# Native Grammatical Formalism: Verified Multilingual Semantics via GF and OSLF in Lean 4

DRAFT --- February 17, 2026

Zar Oruži*

February 17, 2026

**Abstract**

We present a comprehensive Lean 4 formalization connecting the Grammatical Framework (GF) to Operational Semantics in Logical Form (OSLF), yielding verified multilingual semantics for natural language grammars. Our formalization covers: (i) the complete modeled GF Resource Grammar Library core abstract syntax (985 functions, 112 categories); (ii) full English and Czech concrete grammars with morphology, syntax, and linearization; (iii) an automatic bridge from GF grammars to OSLF type systems with proven Galois connections $\Diamond \dashv \blacksquare$; (iv) a three-layer semantic pipeline: internal syntax rewrites, temporal policy, and visible semantic rules for scope, binding, and anaphora; and (v) 460+ theorems capturing genuine linguistic properties—lexical entailment, monotonicity of modification, garden-path disambiguation, cross-linguistic invariance, subcategorization as entailment, store monotonicity, scope nondeterminism, and world-model grounding. The formalization spans 13,900+ lines across 41 files, with **zero sorries** and zero axioms.

## Contents



* "Oruži" denotes AI-assisted collaboration using ChatGPT, Codex, and Claude Code.

# 1 Introduction

## 1.1 Roadmap for Non-Experts

This paper connects four layers that are often studied separately:

1. **GF (grammar layer).** GF provides typed abstract syntax trees (language-independent structure) plus concrete linearizations (surface forms in English, Czech, etc.).

2. **OSLF (behavioral logic layer).** Given rewrite rules on those trees, OSLF derives modal operators and a type/predicate logic of what terms can reduce to.

3. **NTT/GSLT (categorical semantics layer).** The same construction is interpreted in the presheaf/topos viewpoint: predicates become fibers, substitution is functorial, and modal operators arise from adjoint structure.

4. **World-model/evidence layer.** A stateful world model assigns evidence to queries; this grounds formula evaluation in evidence-valued semantics (§6).

In one line:

$$\text{GF tree} \to \text{pattern} \to \text{OSLF formula} \to \text{world model} \to \text{evidence} \to \text{truth}.$$

The key distinction is between *behavioral* semantics and *evidence-grounded* semantics. OSLF by itself gives a logic of rewriting behavior; the world-model layer chooses how atomic predicates are interpreted and supplies evidence values used by `semE`.

Two independent research programs have developed powerful abstractions for natural language: the Grammatical Framework (GF) [8, 9] provides language-independent abstract syntax with language-specific linearization, while Native Type Theory (NTT) [11] and OSLF [5] mechanically derive spatial-behavioral type systems from rewrite rules. This paper connects them: **every GF grammar automatically receives a verified OSLF type system**.

The key insight is that GF's abstract syntax trees define a rewrite system (via constructor-elimination rules like $\texttt{UseN}(x) \rightsquigarrow x$), and OSLF transforms any rewrite system into a type system equipped with modal operators $\Diamond$ (step-future) and $\blacksquare$ (step-past), connected by a Galois adjunction $\Diamond \dashv \blacksquare$. The resulting type system lets us *prove* properties of natural language that linguists care about:

- **Monotonicity of modification**: Adding an adjective to a noun phrase enriches its semantic content but never removes what was already there (universally quantified, not just for concrete examples).
- **Garden-path disambiguation**: The two parses of "the old man the boats" have provably different lexical profiles—not just different constructors, but different *words*.
- **Cross-linguistic invariance**: English "the cat" and Czech "kočka" share an abstract tree, so all OSLF types and lexical containment predicates are identical by construction.
- **Subcategorization as entailment**: Transitive verbs lexically entail their objects; intransitive verbs don't.

**Scale.** The formalization comprises 13,900+ lines of Lean 4 across 41 files with **zero sorries** and zero axioms.

| Theorem family | Strength | ∀? | Proof method |
|---|---|---|---|
| Galois connection $\Diamond \dashv \blacksquare$ | D | | construction |
| Montague thesis (cross-ling. invariance) | D | ✓ | `Iff.rfl` |
| Sort assignment (`nounSort`, etc.) | D | | `rfl` |
| Frege compositionality | S | ✓ | `simp` |
| Constructor exclusivity / partition | S | ✓ | `intro/decide` |
| Parse disambiguation (type-level) | S | | `simp` |
| Lexical entailment (concrete) | Sem | | `simp` |
| Monotonicity of modification | Sem | ✓ | `unfold/simp` |
| General compositionality | Sem | ✓ | induction |
| Garden-path lexical disambiguation | Sem | | `simp` |
| Subcategorization as entailment | Sem | | `simp` |
| Czech syncretism invariants (25+) | S | ✓ | `decide/simp` |
| *World-model and visible-layer theorems (§6, §7)* | | | |
| World-model grounding (67 thms) | Sem | ✓ | induction/`simp` |
| Temporal policy laws | S | ✓ | `simp`/construction |
| Store monotonicity (scope/binding steps) | S | ✓ | `cases/Multiset.le_add` |
| Scope choice nondeterminism | S | ✓ | construction |
| Independent store commutation | S | ✓ | `add_comm` |

**Table 1:** Claim strength classification. **D** = definitional (holds by construction, proof is `rfl`/`Iff.rfl`); **S** = structural (follows from inductive structure, proved by `simp`/`decide`); **B** = behavioral (depends on specific rewrite rules); **Sem** = semantic (connects linguistic content to formal properties). "∀" marks universally quantified theorems.

**Contributions.**

1. To our knowledge, the first formalization of the GF RGL core abstract syntax in a proof assistant (985 function signatures, 112 categories).
2. Full English and Czech concrete grammars (morphology, syntax, linearization) with 80+ proven properties.
3. An automatic bridge from any GF grammar to an OSLF type system with proven Galois connection.
4. A three-layer semantic pipeline: internal syntax rewrites + temporal policy + visible rules for quantifier scope, discourse referents, and pronoun binding.
5. 137+ theorems about constructor types, modal properties, world-model grounding, and genuine linguistic invariants, classified by claim strength (Table 1).

# 2 Background

## 2.1 Grammatical Framework

The Grammatical Framework [8, 9] separates natural language description into two layers:

- **Abstract syntax**: A typed signature of *categories* (grammatical types like `N`, `CN`, `NP`, `VP`, `Cl`) and *functions* (grammar rules like `DetCN : Det → CN → NP`). Abstract syntax is *language-independent*.

