- **Words** as basic units
- **Boundary marker**: space (separates phrasal CEs)
- **MWE marker**: ^ (joins fixed multiword expressions)
- **Construction Element Labels**: Head, Mod, Adm, Adp, Lnk, Clf, Idx, Conj

Parser Operations at Transcription Stage:

2.2.1 Type Classification with Morphological Features

python

```
Input: "tomei café da manhã"
    (I-had coffee of-the morning)

Step 1: Extract UD features for each word

tomei:  VERB  | Mood=Ind|Number=Sing|Person=1|Tense=Past|VerbForm=Fin
café:   NOUN  | Gender=Masc|Number=Sing
da:     ADP   | (contraction of de+a)
manhã:  NOUN  | Gender=Fem|Number=Sing

Step 2: Map to parser types (E/V/A/F)

tomei  → V (eventive) [VerbForm=Fin → predicative potential]
café   → E (entity)   [NOUN → referential]
da     → F (function) [ADP → grammatical]
manhã  → E (entity)   [NOUN → referential]
```

2.2.2 MWE Assembly with Prefix Hierarchy

Biological analogy: Like codons (3-nucleotide units) encoding amino acids, MWEs are multi-word units encoding single concepts.

```
MWE Definition: "café^da^manhã" (breakfast)
- Type: E (entity)
- Components: [café, da, manhã]
- Threshold: 3 (requires all components)

Prefix Hierarchy Generation:
1. café          → activation 1/3, threshold 1
2. café da       → activation 2/3, threshold 2
3. café da manhã → activation 3/3, threshold 3 [STABLE]

Biological Parallel:
Just as secondary structures (α-helices) form when enough
amino acids with proper properties accumulate, MWEs form
when all components with proper features are present.
```

2.3 Morphological Features in Transcription

Role of Features in Stage 1:

1. **Type Disambiguation**

"tomei" could be:
- NOUN (a type of cloth) → E
- VERB (I took/drank) → V

Disambiguating features:
VerbForm=Fin → definitely VERB → V
Mood=Ind, Tense=Past, Person=1 → confirms verbal nature

2. **MWE Component Validation**

For MWE "café da manhã":

Component features must match expectations:
- café: Number=Sing (not Plur: *cafés da manhã)
- da: correct preposition+article form
- manhã: Number=Sing (not Plur: *café da manhãs)

Feature mismatches prevent MWE activation:
Like wrong amino acids preventing secondary structure formation

3. **Lexical Feature Marking**

Fixed lexical features (like amino acid properties):
- Gender: café = Masc, manhã = Fem
- Inherent animacy, noun class, etc.

These features persist through all stages,
influencing later linking operations.

2.4 Transcription Stage Output

Result: Stable lexical units with complete feature bundles

```
Lexical Units after Transcription:

[1] tomei
   Type: V
   Features: Mood=Ind|Number=Sing|Person=1|Tense=Past|VerbForm=Fin
   Status: WORD_NODE (threshold=1, activation=1)

[2] café_da_manhã
   Type: E
   Features: Gender=Masc|Number=Sing|Definite=Def
   Status: MWE_NODE (threshold=3, activation=3) ✓ AGGREGATED
   Components: [café, da, manhã] → transferred links
```

Quality Control (like RNA quality control):

- Sub-threshold MWE prefixes garbage collected
- Malformed feature bundles rejected
- Only stable units proceed to Translation stage

---

3. Stage 2: Translation (Phrasal Level) {#stage-2-translation}

3.1 Biological Parallel: mRNA to Amino Acid Chain

In Biology:

- Ribosome reads mRNA codons sequentially
- tRNA brings amino acids matching codons
- Peptide bonds form between adjacent amino acids
- Creates linear polypeptide chain
- Local properties (hydrophobic/hydrophilic) influence interactions

In Parsing:

- Parser reads lexical units sequentially
- Focus queue manages active prediction sites
- Links form between units when predictions match
- Creates phrasal dependencies
- Morphological features (Case/Gender/Number) influence linking

3.2 Croft's Clausal Level Mapping

Croft's clausal level consists of:

- **Phrases** as basic units
- **Boundary marker**: (+) (separates clausal CEs)
- **Construction Element Labels**: Pred, Arg, CPP, Gen, FPM, Conj

Parser Operations at Translation Stage:

3.2.1 Phrasal Constituent Assembly

```
Input: Lexical units from Transcription
    [tomei:V] [café_da_manhã:E]

Step 1: Predict phrase structure
tomei (V type) predicts:
- Subject (Arg) - implicit in Portuguese (pro-drop)
- Object (Arg) - expects entity

Step 2: Match predictions with following units
café_da_manhã (E type) matches Object prediction:
- Type compatibility: E can fill object slot
- Feature checking: no Case conflict (Portuguese has minimal case)
- Link formation: tomei → café_da_manhã [OBJ relation]

Result: [VP: tomei + café_da_manhã]
```

3.2.2 Feature-Driven Agreement

Biological analogy: Like amino acids forming peptide bonds through complementary chemical properties, words link through complementary morphological features.

```
Example: "la casa grande" (Spanish: the big house)

Lexical units with features:
la:     DET  | Definite=Def|Gender=Fem|Number=Sing
casa:   NOUN | Gender=Fem|Number=Sing
grande: ADJ  | Number=Sing

Agreement checking:
1. la → casa:
  - Gender: Fem = Fem ✓
  - Number: Sing = Sing ✓
  - Link forms: la --[det]--> casa

2. grande → casa:
  - Number: Sing = Sing ✓
  - Gender agreement (adjectives can be underspecified)
  - Link forms: grande --[amod]--> casa

Result: [NP: la + casa + grande]
      Unified features: Gender=Fem|Number=Sing|Definite=Def
```

Chemical Property Parallel:

Amino Acid Property	Morphological Feature	Linking Effect
Hydrophobic	Case=Nom	Attracts Subject position
Hydrophilic	Case=Acc	Attracts Object position
Charged (positive)	Gender=Masc	Must agree with Masc head
Charged (negative)	Gender=Fem	Must agree with Fem head
Hydrogen bond donor	Number=Sing	Must agree with Sing head
Hydrogen bond acceptor	Number=Plur	Must agree with Plur head

3.3 Local vs. Non-Local Phrase Types

Croft's Clausal CE Types and Their Phrase Character:

1. **Pred (Predicate)**: Core verbal phrase
   - Usually single word or complex predicate (CPP + Pred)
   - Local assembly from auxiliary + main verb
2. **Arg (Argument)**: Nominal phrases
   - Can be distant from Pred (flexible word order)
   - Feature agreement may be long-distance

3. **Gen (Genitive)**: Possessive modifier phrases
    - Often interrupting (center-embedded)
    - Croft uses {} markers: "the tree {of the house}"
4. **FPM (Flagged Phrase Modifier)**: Adjunct phrases
    - Adpositional phrases
    - Local assembly: preposition + nominal

3.4 Feature-Driven Phrase Boundary Detection

```
Features signal phrase boundaries (like codon boundaries):

Example: "tomei café da manhã cedo"
         (I-had coffee of-the morning early)

Phrase boundary detection:

1. [tomei:V Mood=Ind|VerbForm=Fin]
   → Finite verb signals Pred boundary
   → New phrase begins here

2. [café_da_manhã:E Gender=Masc|Number=Sing]
   → Nominal signals Arg boundary
   → MWE already aggregated at Transcription

3. [cedo:ADV]
   → Adverb signals modifier boundary
   → Could be FPM or Arg-internal depending on scope

Translation stage outputs:
- [Pred: tomei]
- [Arg: café_da_manhã]
- [FPM: cedo]
```

---

4. Stage 3: Folding (Sentential Level) {#stage-3-folding}

4.1 Biological Parallel: Polypeptide to 3D Protein

In Biology:

- Linear amino acid chain folds into 3D structure
- Hydrophobic collapse: nonpolar residues cluster in core

- Disulfide bridges: long-distance covalent bonds
- Tertiary structure: complete functional fold
- Energy landscape guides folding pathway

In Parsing:

- Separate phrases integrate into sentence structure
- Thematic role assignment: arguments cluster around predicate
- Long-distance dependencies: relative clauses, wh-movement
- Complete parse graph: functional semantic structure
- Activation landscape guides parse pathway

4.2 Croft's Sentential Level Mapping

Croft's sentential level consists of:

- **Clauses** as basic units
- **Boundary marker**: (#) (separates sentential CEs)
- **Construction Element Labels**: Main, Adv, Rel, Comp, Dtch, Int

Parser Operations at Folding Stage:

4.2.1 Clause Integration

Example: "O menino que eu vi chegou cedo"
(The boy that I saw arrived early)

Phrases from Translation stage:
[Arg: o menino]
[Rel: que eu vi] (relative clause)
[Pred: chegou]
[FPM: cedo]

Folding operations:

1. Identify Main clause backbone:
- Main Pred: chegou
- Main Arg (subject): o menino

2. Attach relative clause (Rel):
- que → menino (relative pronoun links to head)
- Creates long-distance dependency
- CROSSING EDGE (violates projectivity)

3. Attach temporal modifier:
- cedo → chegou

4. Internal structure of Rel clause:
- eu (implicit subject) → vi (predicate)
- que functions as object of vi

Biological Parallel:

Just as disulfide bridges (Cys-Cys bonds) connect distant amino acids in the linear sequence, relative clauses create long-distance dependencies that connect distant phrases.

Protein:
Sequence: ...Cys_{15}...Ala...Gly...Val...Cys_{78}...
3D: Cys_{15}—S—S—Cys_{78} (disulfide bridge)

Parse:
Linear: O_1 $menino_2$ que_3 eu_4 vi_5 $chegou_6$
Graph: $menino_2$ ←(subject)← $chegou_6$ (crosses 3,4,5)
$menino_2$ ←(rel)← que_3
que_3 ←(obj)← vi_5

4.2.2 Non-Projective Structures

Croft's flat-syntax representation:

```
O menino {que + eu + vi} + chegou + cedo .
Main:    Arg              Pred    FPM
```

The {} marks interruption - the relative clause is embedded within the main clause's subject NP.

Parser graph representation:

```
    chegou (root)
     /   \
  menino   cedo
    |
   que
  /  \
 eu   vi
```

Feature-Driven Integration:

```
Agreement features propagate through long-distance links:

menino: Gender=Masc|Number=Sing
  ↓ (relative pronoun agreement)
que: matches menino features
  ↓ (object of verb)
vi: must have compatible argument structure

Feature mismatch would prevent link formation,
like incompatible amino acids preventing folding.
```

4.3 Energy Landscape and Parse Selection

Biological Folding Energy:

```
Energy = E(local_interactions) + E(long_range_interactions) + E(solvent)

Protein seeks minimum energy configuration
Multiple intermediates possible
Chaperones guide folding pathway
```

Parse Graph Optimization:

```
Parse_Score = Score(local_links) + Score(long_dist_links) + Score(MWE_stability)

Parser seeks maximum coherence configuration
Multiple parses possible (ambiguity)
Context guides parse selection (like chaperones)
```

Activation-Based Selection:

```
Competing structures at Folding stage:

Option 1: "café da manhã" as MWE (breakfast)
  - High activation (3/3, fully assembled)
  - Stable from Transcription stage
  - Preferred (lower energy)

Option 2: "café" as separate entity
  - Lower activation (interrupted at Translation)
  - Would require additional links
  - Higher energy (more unstable)

Garbage collection removes Option 2 (sub-threshold nodes)
Like misfolded proteins being degraded
```

4.4 Quaternary Structure (Discourse Level)

Future extension: Multiple sentences forming discourse relations

```
Biological: Multiple protein chains → protein complex
Linguistic: Multiple sentences → discourse structure

"Tomei café da manhã. Depois fui trabalhar."
(I had breakfast. Then I went to work.)

Discourse relation: SEQUENCE
- Temporal ordering
- Causal implication (breakfast → energy → work)
- Cross-sentence anaphora possible
```

---

5. Morphological Features as Chemical Properties {#morphological-features}

5.1 Core Principle: Features as Valency

Just as amino acids have chemical properties that determine bonding patterns, morphological features determine

linking patterns in syntax.

5.2 Universal Dependencies Feature Categories

Based on UD documentation, features fall into several functional classes:

5.2.1 Nominal Features (like amino acid side chains)

UD Feature	Chemical Analog	Linking Function	Example
Case	Charge (+/-)	Determines grammatical function	Nom → Subject, Acc → Object
Gender	Polarity	Agreement matching	Masc agrees with Masc
Number	Valency	Agreement matching	Sing ↔ Sing, Plur ↔ Plur
Definite	Hydrophobicity	Information structure	Def requires previous mention
Animacy	Reactivity	Selectional restrictions	Human subjects preferred for some verbs

5.2.2 Verbal Features (like catalytic sites)

UD Feature	Chemical Analog	Linking Function	Example
VerbForm	Catalytic activity	Clause type	Fin → main clause, Inf → subordinate
Mood	Reaction conditions	Clause mood	Ind → assertion, Sub → uncertainty
Tense	Temporal properties	Time reference	Past/Present/Future
Aspect	Reaction completeness	Event structure	Perf → completed, Imp → ongoing
Voice	Reaction direction	Argument structure	Act → agent-subject, Pass → patient-subject

5.2.3 Agreement Features (like hydrogen bonding)

These features create non-covalent links (like H-bonds in proteins):

```
Spanish Example: "la casa grande"

Primary bond (covalent-like): casa is head
Secondary bonds (H-bond-like): agreement links

la:     Definite=Def | Gender=Fem | Number=Sing
        ↓ (agreement)
casa:   Gender=Fem | Number=Sing
        ↓ (agreement)
grande: Number=Sing (can agree)

Agreement features act like hydrogen bonds:
- Multiple weak interactions
- Stabilize overall structure
- Can be broken/reformed
- Distance-dependent (local agreement stronger)
```