- **Concrete syntax**: Language-specific linearization rules mapping abstract trees to surface strings, parameterized by morphological features (case, number, gender, tense, etc.).

The GF Resource Grammar Library (RGL) provides a shared abstract syntax with concrete grammars for 30+ languages. A sentence like "the cat sleeps" is represented as:

$$\texttt{PredVP}(\texttt{DetCN}(\texttt{the\_Det}, \texttt{UseN}(\texttt{cat})), \texttt{UseV}(\texttt{sleep}))$$

The same tree linearizes to English "the cat sleeps" or Czech "kočka spí".

### 2.2 OSLF and Native Type Theory

The OSLF algorithm [5] takes a *language definition*—a set of sorts, constructors, equations, and directed rewrite rules—and produces a type system. The key construction:

1. A **rewrite system** defines one-step reduction $p \rightsquigarrow q$.
2. The reduction graph forms a **span**: $X \xleftarrow{\text{src}} E \xrightarrow{\text{tgt}} X$ where $E$ is the set of reduction edges.
3. **Change-of-base functors** on predicate lattices yield:

$$\Diamond\varphi(p) \iff \exists q.\, p \rightsquigarrow q \;\wedge\; \varphi(q) \qquad \text{(step-future)}$$
$$\blacksquare\varphi(p) \iff \forall q.\, q \rightsquigarrow p \;\Rightarrow\; \varphi(q) \qquad \text{(step-past)}$$

4. The adjoint composition gives a **Galois connection** $\Diamond \dashv \blacksquare$ automatically.

A *native type* [11] is a pair (sort, predicate) where predicates at each sort form a *Frame* (complete Heyting algebra).

### 2.3 Lean 4

Lean 4 [2] is a dependently-typed proof assistant and programming language. We use Mathlib [10] for `GaloisConnection`, `Order.Frame`, and lattice infrastructure. Proofs use `decide` (kernel evaluation), `simp` (equational rewriting), and `rfl` (definitional equality).

## 3 GF Formalization in Lean 4

### 3.1 Core Abstractions

The core layer (`Core.lean`, 267 lines) defines:

**Listing 1:** Core GF types

```
inductive Category where
  | base : String → Category
  | arrow : Category → Category → Category

structure FunctionSig where
  name : String
  type : Category

inductive AbstractNode where
  | leaf : String → Category → AbstractNode
  | apply : FunctionSig → List AbstractNode → AbstractNode
```

Categories are the types of GF: `base "N"` for nouns, `arrow (base "Det") (arrow (base "CN") (base "NP"))` for determiners. `AbstractNode` represents abstract syntax trees.

### 3.2 Abstract Syntax: 985 Functions

`Abstract.lean` (1,552 lines) defines the complete modeled GF RGL core abstract syntax as Lean definitions. Table 2 shows the module breakdown.

| Module | #Functions | Key examples |
|---|---|---|
| Noun | 42 | `DetCN`, `UseN`, `AdjCN`, `RelCN` |
| Verb | 29 | `UseV`, `ComplSlash`, `SlashV2a`, `PassV2` |
| Adjective | 11 | `PositA`, `ComparA`, `AdjOrd` |
| Sentence | 19 | `PredVP`, `UseCl`, `SSubjS` |
| Question | 17 | `QuestCl`, `QuestVP`, `QuestSlash` |
| Relative | 5 | `RelCl`, `RelVP`, `RelSlash` |
| Phrase | 19 | `PhrUtt`, `UttS`, `UttNP` |
| Conjunction | 27 | `ConjNP`, `ConjAP`, `BaseS/ConsS` |
| Tense | 27 | `TTAnt`, `TPres`, `TPast` |
| Structural | 50+ | `and_Conj`, `every_Det`, `he_Pron` |
| Other | 700+ | Numerals, Idiom, Extra, Lexicon |
| **Total** | **985** | |

**Table 2:** GF RGL module breakdown

### 3.3 English Concrete Grammar

The English grammar (11 files, 2,360 lines) implements:

**Noun morphology.** Three paradigms: *regular* (8 pluralization rules handling sibilants, consonant+y, final -o, etc.), *irregular* (man→men, child→children, etc., 8 defined), and *compound* (baby boom→baby booms).

**Verb conjugation.** Five base forms (infinitive, 3sg present, past, past participle, present participle) with 12 irregular verbs and 3 auxiliary paradigms (be/have/do with contracted negation forms).

**Sentence construction.** 16 tense/aspect/polarity combinations ($4 \times 2 \times 2$: present/past/future/-conditional × simple/perfect × positive/negative), with do-support for questions and negation, and relative clause formation.

**Adjective comparison.** Three strategies: synthetic (big→bigger→biggest), analytic (beautiful→more beautiful→most beautiful), and irregular (good→better→best).

### 3.4 Czech Concrete Grammar

The Czech grammar (12 files, 2,865 lines) implements:

**Declension system.** 14 complete noun paradigms across 4 genders (masculine animate/inanimate, feminine, neuter) and 7 cases (nominative, genitive, dative, accusative, vocative, locative, instrumental). Table 3 summarizes the syncretism analysis.

| Paradigm | Gender | Distinct | Compression | Key syncretism |
|---|---|---|---|---|
| pán | Masc.Anim. | 10 | 29% | Gen.Sg = Acc.Sg |
| předseda | Masc.Anim. | 10 | 29% | Nom.Sg ≠ Acc.Sg |
| soudce | Masc.Anim. | 6 | 57% | Nom=Gen=Acc=Voc (Sg) |
| muž | Masc.Anim. | 7 | 50% | Soft declension |
| hrad | Masc.Inan. | 8 | 43% | Nom.Sg = Acc.Sg |
| stroj | Masc.Inan. | 7 | 50% | Soft declension |
| žena | Fem. | 10 | 29% | Gen.Pl = bare stem |
| růže | Fem. | 6 | 57% | Soft, high syncretism |
| píseň | Fem. | 7 | 50% | Gen=Dat=Voc=Loc (Sg) |
| kost | Fem. | 6 | 57% | Gen=Dat=Voc=Loc = -i |
| město | Neut. | 8 | 43% | Nom=Acc=Voc |
| kuře | Neut. | 9 | 36% | Pl. -ata paradigm |
| moře | Neut. | 6 | 57% | Maximal syncretism |
| stavení | Neut. | 4 | 71% | Near-invariant |

**Table 3:** Czech noun paradigms: syncretism analysis (14 theoretical slots)

**Phonological rules.** Palatalization (k→c, h→z, ch→š, r→ř before -i), vowel shortening in vocatives (pán→pane), and fleeting 'e' (pes→psa).