5.3 Feature-Based Linking Rules

5.3.1 Case-Driven Linking (Ionic Interactions)

Languages with rich case systems (Latin, Russian, Finnish):

```
Latin Example: "Puella puerum amat"
               (girl-NOM boy-ACC loves)

puella: Case=Nom | Gender=Fem | Number=Sing
        → Links to amat as SUBJECT (Nom case)

puerum: Case=Acc | Gender=Masc | Number=Sing
        → Links to amat as OBJECT (Acc case)

amat:   Person=3 | Number=Sing
        → Predicate head

Case features work like ionic bonds:
- Strong attractive force (Nom → Subject position)
- Specific directionality (Acc → Object position)
- Can overcome word order effects
```

Chemical parallel:

```
Lysine (positively charged) + Glutamate (negatively charged)
→ Ionic bond (salt bridge)

Nominative (subject marker) + Finite verb (predicate)
→ Subject relation (grammatical bond)
```

5.3.2 Gender/Number Agreement (Hydrogen Bonding)

Languages with agreement systems (Spanish, German, Russian):

```
Spanish: "Los tres hermanos grandes"
        (the-MASC.PL three-MASC.PL brothers-MASC.PL big-PL)

All elements share: Gender=Masc | Number=Plur

Agreement chain:
los → hermanos:    Gender=Masc, Number=Plur ✓
tres → hermanos:   Number=Plur ✓
grandes → hermanos: Number=Plur ✓

Each agreement is like a hydrogen bond:
- Individually weak
- Collectively strong stabilization
- Multiple H-bonds hold secondary structures
- Multiple agreement features hold phrase structures
```

5.3.3 Definiteness and Information Structure (Hydrophobic Effect)

```
English: "I saw a dog. The dog was big."

First mention:  a dog   (Definite=Ind)
Second mention: the dog (Definite=Def)

Definiteness acts like hydrophobicity:
- Definite NPs cluster with previous discourse
- Indefinite NPs introduce new information
- Information packaging drives structure
- Like hydrophobic collapse drives folding
```

5.4 Feature Compatibility Matrix

Local linking preferences (like amino acid interaction potentials):

Feature Pair Compatibility:

STRONG ATTRACTION (form links readily):
- Case=Nom + VerbForm=Fin → Subject link
- Case=Acc + Transitive Verb → Object link
- Gender=Masc + Gender=Masc → Agreement link
- Number=Sing + Number=Sing → Agreement link

NEUTRAL (no preference):
- Case=Gen + Case=Nom → No direct interaction
- Gender=Masc + Gender=Fem → No agreement

REPULSION (incompatible, prevent linking):
- Case=Nom + Object position → Blocked
- Gender=Masc + Gender=Fem modifier → Blocked
- Number=Sing + Number=Plur agreement → Blocked
- VerbForm=Inf + Main clause → Blocked (needs Fin)

---

6. Feature-Driven Linking Mechanisms {#feature-driven-linking}

6.1 Linking Algorithm with Feature Checking

Extension to current parser algorithm:

python

```
def create_link_with_feature_checking(node1, node2, grammar):
    """
    Extends basic prediction-matching with feature compatibility
    """
    # Step 1: Type-level prediction (existing mechanism)
    if not grammar.predicts(node1.type, node2.type):
        return False

    # Step 2: Feature-level compatibility (NEW)
    compatibility_score = calculate_feature_compatibility(
        node1.features,
        node2.features,
        link_type=predicted_relation
    )

    if compatibility_score < THRESHOLD:
        return False  # Features block linking

    # Step 3: Create link with strength proportional to compatibility
    link = create_link(node1, node2)
    link.strength = compatibility_score
    link.features = merge_features(node1.features, node2.features)

    return True

def calculate_feature_compatibility(features1, features2, link_type):
    """
    Chemical bonding analog: calculate interaction energy
    """
    score = 1.0  # Base compatibility

    # Agreement features (like hydrogen bonds)
    for feature in ['Gender', 'Number', 'Person']:
        if feature in features1 and feature in features2:
            if features1[feature] == features2[feature]:
                score += AGREEMENT_BONUS  # Stabilizing
            else:
                score -= AGREEMENT_PENALTY  # Destabilizing

    # Case features (like ionic interactions)
    if link_type == 'subject':
        if features2.get('Case') == 'Nom':
            score += CASE_MATCH_BONUS
        else:
            score -= CASE_MISMATCH_PENALTY
```

```
    if link_type == 'object':
        if features2.get('Case') == 'Acc':
            score += CASE_MATCH_BONUS
        else:
            score -= CASE_MISMATCH_PENALTY

    # Definiteness features (like hydrophobic effect)
    if link_type == 'anaphora':
        if features2.get('Definite') == 'Def':
            score += DEFINITENESS_BONUS

    return score
```

6.2 Feature Propagation (Like Charge Distribution)

python

```
class ParseNode:
    def __init__(self, word, type, features):
        self.word = word
        self.type = type
        self.lexical_features = features  # Fixed (like amino acid properties)
        self.derived_features = {}        # Acquired through linking

    def propagate_features_from_head(self, head_node):
        """
        Agreement features propagate from head to dependent
        Like charge distribution in molecular orbitals
        """
        for feature in ['Gender', 'Number', 'Case']:
            if feature in head_node.lexical_features:
                self.derived_features[feature] = head_node.lexical_features[feature]

    def get_effective_features(self):
        """
        Combine lexical and derived features
        Lexical features override derived (like covalent > ionic bonds)
        """
        return {**self.derived_features, **self.lexical_features}
```

6.3 Multi-Level Feature Interaction

Layered features in UD (possessive example):

Hungarian: "a szomszéd házának kerítése"
(the neighbor house-POSS.3SG fence-POSS.3SG)
"the fence of the neighbor's house"

Feature layers:

kerítése: Gender=Neut | Number=Sing (agreement layer)
Gender[psor]=Fem | Number[psor]=Sing (possessor layer)

Two simultaneous feature systems:
1. Agreement with sentence structure (like backbone hydrogen bonds)
2. Possessor reference (like side chain interactions)

Parser must track both:
- Primary features: link to main structure
- Secondary features: link to sub-structure

6.4 Feature-Driven MWE Detection (Transcription Stage)

python

```
def validate_mwe_with_features(mwe_definition, word_sequence):
    """
    MWE assembly requires feature compatibility
    Like secondary structure formation requires proper amino acids
    """
    expected_features = mwe_definition.component_features
    actual_features = [word.features for word in word_sequence]

    for expected, actual in zip(expected_features, actual_features):
        # Check critical features
        for feature in ['Gender', 'Number', 'Case']:
            if feature in expected:
                if expected[feature] != actual.get(feature):
                    return False  # Feature mismatch prevents MWE

    return True  # All features compatible, MWE can form
```

Example:
MWE: "café da manhã"
Expected: café(Gender=Masc, Number=Sing) + da + manhã(Number=Sing)