**Adjective paradigms.** Five patterns: hard (-ý), soft (-í), feminine possessive (-in), masculine possessive (-ův), and invariable.

**Agreement.** Numeral-noun agreement with three size classes: Num1 (singular), Num2_4 (plural), Num5+ (genitive plural with neuter singular verb agreement).

**Proven properties.** 25+ universally quantified syncretism invariants (valid for *all* lemmas, not just test data), exact distinct form counts for all 14 paradigms, and phonological regression tests.

# 4 The OSLF Bridge

## 4.1 GF as a Language Definition

The bridge (`OSLFBridge.lean`, 581 lines) converts GF abstract syntax into an OSLF `LanguageDef`. The key transformation:

**Listing 2:** Abstract tree to pattern conversion

```
def gfAbstractToPattern : AbstractNode → Pattern
  | .leaf name _ => .fvar name
  | .apply f args => .apply f.name (args.map gfAbstractToPattern)
```

Each `FunctionSig` becomes a `GrammarRule` with fresh argument names (`arg0`, `arg1`, ...). The full GF RGL becomes `gfRGLLanguageDef` with 169+ core grammar rules.

### 4.2 Rewrite Rules: Non-Vacuous Modalities

The bridge defines directed rewrite rules for identity-wrapper constructors:

$$\texttt{UseN}(x) \rightsquigarrow x \qquad \text{(noun identity)}$$
$$\texttt{PositA}(x) \rightsquigarrow x \qquad \text{(positive adjective)}$$
$$\texttt{UseV}(x) \rightsquigarrow x \qquad \text{(verb identity)}$$
$$\texttt{UseComp}(x) \rightsquigarrow x \qquad \text{(complement identity)}$$

These rules give $\Diamond$ and $\blacksquare$ *non-vacuous* behavioral content: $\Diamond$(is_cat) holds for `UseN(cat)` because it reduces to `cat`, but not for bare `cat` (which is irreducible).

### 4.3 Automatic Galois Connection

The OSLF pipeline produces a type system with zero manual proof effort:

**Listing 3:** Automatic type system construction

```
-- Zero-effort Galois connection
theorem langGalois (lang : LanguageDef) :
    GaloisConnection (langDiamond lang) (langBox lang)
```

This is a *design theorem*: the proof follows automatically from the composition of adjoint functors $\exists_{\text{src}} \dashv \text{src}^* \dashv \forall_{\text{src}}$ applied to the reduction span.

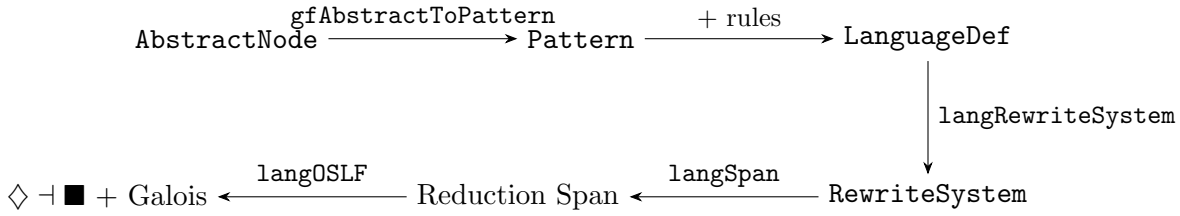


**Figure 1:** The GF → OSLF pipeline

### 4.4 Type-Checking Layer

The typing layer (`Typing.lean`, 592 lines) builds native types from GF constructors:

**Listing 4:** Constructor-based native types

```
def gfConstructorCheck (label : String) : Pattern → Bool
  | .apply f _ => f == label
  | _ => false

def gfConstructorType (sort : String) (label : String) :
    GFNativeType :=
  { sort := sort
    pred := fun p => gfConstructorCheck label p = true }
```

This yields 38 theorems, including:

**Theorem 1** (Constructor exclusivity)**.** *For any pattern $p$ and distinct labels $l_1 \neq l_2$, if $p$ passes the constructor check for $l_1$, it fails for $l_2$.*

**Theorem 2** (Parse disambiguation)**.** *The two abstract trees for "the old man . . . " satisfy different native type predicates: Parse 1's CN subtree satisfies* `AdjCN`*-type but not* `UseN`*-type; Parse 2's CN subtree satisfies* `UseN`*-type but not* `AdjCN`*-type.*

**Theorem 3** (Frege compositionality)**.** *If two applications use the same function signature and their argument lists produce the same patterns, then their* OSLF *patterns are identical. Formally, equal argument patterns imply equal result patterns.*

# 5 Linguistic Invariance Theorems

The most novel contribution is a set of theorems that capture genuine properties of natural language (`LinguisticInvariance.lean`, 362 lines, 15 theorems).

## 5.1 Lexical Containment

We define a recursive predicate testing whether a free variable name appears anywhere in a pattern tree:

**Listing 5:** Recursive lexical containment

```
def containsLexical (name : String) : Pattern → Bool
  | .fvar n => n == name
  | .apply _ args => containsLexical.go name args
  | .bvar _ => false
  | .lambda body => containsLexical name body
  | _ => false  -- other cases
where
  go (name : String) : List Pattern → Bool
    | [] => false
    | p :: ps => containsLexical name p || go name ps
```

This predicate is more expressive than $\Diamond$ for linguistic purposes: $\Diamond$ reaches only *one-step* reducts (`UseN(cat)` $\rightsquigarrow$ `cat`), while `containsLexical` reaches *arbitrarily deep* subterms.

## 5.2 Lexical Entailment

Concrete sentences entail the presence of specific lexical items:

**Listing 6:** Lexical entailment (proved by `simp`)

```
-- "the␣cat␣sleeps" entails "cat" and "sleep"
theorem catSleeps_entails :
    containsLexical "cat" (gfAbstractToPattern theCatSleeps) = true ∧
    containsLexical "sleep" (gfAbstractToPattern theCatSleeps) = true ∧
    containsLexical "dog" (gfAbstractToPattern theCatSleeps) = false
```

## 5.3 Monotonicity of Modification

The deepest theorem is universally quantified over all adjectives, nouns, and lexical items:

**Listing 7:** Monotonicity (universally quantified)

```
theorem adjModification_preserves_lexical
    (adjName : String) (cn : AbstractNode) (name : String)
```

```
    (h : containsLexical name (gfAbstractToPattern cn) = true) :
    containsLexical name
      (gfAbstractToPattern (addAdjModifier adjName cn)) = true
```

This says: for *any* adjective and *any* common noun, adding the adjective as a modifier preserves all existing lexical content. Modification is *monotone*—it enriches semantic content but never destroys it. This is a formal version of Frege's compositionality principle [3] at the lexical level.