Valid: "café da manhã" ✓ (features match)
Invalid: "cafés da manhã" ✗ (café: Number=Plur conflicts)
Invalid: "café das manhãs" ✗ (manhã: Number=Plur conflicts)

7. Integration with Existing Parser {#parser-integration}

7.1 Current Parser Architecture

From your implementation documentation:

```
Current Parser Flow:
1. Tokenize sentence
2. Get UD parse (lemmas + POS tags)
3. For each word:
   a. Create word node
   b. Determine type using UD POS
   c. Instantiate MWE prefixes
   d. Check MWE activation
   e. Check focus queue predictions
   f. If no match, add to focus queue
4. Garbage collection
5. Validate connectivity
```

7.2 Enhanced Three-Stage Architecture

Proposed modification:

python

```
class EnhancedParser:
    """
    Parser with explicit three-stage processing
    """

    def parse(self, sentence, grammar):
        # Get UD features for all words
        ud_parse = self.ud_parser.parse(sentence)

        # STAGE 1: TRANSCRIPTION
        lexical_units = self.transcription_stage(sentence, ud_parse, grammar)

        # STAGE 2: TRANSLATION
        phrases = self.translation_stage(lexical_units, grammar)

        # STAGE 3: FOLDING
        parse_graph = self.folding_stage(phrases, grammar)

        return parse_graph

    # ========================================
    # STAGE 1: TRANSCRIPTION (Lexical Level)
    # ========================================

    def transcription_stage(self, sentence, ud_parse, grammar):
        """
        Transcription: Resolve word types and assemble MWEs
        Input: Surface word forms + UD features
        Output: Stable lexical units with feature bundles
        """
        words = self.tokenize(sentence)
        nodes = []

        for i, word in enumerate(words):
            # Extract UD features
            features = ud_parse[i].features
            lemma = ud_parse[i].lemma
            pos = ud_parse[i].upos

            # Type classification with features
            word_type = self.classify_type_with_features(
                word, lemma, pos, features, grammar
            )

            # Create word node
            node = self.create_word_node(
```

```
            word=word,
            type=word_type,
            features=features,
            position=i
        )
        nodes.append(node)

        # MWE detection and assembly
        self.check_and_activate_mwes(node, nodes, grammar)

    # Garbage collect incomplete MWEs
    stable_units = self.garbage_collect_mwes(nodes)

    # Quality control
    self.validate_transcription_output(stable_units)

    return stable_units

def classify_type_with_features(self, word, lemma, pos, features, grammar):
    """
    Enhanced type classification using morphological features
    """
    # Base classification from POS
    base_type = self.map_pos_to_type(pos)

    # Feature-based refinement
    if pos == 'VERB':
        # VerbForm=Fin → main predicate (V type)
        # VerbForm=Inf/Part → might be nominal (A or E type)
        if features.get('VerbForm') == 'Fin':
            return 'V'
        elif features.get('VerbForm') in ['Inf', 'Ger']:
            # Gerunds/infinitives can be nominal
            return 'E'  # or 'A' depending on usage

    if pos == 'NOUN':
        # Check if verbal noun (VerbForm=Vnoun)
        if features.get('VerbForm') == 'Vnoun':
            return 'V'  # Treat as eventive
        else:
            return 'E'

    # Check lexicon for override
    lexicon_type = grammar.lookup_word_type(lemma)
    if lexicon_type:
        return lexicon_type
```

```
    return base_type

# ========================================
# STAGE 2: TRANSLATION (Phrasal Level)
# ========================================

def translation_stage(self, lexical_units, grammar):
    """
    Translation: Build phrasal constituents
    Input: Stable lexical units
    Output: Separate phrase structures
    """
    phrases = []
    focus_queue = []

    for unit in lexical_units:
        # Check for phrase boundaries
        if self.starts_new_phrase(unit, grammar):
            # Start new phrase
            phrase = Phrase(head=unit)
            phrases.append(phrase)
            focus_queue.append(unit)

        # Try linking to active phrases
        for focus_node in focus_queue:
            if self.can_link_with_features(focus_node, unit, grammar):
                # Create link with feature checking
                link = self.create_link(focus_node, unit)

                # Add to phrase
                self.add_to_phrase(unit, phrases, link)

        # If no link, start new phrase
        if not unit.linked:
            phrase = Phrase(head=unit)
            phrases.append(phrase)
            focus_queue.append(unit)

    # Identify phrase types (Pred, Arg, FPM, etc.)
    self.label_phrase_types(phrases, grammar)

    return phrases

def can_link_with_features(self, node1, node2, grammar):
    """
    Feature-aware linking at phrasal level
    """
```

```
    # Type-level prediction
    if not grammar.predicts(node1.type, node2.type):
        return False

    # Feature compatibility check
    compatibility = self.check_feature_compatibility(
        node1.features,
        node2.features,
        relation='phrasal'
    )

    return compatibility > self.FEATURE_THRESHOLD

def check_feature_compatibility(self, features1, features2, relation):
    """
    Calculate feature compatibility score
    Based on agreement, case, and other features
    """
    score = 1.0

    # Agreement features (Gender, Number, Person)
    agreement_score = self.check_agreement_features(features1, features2)
    score += agreement_score * 0.5

    # Case compatibility
    case_score = self.check_case_compatibility(features1, features2, relation)
    score += case_score * 0.3

    # Definiteness and information structure
    def_score = self.check_definiteness(features1, features2)
    score += def_score * 0.2

    return score

# ========================================
# STAGE 3: FOLDING (Sentential Level)
# ========================================

def folding_stage(self, phrases, grammar):
    """
    Folding: Integrate phrases into sentence structure
    Input: Separate phrases
    Output: Complete parse graph with long-distance links
    """
    # Identify main predicate (root of parse tree)
    main_predicate = self.find_main_predicate(phrases)
```

```
    # Build core argument structure around predicate
    parse_graph = ParseGraph(root=main_predicate)

    # Attach arguments to predicate
    self.attach_arguments(main_predicate, phrases, parse_graph)

    # Handle long-distance dependencies
    self.resolve_long_distance_deps(phrases, parse_graph, grammar)

    # Handle subordination and embedding
    self.integrate_subordinate_clauses(phrases, parse_graph, grammar)

    # Feature propagation through structure
    self.propagate_features(parse_graph)

    # Final validation
    self.validate_folding_output(parse_graph)

    return parse_graph

def resolve_long_distance_deps(self, phrases, parse_graph, grammar):
    """
    Handle relative clauses, wh-movement, etc.
    Creates non-projective links (like disulfide bridges)
    """
    # Find relative pronouns, wh-words
    relative_markers = self.find_relative_markers(phrases)

    for marker in relative_markers:
        # Find antecedent (may be distant)
        antecedent = self.find_antecedent(marker, phrases, grammar)

        if antecedent:
            # Create long-distance link
            link = self.create_link(
                marker,
                antecedent,
                relation='relative'
            )

            # Mark as non-projective if crossing
            if self.crosses_other_links(link, parse_graph):
                link.non_projective = True

            parse_graph.add_link(link)
```

7.3 Database Schema Extensions

Additional tables for feature tracking:

sql

```sql
-- Extend parser_node table
ALTER TABLE parser_node ADD COLUMN lexical_features JSONB;
ALTER TABLE parser_node ADD COLUMN derived_features JSONB;
ALTER TABLE parser_node ADD COLUMN effective_features JSONB;
ALTER TABLE parser_node ADD COLUMN stage VARCHAR(20); -- 'transcription', 'translation', 'folding'

-- Add feature compatibility tracking
CREATE TABLE parser_feature_compatibility (
    id BIGSERIAL PRIMARY KEY,
    feature1_name VARCHAR(50),
    feature1_value VARCHAR(50),
    feature2_name VARCHAR(50),
    feature2_value VARCHAR(50),
    relation_type VARCHAR(50),
    compatibility_score DECIMAL(3,2),
    notes TEXT
);

-- Track stage transitions
CREATE TABLE parser_stage_log (
    id BIGSERIAL PRIMARY KEY,
    id_parse_graph BIGINT REFERENCES parser_graph(id),
    stage VARCHAR(20),
    stage_start TIMESTAMP,
    stage_end TIMESTAMP,
    nodes_created INT,
    links_created INT,
    notes TEXT
);
```

7.4 Configuration Extensions

php

```php
// config/parser.php

return [
    // ... existing config ...

    'stages' => [
        'transcription' => [
            'enabled' => true,
            'mwe_assembly' => true,
            'feature_extraction' => true,
            'quality_control' => true,
        ],
        'translation' => [
            'enabled' => true,
            'phrase_assembly' => true,
            'feature_checking' => true,
            'local_linking' => true,
        ],
        'folding' => [
            'enabled' => true,
            'long_distance_deps' => true,
            'subordination' => true,
            'feature_propagation' => true,
        ],
    ],

    'features' => [
        'agreement' => [
            'enabled' => true,
            'features' => ['Gender', 'Number', 'Person'],
            'weight' => 0.5,
        ],
        'case' => [
            'enabled' => true,
            'features' => ['Case'],
            'weight' => 0.3,
        ],
        'definiteness' => [
            'enabled' => true,
            'features' => ['Definite'],
            'weight' => 0.2,
        ],
    ],

    'compatibility_thresholds' => [
        'strong_agreement' => 1.5,  // Features strongly support link
```

```
        'weak_agreement' => 1.0,    // Features neutral
        'disagreement' => 0.5,      // Features weakly oppose link
        'strong_disagreement' => 0.0, // Features block link
    ],
];
```

---

8. Cross-Linguistic Implications {#cross-linguistic}

8.1 Typological Variation as "Protein Families"

Different languages use different "folding strategies" just as different proteins use different folding pathways.

8.1.1 Case-Based Languages (Ionic Bonding Dominant)

Languages: Latin, Russian, Finnish, Turkish, Japanese

Feature dominance: Case marking drives linking

```
Russian Example: "Мальчик видит девочку"
                 (Boy-NOM sees girl-ACC)

Transcription: Extract rich case features
мальчик: Case=Nom | Gender=Masc | Number=Sing
видит:   Person=3 | Number=Sing
девочку: Case=Acc | Gender=Fem | Number=Sing

Translation: Case determines grammatical function
- Nom case → attracts to Subject position
- Acc case → attracts to Object position
- Word order relatively free (flexible folding pathway)

Folding: Case features dominate over position
- мальчик ← (subject) ← видит (Case=Nom determines link)
- девочку ← (object) ← видит (Case=Acc determines link)
```

Chemical parallel: Strongly charged amino acids (Lys, Arg, Glu, Asp) dominate folding through ionic interactions.

8.1.2 Agreement-Based Languages (Hydrogen Bonding Dominant)

Languages: Spanish, French, Italian, German

Feature dominance: Gender/Number agreement

Spanish Example: "Las tres hermanas grandes"
(The-FEM.PL three-FEM.PL sisters-FEM.PL big-PL)

Transcription: Extract agreement features
las: Gender=Fem | Number=Plur | Definite=Def
tres: Gender=Fem | Number=Plur
hermanas: Gender=Fem | Number=Plur
grandes: Number=Plur

Translation: Multiple agreement bonds stabilize phrase
- las → hermanas: Gender + Number agreement (2 bonds)
- tres → hermanas: Gender + Number agreement (2 bonds)
- grandes → hermanas: Number agreement (1 bond)

Total: 5 agreement bonds create stable [NP las tres hermanas grandes]

Folding: Phrase integrity maintained by agreement network

Chemical parallel: Multiple hydrogen bonds stabilize secondary structures (α-helices, β-sheets).

8.1.3 Position-Based Languages (Hydrophobic Effect Dominant)

Languages: English, Mandarin Chinese, Thai

Feature dominance: Word order and information structure

English Example: "The big dog saw a cat"

Transcription: Minimal morphological features
the: Definite=Def
big: (no features)
dog: Number=Sing
saw: Tense=Past | VerbForm=Fin
a: Definite=Ind
cat: Number=Sing

Translation: Position determines function
- Subject position: "the big dog" (no case marking)
- Predicate position: "saw" (finite verb)
- Object position: "a cat" (no case marking)

Folding: Word order + definiteness structure sentence
- Definite=Def ("the dog") → topic/given information
- Definite=Ind ("a cat") → focus/new information
- Like hydrophobic core (given) vs surface (new)

Chemical parallel: Hydrophobic collapse drives folding in globular proteins, with hydrophobic residues forming core and hydrophilic residues on surface.

8.2 Feature Inventory by Language Type

Language Type	Dominant Features	Linking Strategy	Protein Analog
Highly Inflected (Latin, Russian)	Case, Gender, Number	Feature-driven linking	Charged residues dominate
Moderately Inflected (Spanish, German)	Gender, Number, Definiteness	Agreement networks	H-bond networks stabilize
Analytic (English, Chinese)	Definiteness, Tense	Position + information structure	Hydrophobic effect drives
Agglutinative (Turkish, Finnish)	Case, Number, Person, Possessive	Layered features	Multi-level interactions
Polysynthetic (Inuktitut)	Person, Number, Mood (on verb)	Head-marking	Catalytic site complexity

8.3 Universal vs. Language-Specific Features

Universal features (present in all "proteins"):

- Some form of reference tracking (definiteness, specificity)
- Some form of predication (finite verbs, TAM marking)
- Some form of modification (adjectives, adverbs)

Language-specific features (unique "amino acids"):

- Evidentiality (Turkish, Quechua): marks information source
- Classifiers (Mandarin, Japanese): categorize nouns
- Switch-reference (Papuan languages): tracks subject continuity
- Noun classes (Bantu languages): elaborate gender system