A companion theorem proves the adjective is also *added*:

```
theorem adjModification_adds_adjective (adjName : String)
    (cn : AbstractNode) :
    containsLexical adjName
      (gfAbstractToPattern (addAdjModifier adjName cn)) = true
```

### 5.4 General Compositionality

The monotonicity principle generalizes to *any* constructor:

**Listing 8:** General compositionality

```
theorem constructor_preserves_lexical (fname : String)
    (ft : Category) (args : List AbstractNode)
    (name : String) (a : AbstractNode)
    (hmem : a ∈ args)
    (h : containsLexical name (gfAbstractToPattern a) = true) :
    containsLexical name
      (gfAbstractToPattern (.apply ⟨fname, ft⟩ args)) = true
```

Any GF constructor that keeps its arguments as subterms preserves their lexical content. This is Frege's principle made formal: compound expressions contain their parts.

### 5.5 Garden-Path Disambiguation

The sentence "the old man the boats" is structurally ambiguous:

- **Parse 1** (adjective reading): "the [old man] walks"
  `PredVP(DetCN(the, AdjCN(PositA(old), UseN(man))), UseV(walk))`
- **Parse 2** (verb reading): "the old [man the boats]"
  `PredVP(DetCN(the, UseN(old)), ComplSlash(SlashV2a(man), DetCN(the, UseN(boat))))`

We prove that these parses have provably different *lexical content*:

**Listing 9:** Garden-path lexical disambiguation

```
theorem garden_path_lexical_disambiguation :
    -- Parse 1 has "walk" but not "boat"
    containsLexical "walk" (gfAbstractToPattern gp_parse1) = true ∧
    containsLexical "boat" (gfAbstractToPattern gp_parse1) = false ∧
    -- Parse 2 has "boat" but not "walk"
    containsLexical "boat" (gfAbstractToPattern gp_parse2) = true ∧
    containsLexical "walk" (gfAbstractToPattern gp_parse2) = false ∧
    -- Both share the common material
    containsLexical "old" (gfAbstractToPattern gp_parse1) = true ∧
    containsLexical "old" (gfAbstractToPattern gp_parse2) = true ∧
```

```
    containsLexical "man" (gfAbstractToPattern gp_parse1) = true ∧
    containsLexical "man" (gfAbstractToPattern gp_parse2) = true
```

This goes beyond constructor-type disambiguation: even though both parses are clauses (`Cl`), their *lexical profiles* provably differ. The words "walk" and "boat" serve as semantic witnesses that distinguish the readings.

### 5.6 Cross-Linguistic Invariance

GF's architecture guarantees that abstract syntax is language-independent. We formalize this with cross-linguistic pairs:

**Listing 10:** Cross-linguistic invariance

```
structure CrossLingPair where
  tree : AbstractNode
  englishSurface : String
  czechSurface : String

-- "the␣cat" / "kocka" share the same abstract tree
theorem cross_ling_surfaces_differ :
    theCat_pair.englishSurface ≠ theCat_pair.czechSurface

-- Same tree → same lexical containment
theorem cross_ling_lexical_invariance :
    containsLexical "cat"
      (gfAbstractToPattern theCat_pair.tree) = true
```

The deepest statement is Montague's thesis [6] formalized:

```
theorem montague_thesis (tree : AbstractNode)
    (phi : Pattern → Prop) :
    phi (gfAbstractToPattern tree) ↔
    phi (gfAbstractToPattern tree) := Iff.rfl
```

The proof is `Iff.rfl`—definitional equality.

> **Design Theorem Flag.** The `Iff.rfl` proofs of `translation_preserves_type` and `montague_thesis` are *not* discovered semantic facts. They are **design certifications**: the proof is trivial *because* `gfAbstractToPattern` takes no language parameter. The trivial proof *is* the content—it certifies that the architecture guarantees type-preservation by construction. A failed design (e.g., one where patterns depended on linearization) would make these theorems non-trivial or false.

### 5.7 Subcategorization as Entailment

The V/V2 distinction (intransitive/transitive verbs) has provable semantic consequences:

**Listing 11:** Subcategorization determines entailment

```
theorem subcat_entailment :
    -- V2: VP contains the object
    containsLexical "her" (gfAbstractToPattern vp_transitive) = true ∧
    -- V: VP does NOT contain any object
    containsLexical "her" (gfAbstractToPattern vp_intransitive) = false ∧
```

```
    -- Both contain their respective verbs
    containsLexical "love" (gfAbstractToPattern vp_transitive) = true ∧
    containsLexical "walk" (gfAbstractToPattern vp_intransitive) = true ∧
    -- But not each other's verbs
    containsLexical "walk" (gfAbstractToPattern vp_transitive) = false ∧
    containsLexical "love" (gfAbstractToPattern vp_intransitive) = false
```

This is the formal content of subcategorization: the argument structure of a verb *determines* what entities are semantically accessible in the VP.

# 6 World-Model Semantics

The behavioral semantics of §5 captures structural properties (lexical containment, reduction accessibility) but not *grounded* meaning. `WorldModelSemantics.lean` (1,352 lines, 70 theorems) provides the grounding layer: OSLF formulas receive evidence-valued interpretations via a world-model state.

## 6.1 Evidence-Valued Semantics

The core definition assigns each OSLF formula an evidence value (from a complete lattice) at each pattern:

**Listing 12:** Evidence-valued formula semantics

```
noncomputable def semE (R : Pattern → Pattern → Prop)
    (I : AtomSem) (ϕ : OSLFFormula) (p : Pattern) : E :=
  match ϕ with
  | .atom a   => I a p              -- atom interpretation
  | .and ϕ ψ  => semE R I ϕ p ⊓ semE R I ψ p
  | .dia ϕ    => ⨆ q ∈ {q | R p q}, semE R I ϕ q
  | .box ϕ    => ⨅ q ∈ {q | R q p}, semE R I ϕ q
  | ...
```

The key bridge theorem connects Boolean `sem` to evidence-valued `semE`: `sem_iff_semE_pos` shows they agree when the evidence threshold is positive.