Parser must adapt:

```
python
```

```
class LanguageSpecificFeatureHandler:
    def __init__(self, language):
        self.language = language
        self.feature_set = self.load_language_features(language)

    def extract_features(self, ud_parse):
        # Extract universal features
        features = self.extract_universal_features(ud_parse)

        # Add language-specific features
        if self.language == 'Turkish':
            features['Evident'] = self.extract_evidentiality(ud_parse)
        elif self.language == 'Mandarin':
            features['Classifier'] = self.extract_classifier(ud_parse)
        elif self.language == 'Swahili':
            features['NounClass'] = self.extract_noun_class(ud_parse)

        return features
```

---

9. Implementation Strategy {#implementation}

9.1 Phased Implementation Approach

Phase 1: Core Three-Stage Architecture (2-3 weeks)

Objectives:

- Refactor current parser into explicit stages
- Implement stage boundaries and intermediate outputs
- Add logging for stage transitions

Deliverables:

```
1. New service classes:
   - TranscriptionService.php
   - TranslationService.php
   - FoldingService.php

2. Modified main parser to orchestrate stages:
   - ParserService.php (updated)

3. Database schema updates:
   - Stage tracking tables
   - Intermediate representation storage

4. Configuration updates:
   - Stage enable/disable flags
   - Stage-specific parameters
```

Phase 2: Feature Extraction and Storage (2 weeks)

Objectives:

- Extract full UD feature sets
- Store features in database
- Implement feature access methods

Deliverables:

```
1. Enhanced UD parser integration:
   - UDParserService.php (updated)
   - Complete feature extraction

2. Feature storage:
   - JSONB columns in parser_node table
   - Feature indexing for queries

3. Feature data classes:
   - FeatureBundle.php
   - FeatureCompatibility.php
```

Phase 3: Feature-Driven Linking (3-4 weeks)

Objectives:

- Implement feature compatibility checking
- Add agreement, case, definiteness handling

- Test with multiple languages

Deliverables:

```
1. Feature compatibility system:
   - FeatureCompatibilityService.php
   - Compatibility scoring
   - Language-specific handlers

2. Enhanced linking algorithm:
   - Feature-aware prediction matching
   - Compatibility thresholds

3. Test suites:
   - Spanish (agreement-heavy)
   - Russian (case-heavy)
   - English (position-heavy)
```

Phase 4: Advanced Features (4-5 weeks)

Objectives:

- Long-distance dependencies
- Non-projective structures
- Feature propagation

Deliverables:

```
1. Advanced folding operations:
   - Relative clause handling
   - Subordination
   - Crossing edges

2. Feature propagation:
   - Agreement percolation
   - Layered features

3. Visualization updates:
   - Feature display on nodes
   - Non-projective edge rendering
```

9.2 Testing Strategy

Unit Tests (Pest Framework)

php

```php
// tests/Unit/Parser/TranscriptionStageTest.php

it('correctly classifies word types with UD features', function () {
    $transcription = new TranscriptionService();

    // Test verb classification
    $word = 'tomei';
    $features = ['VerbForm' => 'Fin', 'Mood' => 'Ind', 'Tense' => 'Past'];
    $type = $transcription->classifyType($word, 'VERB', $features);

    expect($type)->toBe('V');
});

it('assembles MWEs with feature validation', function () {
    $transcription = new TranscriptionService();

    // MWE: café da manhã
    $words = [
        ['word' => 'café', 'features' => ['Gender' => 'Masc', 'Number' => 'Sing']],
        ['word' => 'da', 'features' => []],
        ['word' => 'manhã', 'features' => ['Gender' => 'Fem', 'Number' => 'Sing']],
    ];

    $mwe = $transcription->assembleMWE($words, $mweDefinition);

    expect($mwe)->toBeInstanceOf(MWENode::class);
    expect($mwe->activation)->toBe(3);
});

// tests/Unit/Parser/FeatureCompatibilityTest.php

it('calculates agreement compatibility', function () {
    $compatibility = new FeatureCompatibilityService();

    $features1 = ['Gender' => 'Fem', 'Number' => 'Sing'];
    $features2 = ['Gender' => 'Fem', 'Number' => 'Sing'];

    $score = $compatibility->calculateAgreement($features1, $features2);

    expect($score)->toBeGreaterThan(1.0); // Positive score for agreement
});

it('detects agreement violations', function () {
    $compatibility = new FeatureCompatibilityService();

    $features1 = ['Gender' => 'Masc', 'Number' => 'Sing'];
```

```
    $features2 = ['Gender' => 'Fem', 'Number' => 'Plur'];

    $score = $compatibility->calculateAgreement($features1, $features2);

    expect($score)->toBeLessThan(1.0); // Negative score for disagreement
});
```

Integration Tests

```
php
```

```php
// tests/Feature/Parser/ThreeStageParsingTest.php

it('processes sentence through all three stages', function () {
    $parser = new EnhancedParser();

    $sentence = "Tomei café da manhã cedo";
    $result = $parser->parse($sentence, $grammar);

    // Verify stage completion
    expect($result->stages_completed)->toBe(['transcription', 'translation', 'folding']);

    // Verify MWE assembled at transcription
    $lexicalUnits = $result->transcription_output;
    expect($lexicalUnits)->toContain(function ($unit) {
        return $unit->word === 'café_da_manhã' && $unit->type === 'E';
    });

    // Verify phrases built at translation
    $phrases = $result->translation_output;
    expect($phrases)->toHaveCount(3); // [Pred], [Arg], [FPM]

    // Verify complete graph at folding
    $graph = $result->folding_output;
    expect($graph->nodes)->toHaveCount(3); // tomei, café_da_manhã, cedo
    expect($graph->edges)->toHaveCount(2); // tomei → café_da_manhã, tomei → cedo
});

it('handles long-distance dependencies at folding stage', function () {
    $parser = new EnhancedParser();

    $sentence = "O menino que eu vi chegou";
    $result = $parser->parse($sentence, $grammar);

    // Verify relative clause creates crossing edge
    $nonProjectiveEdges = $result->getNonProjectiveEdges();
    expect($nonProjectiveEdges)->not->toBeEmpty();

    // Verify "que" links to both "menino" and "vi"
    $queNode = $result->findNode('que');
    expect($queNode->incomingLinks)->toHaveCount(2);
});
```

Cross-Linguistic Tests

```
php
```

```
// tests/Feature/Parser/CrossLinguisticTest.php

it('handles Spanish agreement-based linking', function () {
    $parser = new EnhancedParser('es');

    $sentence = "Las tres hermanas grandes";
    $result = $parser->parse($sentence, $spanishGrammar);

    // All modifiers should link to head noun "hermanas"
    $hermanas = $result->findNode('hermanas');
    expect($hermanas->incomingLinks)->toHaveCount(3); // las, tres, grandes

    // Verify agreement features
    foreach ($hermanas->incomingLinks as $link) {
        expect($link->compatibilityScore)->toBeGreaterThan(1.0);
    }
});

it('handles Russian case-based linking', function () {
    $parser = new EnhancedParser('ru');

    $sentence = "Мальчик видит девочку";
    $result = $parser->parse($sentence, $russianGrammar);

    // Case should determine grammatical function
    $maljchik = $result->findNode('мальчик');
    $devochku = $result->findNode('девочку');
    $vidit = $result->findNode('видит');

    expect($maljchik->getLinkTo($vidit)->relation)->toBe('subject'); // Case=Nom
    expect($devochku->getLinkTo($vidit)->relation)->toBe('object');  // Case=Acc
});
```