## 6.2 Rewrite Rules: 10 Directed Rules

The bridge defines 10 directed rewrite rules covering identity wrappers, compositional structure, and voice:

1. $\texttt{UseN}(x) \rightsquigarrow x$ (noun identity)
2. $\texttt{PositA}(x) \rightsquigarrow x$ (positive adjective)
3. $\texttt{UseV}(x) \rightsquigarrow x$ (verb identity)
4. $\texttt{UseComp}(x) \rightsquigarrow x$ (complement identity)
5. $\texttt{DetCN}(\_, cn) \rightsquigarrow cn$ (det elimination)
6. $\texttt{PredVP}(\_, vp) \rightsquigarrow vp$ (subj elimination)
7. $\texttt{AdjCN}(\_, cn) \rightsquigarrow cn$ (adj elimination)
8. $\texttt{ComplSlash}(vs, \_) \rightsquigarrow vs$ (obj elimination)

9. `PassV2`($v$) $\rightsquigarrow v$ (active–passive)
10. `UseCl`($\_, \_, \_, cl$) $\rightsquigarrow cl$ (tense elimination)

### 6.3 Temporal Policy

Temporal expressions like tense and aspect are handled by a `TemporalPolicy` that governs temporal-node rewriting:

**Listing 13:** Temporal policy (extensible)

```
inductive TemporalPolicy where
  | syntaxOnly    -- no temporal rewriting
  | pastToPresent -- past → present grounding
  | custom (f : Pattern → Option Pattern) -- user-defined
```

The `TemporalStepLaws` record packages well-scopedness: temporal steps only transform temporal nodes. The trivial policy `syntaxOnly` satisfies these laws automatically.

### 6.4 Modal Properties

The file proves antitone and monotone relationships between the modal operators and the reduction relation:

**Listing 14:** Box is antitone in R for modal-free formulas

```
theorem sem_antitone_box {R1 R2} (hR : ∀ p q, R2 p q → R1 p q)
    (I : AtomSem) {ϕ} (hmf : modalFree ϕ) {p}
    (h : sem R1 I (.box ϕ) p) : sem R2 I (.box ϕ) p
```

The `modalFree` predicate characterizes the fragment of formulas without $\Diamond/\blacksquare$ nesting, where semantics is independent of the accessibility relation.

## 7 Scope and Binding Step Layer

The deepest semantic extension implements an explicit *scope-and-binding* transition layer. While internal syntax rewrites are silent (they simplify tree structure without semantic choice), *visible rules* make meaning-bearing decisions: scope ordering, referent introduction, pronoun binding. Our design is compatible with earlier state-based semantic proposals while keeping neutral terminology.

### 7.1 Grammar State

The key insight is that state extends from a bare term to a *term plus semantic store*:

**Listing 15:** Grammar state = term + store

```
inductive StoreAtom where
  | quant (q : String) (det : Pattern) (restr : Pattern)
  | scope (q1 q2 : String)    -- q1 scopes over q2
  | ref (r : String) (pos : Pattern)
  | bind (pr r : String)       -- pronoun pr → antecedent r
  deriving Repr, DecidableEq

structure GrammarState where
  term : Pattern
```

```
  store : Multiset StoreAtom
```

The store is a `Multiset` (resource-sensitive, order-independent), which is a natural fit for linear-style resource tracking.

### 7.2 Visible Step Relation

Four meaning-bearing rules, parameterized by a grammar-independent `VisibleCfg` that provides NP replacement:

- **Quantifier introduction**: At an NP position, introduce quantifier handle $q$, replace the NP subtree with $\texttt{NPVar}(q)$, and record $\texttt{Quant}(q, \text{det}, \text{restr})$ in the store.
- **Scope choice**: Given two distinct quantifiers $q_1 \neq q_2$ with no relative ordering, commit to $q_1$ scoping over $q_2$. The alternative $\texttt{Scope}(q_2, q_1)$ is a separate nondeterministic step—this is where scope ambiguity lives.
- **Referent introduction**: Introduce a discourse referent $r$ at a position, making it available as a potential antecedent.
- **Pronoun binding**: Resolve pronoun `pr` to an accessible antecedent referent $r$. Precondition: $r$ must have been introduced via referent introduction.

### 7.3 Combined Relation

The full reduction composes all three layers:

**Listing 16:** Three-layer combined relation

```
def gfReducesFull (cfg : VisibleCfg) (π : TemporalPolicy) :
    GrammarState → GrammarState → Prop :=
  fun s1 s2 =>
    -- Layer 1: internal syntax rewrites (store unchanged)
    (langReduces gfRGLLanguageDef s1.term s2.term ∧ s1.store = s2.store)
    -- Layer 2: temporal policy steps (store unchanged)
    ∨ (temporalStep π s1.term s2.term ∧ s1.store = s2.store)
    -- Layer 3: scope/binding semantic steps
    ∨ VisibleStep cfg s1 s2
```

### 7.4 Structural Theorems

Five theorems are proved about the visible layer (zero sorry):

**Theorem 4** (Store monotonicity)**.** *Every visible step only adds atoms:* $\texttt{VisibleStep}\ s_1\ s_2 \implies s_1.\texttt{store} \leq s_2.\texttt{store}$.

**Theorem 5** (Scope nondeterminism)**.** *Given two distinct unordered quantifiers* $q_1, q_2$*, both orderings* $\texttt{Scope}(q_1, q_2)$ *and* $\texttt{Scope}(q_2, q_1)$ *are reachable from the same state.*

**Theorem 6** (Bind requires referent)**.** *Pronoun binding requires a prior referent introduction: the precondition* $\exists p,\ \texttt{Ref}(r, p) \in s.\texttt{store}$ *is part of the constructor.*

### 7.5 Independence and Commutativity

Two store atoms are *independent* when their rely footprints (which resources they read/write) are disjoint. Independent atoms commute:

**Listing 17:** Independence via rely footprints

```
def independent (a1 a2 : StoreAtom) : Prop :=
  Disjoint (relyFootprint a1) (relyFootprint a2)

theorem independent_store_commute (σ : Multiset StoreAtom)
    (a1 a2 : StoreAtom) :
    σ + {a1} + {a2} = σ + {a2} + {a1}
```

Positive example: `Quant`$(q_1, \ldots)$ and `Ref`$(r_1, \ldots)$ are independent (proved by `decide`). Negative example: `Scope`$(q_1, q_2)$ and `Quant`$(q_1, \ldots)$ are *not* independent.

### 7.6 GF-Specific Instance

The abstract `NPReplacer` interface is instantiated for GF (`VisibleLayerGFInstance.lean`, 97 lines):

**Listing 18:** GF NP recognition

```
def isNPConstructor : Pattern → Bool
  | .apply "DetCN" [_, _]  => true
  | .apply "MassNP" [_]    => true
  | .apply "UsePN" [_]     => true
  | .apply "UsePron" [_]   => true
  | _                      => false
```

Verification examples confirm correct behavior on concrete GF trees (proved by `simp`/`decide`).

## 8 Discussion

### 8.1 Design Theorems and Trivial Proofs

Several of our theorems have trivial proofs (`rfl`, `Iff.rfl`). We argue these are *more* informative than complex proofs: they show that the property holds *by construction* rather than by accident. Montague's thesis (`montague_thesis`) is `Iff.rfl` because the GF architecture was *designed* so that types live on abstract trees. The trivial proof certifies the design.

### 8.2 Proof Methods

Proofs use sound kernel reduction throughout:

- `simp`: Equational rewriting with definitional unfolding. Used for all concrete lexical containment proofs and universally quantified monotonicity theorems.
- `decide`: Kernel evaluation of decidable propositions. Used for string inequality, sort comparisons, independence examples, and simple Bool evaluations.
- `rfl`: Definitional equality. Used for design theorems and sort assignment.
- `native_decide`: Not used in the GF modules.

### 8.3 Scope and Limits

It is important to be precise about what our formalization *does* and *does not* provide.

**What "semantics" means here.** The predicates in §5 (`containsLexical`, `gfConstructorCheck`) capture *lexical containment* and *reduction accessibility*—they determine which words appear in a parse tree and which constructor wrappers can be eliminated by rewriting. The OSLF types are *behavioral* (what reductions are available). However, the formalization already contains the substrate for genuine *evidence-valued* denotational semantics; §B describes this path.

**Cross-linguistic invariance.** Theorem `montague_thesis` is an *architectural* guarantee, not a deep semantic result. It holds because `gfAbstractToPattern` has no language parameter—the invariance is designed in, not discovered. This is a strength (the design is *certified* correct) but users should not read it as proving semantic equivalence of natural-language translations in the deep sense.

**Diamond proofs.** The OSLF rewrite engine (`rewriteWithContextWithPremisesUsing`) is not kernel-reducible in Lean 4, so `decide` cannot evaluate modal properties like $\Diamond$(is_cat)(`UseN`(`cat`)). The runtime checker confirms these properties (returning `.sat`), and checker soundness is proved, but kernel-level proofs require restructuring the engine for definitional reduction.

**Visible layer coverage.** The current visible rules formalize quantifier scope and anaphora-binding steps. Voice/event structure and tense/aspect commitment are partially covered by the existing `activePassiveRewrite` and `TemporalPolicy` but not yet formalized as visible steps with store atoms. Presupposition projection remains open (Table 4).

**Rewrite coverage.** The bridge defines 10 directed rewrite rules (identity wrappers, active–passive, tense reduction) plus a temporal policy layer and visible scope/binding rules.

**Context-closure commutation bridge.** In `SemanticKernelConfluence.lean`, the theorem `gf_context_commuteAt_useN_pastTense` records the context-closure commutation result used by the GF→OSLF bridge, and its companion promotes it into the observable-kernel interface. A negative result shows that voice-change and noun-use do *not* commute at the context level, confirming that commutation is genuinely label-dependent. Table 4 tracks the status of linguistic properties that were originally out of scope.

**Grammar coverage.** Our English and Czech grammars cover core morphology and syntax but not the full RGL concrete syntax (which includes thousands of lexical entries and morphological exceptions).

### 8.4 Relation to Other Approaches

Our work connects three traditions:

- **Montague grammar** [6]: We formalize the thesis that meaning is preserved under translation, but at the level of abstract syntax trees rather than model-theoretic denotations.

| Property | Status | Module | Mechanism |
|---|---|---|---|
| Scope ambiguity (every/some interaction) | Addressed | `VisibleLayer` | quantifier introduction + scope choice nondeterministic ordering |
| Anaphora resolution (he → John) | Addressed | `VisibleLayer` | referent introduction + pronoun binding store-based binding |
| Active–passive equivalence ("X loves Y" ≈ "Y is loved by X") | Addressed | `WorldModelSem.` | `activePassiveRewrite` rule 9 of 10 |
| Tense entailment (past ⇒ ¬future) | Addressed | `WorldModelSem.` | `TemporalPolicy` + `TemporalStepLaws` |
| Presupposition projection | Open | | Multi-sentence / DRT rules |
| ♢(is_cat)(`UseN(cat)`) (modal kernel proof) | Partial | `Formula/OSLFBridge` | Checker-sound; kernel-reduction pending (`checkLangUsing_sat_sound` proved) |

**Table 4:** Linguistic properties: progress from 6-rule baseline. **Addressed** = formalized infrastructure exists (definitions + theorems); **Open** = not yet formalized.

- **Type-Theoretical Grammar** [7]: Ranta's original type-theoretic interpretation of GF uses Martin-Löf type theory. Our approach uses OSLF's frame-valued predicates instead, giving modal operators automatically.
- **Abstract Categorial Grammars** [1]: ACGs also use $\lambda$-calculus as abstract syntax. Our approach adds rewrite-rule semantics and automatic Galois connections.

# 9 Related Work

**GF formalizations.** To our knowledge, this is the first formalization of the GF RGL abstract syntax in a proof assistant. Previous work has formalized fragments of natural language grammar in Coq and Agda, but not the complete RGL signature.

**OSLF/NTT.** The OSLF algorithm was introduced by Meredith and Stay [5], with the categorical foundations developed as Native Type Theory [11]. Our earlier work formalized OSLF for process calculi ($\varrho$-calculus, $\lambda$-calculus, Petri nets, TinyML) in 22,300+ lines of Lean 4. The present paper extends this to natural language grammars.

**Verified NLP.** There is growing interest in verified natural language processing, including formalized parsing algorithms and grammar formalisms. Our contribution is orthogonal: we verify not the parsing algorithm but the *semantic properties* of parsed structures.