9.3 Validation and Evaluation

Metrics for Three-Stage Processing

python

```
# Evaluation script

class ParserEvaluator:
    def evaluate_stage_outputs(self, test_corpus, parser):
        """
        Evaluate each stage separately
        """
        results = {
            'transcription': {},
            'translation': {},
            'folding': {}
        }

        for sentence in test_corpus:
            parse_result = parser.parse(sentence.text)

            # Stage 1: Transcription accuracy
            results['transcription'][sentence.id] = self.evaluate_transcription(
                expected_units=sentence.gold_lexical_units,
                actual_units=parse_result.transcription_output
            )

            # Stage 2: Translation accuracy
            results['translation'][sentence.id] = self.evaluate_translation(
                expected_phrases=sentence.gold_phrases,
                actual_phrases=parse_result.translation_output
            )

            # Stage 3: Folding accuracy
            results['folding'][sentence.id] = self.evaluate_folding(
                expected_graph=sentence.gold_parse,
                actual_graph=parse_result.folding_output
            )

        return results

    def evaluate_transcription(self, expected_units, actual_units):
        """
        Metrics:
        - MWE detection F1 score
        - Type classification accuracy
        - Feature extraction completeness
        """
        return {
            'mwe_f1': calculate_f1(expected_units.mwes, actual_units.mwes),
            'type_accuracy': calculate_accuracy(expected_units.types, actual_units.types),
```

```
        'feature_coverage': calculate_coverage(expected_units.features, actual_units.features)
    }

def evaluate_translation(self, expected_phrases, actual_phrases):
    """
    Metrics:
    - Phrase boundary F1 score
    - Intra-phrase link accuracy
    - Feature agreement accuracy
    """
    return {
        'boundary_f1': calculate_f1(expected_phrases.boundaries, actual_phrases.boundaries),
        'local_link_accuracy': calculate_accuracy(expected_phrases.links, actual_phrases.links),
        'agreement_accuracy': calculate_agreement_accuracy(expected_phrases, actual_phrases)
    }

def evaluate_folding(self, expected_graph, actual_graph):
    """
    Metrics:
    - UAS (Unlabeled Attachment Score)
    - LAS (Labeled Attachment Score)
    - Non-projective edge accuracy
    """
    return {
        'uas': calculate_uas(expected_graph, actual_graph),
        'las': calculate_las(expected_graph, actual_graph),
        'non_proj_f1': calculate_non_projective_f1(expected_graph, actual_graph)
    }
```

---

10. Research Questions and Future Directions {#future-directions}

10.1 Theoretical Questions

10.1.1 Stage Boundaries

Question: Are the three stages discrete or continuous?

Biological parallel: Co-translational folding suggests stages overlap:

- Protein folding begins before translation completes
- Secondary structures form during ribosome synthesis

Linguistic question:

- Can phrasal assembly begin before all MWEs are resolved?

- Can long-distance dependencies be predicted during phrase building?

Research approach:

```python
# Test incremental vs. batch processing

def incremental_parsing(sentence):
    """Process each word immediately through all three stages"""
    for word in sentence:
        transcription_output = transcription_stage(word)
        translation_output = translation_stage(transcription_output)
        folding_output = folding_stage(translation_output)
    return folding_output

def batch_parsing(sentence):
    """Complete each stage for entire sentence before next stage"""
    transcription_output = transcription_stage(sentence)
    translation_output = translation_stage(transcription_output)
    folding_output = folding_stage(translation_output)
    return folding_output

# Compare:
# - Processing speed
# - Ambiguity resolution effectiveness
# - Garden path effects
```

10.1.2 Feature Sufficiency

Question: Which features are necessary and sufficient for correct parsing?

Research approach:

- Ablation studies: remove feature types one at a time
- Measure impact on parsing accuracy
- Identify critical vs. redundant features

```python
```

```
features_to_test = [
    ['Case'],
    ['Gender', 'Number'],
    ['VerbForm', 'Mood', 'Tense'],
    ['Definite'],
    ['Case', 'Gender', 'Number'],  # Combinations
    # ... all subsets
]

for feature_subset in features_to_test:
    parser = EnhancedParser(enabled_features=feature_subset)
    accuracy = evaluate(parser, test_corpus)
    print(f"Features {feature_subset}: Accuracy {accuracy}")
```

Expected findings:

- Case-rich languages: Case features critical
- Agreement languages: Gender+Number critical
- Analytic languages: Definiteness+Position critical

10.1.3 Universal vs. Language-Specific Features

Question: Can we identify a minimal universal feature set?

Hypothesis: All languages need:

1. Reference tracking (definiteness, specificity, or equivalent)
2. Predication marking (finite verb identification)
3. Modification (attributive vs. predicative distinction)

Beyond that: Language-specific features optimize for particular structures

10.2 Computational Questions

10.2.1 Feature Compatibility Functions

Question: What is the optimal feature compatibility function?

Current approach: Linear combination of feature scores

```
compatibility = α·agreement + β·case + γ·definiteness
```

Alternatives to explore:

1. **Neural feature compatibility:**

```python
def neural_compatibility(features1, features2):
    # Embed features
    embedding1 = feature_encoder(features1)
    embedding2 = feature_encoder(features2)

    # Compute compatibility via neural network
    compatibility = compatibility_network([embedding1, embedding2])
    return compatibility
```

2. **Energy-based models:**

```python
def energy_based_compatibility(features1, features2):
    # Define energy function (lower = more compatible)
    energy = compute_energy(features1, features2, learned_weights)
    compatibility = exp(-energy)  # Boltzmann distribution
    return compatibility
```

3. **Attention mechanisms:**

```python
def attention_compatibility(features1, features2):
    # Learn which features to attend to
    attention_weights = attention_network(features1, features2)
    compatibility = weighted_sum(features1, features2, attention_weights)
    return compatibility
```

10.2.2 Scalability

Question: How does feature-driven parsing scale?