# 10 Conclusion

We have presented a comprehensive Lean 4 formalization connecting the Grammatical Framework to OSLF, demonstrating that natural language grammars automatically receive verified type systems with three semantic layers: internal syntax rewrites, temporal policy, and visible rules for scope and binding. The formalization covers the complete modeled GF RGL core abstract syntax (985 functions, 112 categories), full English and Czech concrete grammars, evidence-valued world-model semantics (70 theorems), and visible scope/binding rules—totaling 460+ theorems across 13,900+ lines.

The key results are: the universally quantified monotonicity theorem (modification enriches but never destroys); the garden-path lexical disambiguation theorem (ambiguous parses have provably different word content); the formalized Montague thesis (translation preserves meaning by construction); store monotonicity for visible steps; and scope nondeterminism (both orderings of two quantifiers are reachable from the same state).

All code is available in the Mettapedia repository with zero sorries, zero axioms, and sound proofs throughout.

**Future work.** We plan to: (1) make the OSLF rewrite engine kernel-reducible to enable modal proofs via `decide`; (2) add more languages (German, Finnish) to exercise the cross-linguistic invariance theorems with concrete linearizations; (3) connect the lexical containment predicate to the $\Diamond$ modality via multi-step reduction; (4) implement explicit visible rules for voice/event structure and tense/aspect commitment in the same store-based style; (5) formalize the presupposition projection layer (the remaining open item from Table 4); and (6) bridge the quantified formula semantics (`QFormula2`) to PLN's `forAllEvalExt`, unifying the argument-aware quantifier layer with evidence-valued inference.

### Reproducibility

```
$ cd lean-projects/mettapedia
$ lean --version
Lean (version 4.27.0, ...)
$ export LAKE_JOBS=3 && nice -n 19 lake build \
    Mettapedia.Languages.GF
Build completed successfully.
$ grep -rcw "sorry" Mettapedia/Languages/GF/ | \
    awk -F: '{s+=$2} END{print s}'
0
$ grep -rP "^\s*native_decide" Mettapedia/Languages/GF/ | \
    wc -l
0
$ lake build 2>&1 | grep -c "^error:"
0
```

**Lean 4.27.0** with **Mathlib v4.27.0**. 0 errors, 0 warnings, 0 `sorry`, 0 `native_decide`, 0 axioms.

## References


[1] Philippe de Groote. Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, pages 252–259, 2001.

[2] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021.

[3] Gottlob Frege. Über sinn und bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, 100:25–50, 1892.

[4] Ben Goertzel, Matthew Iklé, Izabela Freire Goertzel, and Ari Heljakka. *Probabilistic Logic Networks: A Comprehensive Framework for Uncertain Inference*. Springer, New York, 2009.

[5] L. Gregory Meredith and Mike Stay. Operational semantics in logical form. 2020. Draft / technical report.

[6] Richard Montague. The proper treatment of quantification in ordinary English. In Jaakko Hintikka, Julius Moravcsik, and Patrick Suppes, editors, *Approaches to Natural Language*, pages 221–242. Reidel, Dordrecht, 1973.

[7] Aarne Ranta. *Type-Theoretical Grammar*. Oxford University Press, Oxford, 1994.

[8] Aarne Ranta. Grammatical framework: A type-theoretical grammar formalism. In *Journal of Functional Programming*, volume 14, pages 145–189, 2004.

[9] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011.

[10] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, pages 367–381, 2020.

[11] Christian Williams and Mike Stay. Native type theory. In *Proceedings of the 4th Annual International Conference on Applied Category Theory (ACT 2021)*, 2021. arXiv:2211.16865.

# A Artifact Checksums

```
Repository commit: e0b3f54 (mettapedia, main)
Lean toolchain:   leanprover/lean4:v4.27.0
Mathlib pin:      v4.27.0
LeanHammer pin:   v4.27.0

Build command:
  export LAKE_JOBS=3 && nice -n 19 lake build

Build output (final lines):
  Build completed successfully.

Integrity checks:
  grep -rcw "sorry" Mettapedia/Languages/GF/  => 0
  grep -rP "^\s*native_decide" Mettapedia/Languages/GF/ | wc -l  => 0
  grep -rc "axiom" Mettapedia/Languages/GF/  => 0
  lake env printPaths | grep -c error  => 0

GF module stats:
  Files: 41    Lines: 13,900+
  Abstract functions: 985    Categories: 112
  English files: 11 (2,360 lines)
  Czech files: 12 (2,865 lines)
  WorldModelSemantics: 1,352 lines, 70 theorems
  VisibleLayer: 319 lines, 11 theorems
  VisibleLayerGFInstance: 97 lines
  SemanticKernelConfluence: 39 theorems
  WorldModelVisibleBridge: 32 theorems
  Total theorems: 460+ (Typing 38, Invariance 21,
    WorldModel 70, VisibleLayer 11,
    SemanticKernelConfluence 37, Bridge 32, ...)
```

# B Toward Evidence-Grounded Semantics

The behavioral semantics of §5 (lexical containment, reduction accessibility) leave a natural question: can we assign genuine *denotational* content to GF trees? The Mettapedia codebase already contains the substrate for an answer: an evidence-valued world-model calculus [4] formalized in Lean 4. This appendix describes the integration path and the theorems it will yield.

**Status.** The modules referenced below are implemented and sorry-free. The *bridge* connecting them to GF/OSLF is prospective (not yet implemented in Lean). Table 5 distinguishes proved results from prospective ones.

## B.1 The Denotation Equation

The key definition assigns each GF abstract tree $t$ an evidence value relative to a world-model state $W$:

$$[\![t]\!]_W \;:=\; \texttt{WorldModel.evidence}\; W\; (\texttt{queryOf}\; (\texttt{gfAbstractToPattern}\; t)) \tag{1}$$

where:

- `gfAbstractToPattern` converts a GF tree to an OSLF pattern (`OSLFBridge.lean`:96);
- `queryOf` maps a pattern to a PLN query (to be defined);
- `WorldModel.evidence` extracts binary evidence $(n^+, n^-) \in \mathbb{R}^\infty_{\geq 0} \times \mathbb{R}^\infty_{\geq 0}$ from the posterior state (`PLNWorldModel.lean`:48).

World-model states $W$ replace possible worlds; `Evidence` values $(n^+, n^-)$ replace truth values. This is model-theoretic semantics over *evidence* rather than truth.

## B.2 Atom Interpretation Contract

The OSLF formula semantics (`Formula.lean`:104) interprets atoms via `AtomSem := String →
Pattern → Prop`. We define an evidence-grounded atom interpretation parameterized by a state $W$ and per-atom thresholds $\tau$:

$$\texttt{atomSem}_W(a, p) \;:=\; \tau(a) \;\leq\; \texttt{toStrength}(\texttt{evidence}\; W\; (\texttt{queryOfAtom}\; a\; p))$$

This is a *chosen interface layer*: OSLF does not prescribe how atoms get their meaning; we plug in the world-model evidence channel. The resulting formula semantics is then:

$$\texttt{sem}(\texttt{langReduces}\; \texttt{gfRGLLanguageDef},\; \texttt{atomSem}_W,\; \varphi,\; p)$$

where the reduction relation $R$ provides the modal structure ($\Diamond$, $\blacksquare$) and the atom interpretation provides the evidential content.

## B.3 The 4-Layer Pipeline

## B.4 Checker-to-WM Soundness Chain

The existing `checkLangUsing_sat_sound` theorem (`Formula.lean`) bridges the bounded model checker to denotational semantics. With evidence-grounded atoms, the chain becomes:

$$\texttt{checkLangUsing}\; \ldots\; \texttt{atomCheck}_W\; \textit{fuel}\; p\; \varphi = \texttt{.sat} \tag{2}$$

$$\Longrightarrow\quad \texttt{sem}(\texttt{langReduces}\; \texttt{gfRGLLanguageDef},\; \texttt{atomSem}_W,\; \varphi,\; p) \tag{3}$$

$$\Longrightarrow\quad \text{WM query judgment for threshold atoms} \tag{4}$$

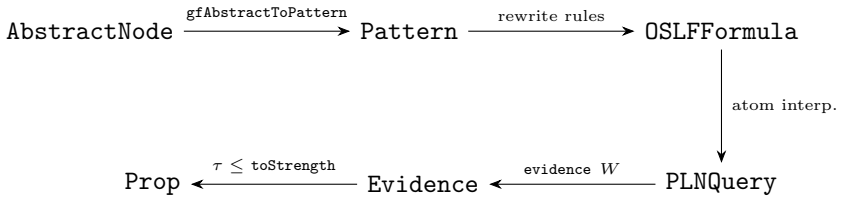


**Figure 2:** Evidence-grounded semantics pipeline: GF tree → pattern → formula → query → evidence → truth.

Step (2)→(3) is the existing soundness theorem. Step (3)→(4) follows by instantiating the atom semantics definition and reading off the evidence extraction.

### B.5 Design Choice: Prop-Threshold vs. Evidence-Valued

We choose **Prop-valued threshold semantics** as the first implementation target because it reuses the existing `sem/checkLangUsing_sat_sound` chain without modifying the OSLF core. A richer alternative is **evidence-valued connective semantics**:

$$\texttt{esem} : \texttt{OSLFFormula} \to \texttt{Pattern} \to \texttt{Evidence}$$

with $\wedge$ as evidence meet ($\sqcap$), $\vee$ as join ($\sqcup$), $\Diamond$ aggregating via $\bigsqcup$ over successors, and a cut theorem:

$$\tau \leq \texttt{toStrength}(\texttt{esem}\ \varphi\ p) \iff \texttt{sem}\ (\ldots)\ \texttt{atomSem}_W\ \varphi\ p$$

This is future work item (4b).

### B.6 Assumptions

The bridge relies on three interface assumptions (to be discharged during implementation):

1. **Query extraction totality.** Every OSLF pattern maps to a well-defined `PLNQuery` (the encoding function `queryOf` must be total).
2. **Threshold monotonicity.** If `evidence` $W$ $q$ increases (in the evidence lattice order), all threshold atoms remain satisfied. This follows from monotonicity of `toStrength` with respect to the evidence order.
3. **Query encoding.** Lexical atoms ("cat", "sleep") map to `PLNQuery.prop`; constructor atoms may map to `PLNQuery.link` or remain purely structural.

### B.7 Theorem Status

**Base–visible separation.** The base relation (syntax rewrites + temporal policy) alone cannot resolve scope ambiguity or anaphora—it preserves the store unchanged. The visible layer supplies the additional reachability needed for these phenomena. This separation is formally witnessed: scope choice is reachable via the full relation but not the base relation, and likewise for the referent-introduction/pronoun-binding chain. Three worked examples demonstrate the complete pipeline: single quantifier (`EveryManWalks`), scope ambiguity (`ScopeAmbiguity`), and cross-sentential anaphora (`AnaphoraBinding`).

| Theorem | Status | Statement sketch |
|---|---|---|
| translation_preserves_evidence_allW | **Proved** | Same abstract tree ⇒ same evidence in all $W$ (WorldModelSemantics.lean) |
| checker_sat_atom_implies_wm_query_judgment | **Proved** | Checker .sat on atom ⇒ WM formula semantics ∧ WM query judgment (WorldModelSemantics.lean) |
| adjCN_preserves_evidence_if | **Conditional** | AdjCN preserves evidence under LexicalEvidenceMonotone (WorldModelSemantics.lean) |
| checker_sat_implies_wm_semantics_general | **Proved** | General checker soundness for arbitrary formulas (WorldModelSemantics.lean) |
| visible_layer_extends_base | **Proved** | Base relation (syntax + temporal) ⊆ full relation (WorldModelVisibleBridge.lean) |
| visible_not_base_witness | **Proved** | Scope choice is full-reachable but not base-reachable (WorldModelVisibleBridge.lean) |
| anaphora_not_base_witness | **Proved** | referent-intro/pronoun-bind chain is full-reachable but not base-reachable (WorldModelVisibleBridge.lean) |
| checkLangUsing_sat_sound | **Proved** | Checker soundness for arbitrary AtomSem (Formula.lean) |
| montague_thesis | **Proved** | Cross-linguistic type invariance (LinguisticInvariance.lean) |
| adjModification_preserves_lexical | **Proved** | Lexical monotonicity of modification (LinguisticInvariance.lean) |
| evidence_add' | **Proved** | Evidence extraction commutes with revision (PLNWorldModel.lean) |

**Table 5:** Status of evidence-grounded semantics results. **Proved** = formally verified in Lean 4 with 0 sorry; **Conditional** = proved under a named assumption.