Concerns:

- Feature extraction overhead
- Compatibility computation costs
- Feature storage requirements

Optimization strategies:

1. **Feature caching:**

```python
class FeatureCache:
    def __init__(self):
        self.cache = {}

    def get_features(self, word, lemma, pos):
        key = (word, lemma, pos)
        if key not in self.cache:
            self.cache[key] = extract_features(word, lemma, pos)
        return self.cache[key]
```

2. **Lazy feature extraction:**

```python
# Only extract features when needed for compatibility checking
def lazy_feature_extraction(node1, node2):
    if not requires_feature_checking(node1, node2):
        return basic_compatibility(node1, node2)
    else:
        features1 = extract_full_features(node1)
        features2 = extract_full_features(node2)
        return feature_compatibility(features1, features2)
```

3. **Feature selection:**

```python
# Identify most informative features per language
critical_features = {
    'es': ['Gender', 'Number'],  # Spanish
    'ru': ['Case', 'Gender', 'Number'],  # Russian
    'en': ['Definite', 'Tense'],  # English
}

def selective_feature_extraction(word, language):
    all_features = extract_all_features(word)
    return {f: all_features[f] for f in critical_features[language]}
```

10.3 Empirical Research Questions

10.3.1 Cross-Linguistic Validation

Research plan:

1. **Implement parser for 5-10 typologically diverse languages:**
   - Indo-European: Spanish, Russian, German
   - Sino-Tibetan: Mandarin
   - Uralic: Finnish
   - Turkic: Turkish
   - Bantu: Swahili
   - Austronesian: Indonesian
2. **Compare feature usage patterns:**
   - Which features are most informative per language?
   - Are there universal feature interactions?
   - How do feature weights vary?
3. **Evaluate against language-specific parsers:**
   - Does universal framework match specialized parsers?
   - What is the cost of cross-linguistic generality?

10.3.2 Psycholinguistic Validation

Question: Does the three-stage model reflect human processing?

Research approaches:

1. **Reading time studies:**
   - Do garden path effects correlate with stage conflicts?
   - Example: "The horse raced past the barn fell"
   - Predicts difficulty at Translation stage (raced initially parsed as main verb)
2. **ERP studies:**
   - Can we detect stage boundaries in brain activity?
   - Stage 1 (Transcription): N400 for lexical access
   - Stage 2 (Translation): LAN for morphosyntactic processing
   - Stage 3 (Folding): P600 for syntactic integration
3. **Incremental processing:**
   - Compare incremental vs. batch parsing with human behavior
   - Do humans show "co-translational" parsing effects?

10.4 Extensions and Applications

10.4.1 Machine Translation

Application: Use biological parallel for better MT

```
Source language → Transcription → Translation → Folding → Target structure
                                                    ↓
Target structure → Unfolding → De-translation → De-transcription → Target language
```

Analogy: Protein homology modeling

- Align source and target "amino acid sequences"
- Transfer known "folds" from source to target
- Adjust for language-specific "mutations"

10.4.2 Language Acquisition

Application: Model child language acquisition

Stages:

1. **Transcription learning:** Word segmentation, category learning
2. **Translation learning:** Basic phrase structures, local dependencies
3. **Folding learning:** Complex sentences, long-distance dependencies

Prediction: Children acquire stages sequentially

- Stage 1: Age 1-2 (word learning)
- Stage 2: Age 2-4 (phrase structure)
- Stage 3: Age 4-7 (complex syntax)

10.4.3 Grammatical Error Correction

Application: Diagnose errors by stage

```
Error: "Las hermanos grandes" (should be "Los hermanos grandes")
Diagnosis: Translation stage error - agreement failure
        Gender=Fem (las) ≠ Gender=Masc (hermanos)

Correction strategy: Propagate correct features from head noun
```

10.4.4 Parsing Domain-Specific Languages

Extension: Apply to programming language parsing

Parallel:

- Transcription: Lexical analysis (tokenization)
- Translation: Syntactic analysis (parse tree building)
- Folding: Semantic analysis (type checking, scope resolution)

Feature analog: Type signatures act like morphological features

- Type compatibility checking
- Inference and propagation
- Constraint satisfaction

---

11. Conclusion

11.1 Summary of Contributions

This framework contributes to several fields:

Computational Linguistics:

- Explicit three-stage parsing architecture
- Feature-driven linking mechanism
- Unified treatment of MWEs and dependencies

Linguistic Theory:

- Integration of Croft's flat-syntax with dependency parsing
- Feature-based constituency/dependency hybrid
- Cross-linguistic framework grounded in morphology

Cognitive Science:

- Processing model with biological plausibility
- Testable predictions for human sentence processing
- Developmental trajectory predictions

Bioinformatics/AI:

- Demonstration of deep structural parallels
- Nature-inspired algorithm design

- Cross-domain insight generation

11.2 Key Insights

1. **Hierarchical assembly is universal:** From proteins to sentences, complex structures emerge from linear sequences through hierarchical assembly processes.
2. **Features are not decorations:** Morphological features actively drive structure formation, like chemical properties drive molecular assembly.
3. **Stages are functionally distinct:** Transcription (lexical), Translation (phrasal), and Folding (sentential) operations are separable and correspond to biological counterparts.
4. **Local rules generate global patterns:** Complex sentence structures emerge from local feature-driven linking rules without global planning.
5. **Cross-linguistic variation is systematic:** Languages differ in which features dominate (like protein families differ in folding strategies) but use the same underlying mechanism.

11.3 Future Vision

Short-term (1 year):

- Full implementation of three-stage parser
- Validation on multiple languages
- Publication of theoretical framework

Medium-term (2-3 years):

- Neural feature compatibility learning
- Large-scale cross-linguistic evaluation
- Psycholinguistic validation studies

Long-term (5+ years):

- Unified theory of hierarchical compositional systems
- Applications to other domains (music, DNA regulation, social networks)
- Integration with neurocognitive models

Ultimate goal: Demonstrate that hierarchical assembly via local activation rules represents a deep computational principle discovered independently by evolution in multiple domains—and that we can leverage this principle for artificial intelligence systems.

References

Protein Folding:

- Anfinsen, C.B. (1973). Principles that govern the folding of protein chains. Science.
- Dill, K.A. & MacCallum, J.L. (2012). The protein-folding problem, 50 years on. Science.
- Dobson, C.M. (2003). Protein folding and misfolding. Nature.

Flat Syntax:

- Croft, W. (2022). Morphosyntax: Constructions of the World's Languages. Cambridge University Press.
- Croft, W. (in preparation). Flat Syntax: A simplified annotation scheme for dependency structure.

Universal Dependencies:

- de Marneffe, M.C. et al. (2021). Universal Dependencies. Computational Linguistics, 47(2).
- Nivre, J. et al. (2020). Universal Dependencies v2: An Evergrowing Multilingual Treebank Collection. LREC.
- Zeman, D. et al. (2018). CoNLL 2018 shared task: Multilingual parsing from raw text to Universal Dependencies. CoNLL.

Universal Morphology:

- Sylak-Glassman, J. et al. (2015). A language-independent feature schema for inflectional morphology. ACL.
- Kirov, C. et al. (2018). UniMorph 2.0: Universal Morphology. LREC.

Relational Network Theory:

- Lamb, S.M. (1999). Pathways of the Brain: The Neurocognitive Basis of Language. John Benjamins.

Computational Parsing:

- Kübler, S., McDonald, R., & Nivre, J. (2009). Dependency Parsing. Morgan & Claypool.
- McDonald, R. et al. (2005). Non-projective dependency parsing using spanning tree algorithms. HLT-EMNLP.

Document Status: Research Framework v1.0
Date: December 2024
Authors: Ely (concept), Claude (documentation)
Next Steps: Implementation Phase 1 - Core Three-Stage Architecture