AMO-Lean: Plan Unificado de Desarrollo

Documento de Diseño y Roadmap Versión: 1.0 **Fecha:** 2026-01-29 **Estado:** Aprobado

---

1. Qué es AMO-Lean

1.1 Definición

AMO-Lean = *Automatic Mathematical Optimizer in Lean*

AMO-Lean es un **optimizador formal** que utiliza el conocimiento de estructuras matemáticas para transformar especificaciones en código optimizado con garantías de corrección.

1.2 La Visión: "Optimización Formal"

```
Especificación       Conocimiento          Código
Matemática      →    Matemático       →    Optimizado
(Lean DSL)           Formal                (C/Rust)
                     (Teoremas)            Correcto por
                                           Construcción
```

Diferenciador clave: - Un compilador tradicional optimiza basándose en **patrones sintácticos** - AMO-Lean optimiza basándose en **propiedades matemáticas verificadas**

1.3 Qué NO es AMO-Lean

NO es	Explicación
Compilador de Lean arbitrario	Solo trabaja con su DSL específico
Una zkVM	Es una herramienta que zkVMs pueden usar
Librería criptográfica	Es un optimizador/verificador
Competidor de Plonky3	Lo complementa

---

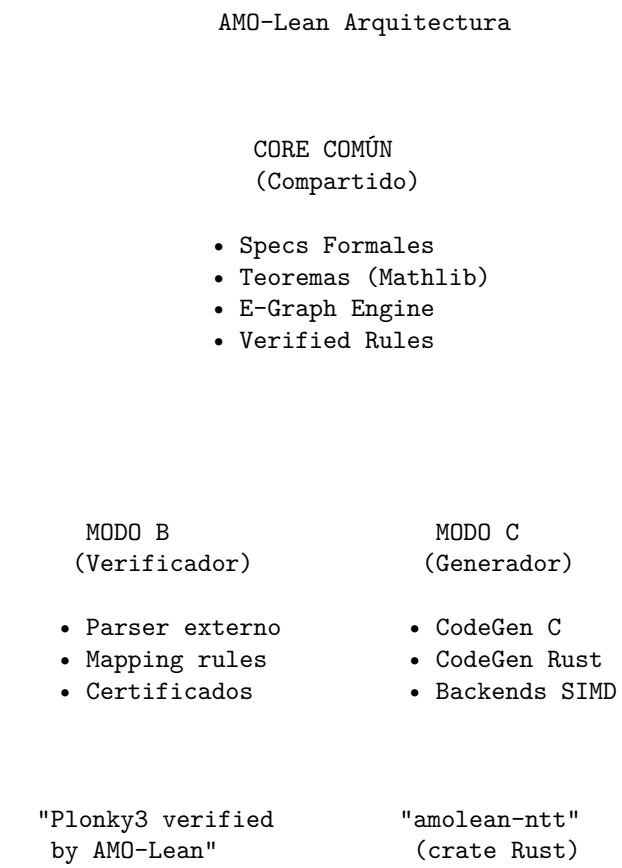
2. Arquitectura Unificada

2.1 Dos Funcionalidades Principales

AMO-Lean tendrá **dos modos de operación** que comparten una base común:

Modo	Nombre	Descripción
B	Verificador	Certifica que optimizaciones externas (ej: Plonky3) son correctas
C	Generador	Produce código optimizado (ej: NTT) con pruebas de corrección

2.2 Diagrama de Arquitectura



2.3 Por Qué Esta Arquitectura

Beneficio	Explicación
Reutilización	~60% del código es compartido entre B y C
Flexibilidad	Usuarios eligen verificar o generar
Valor único	B complementa Plonky3, C compite en NTT verificada
Escalabilidad	Más componentes se agregan igual

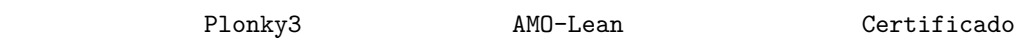
---

3. Modo B: Verificador de Optimizaciones

3.1 Concepto

En lugar de generar código desde cero, AMO-Lean **certifica** que las optimizaciones de proyectos existentes (como Plonky3) son matemáticamente correctas.

3.2 Flujo de Trabajo



```
   (código)
                      1. Parsea regla             "Regla X es
 optimización         2. Mapea a teorema           sound bajo
     X                3. Verifica prueba           axiomas Y"
```

3.3 Ejemplo Concreto

Plonky3 tiene esta optimización:

```rust
// Reducción Goldilocks: usa 2^64  2^32 - 1 (mod p)
fn reduce128(x: u128) -> u64 {
    let (lo, hi) = (x as u64, (x >> 64) as u64);
    let hi_lo = hi & 0xFFFFFFFF;
    let hi_hi = hi >> 32;
    // ... reducción usando la identidad
}
```

AMO-Lean certifica:

```
theorem goldilocks_reduce_correct (x : UInt128) :
    reduce128 x % GOLDILOCKS_P = x % GOLDILOCKS_P := by
  -- Prueba usando 2^64  EPSILON (mod p)
  simp [reduce128, GOLDILOCKS_P, EPSILON]
  ring
```

3.4 Entregables del Modo B

Entregable	Descripción
Framework de mapping	Conecta código externo → teoremas
Formalización Plonky3 Goldilocks	Reglas de campo verificadas
Formalización Plonky3 NTT	Cooley-Tukey verificado
Generador de certificados	Documentos legibles
CI de re-verificación	Automático en cada release

3.5 Valor del Modo B

- **No compite con Plonky3** - lo complementa
- **Aumenta confianza** - "optimizaciones matemáticamente verificadas"
- **Único en el mercado** - nadie más ofrece esto

---

4. Modo C: Generador de Código Verificado

4.1 Concepto

Construir componentes que sean: 1. **Formalmente especificados** en Lean 2. **Implementados con optimizaciones** derivadas de propiedades matemáticas 3. **Usables** desde C y Rust via FFI

4.2 Componente Principal: NTT Verificada

```
                    NTT Verificada de AMO-Lean
```

```
Especificación          Implementación          Binding
(Lean puro)        →    Optimizada (C)     →    (Rust FFI)
                        con prueba de
DFT_n = Σ x_k· ^{jk}    equivalencia

Teoremas:               Código:                 Uso:
• ntt_inverse           • ntt_forward_avx2      • Plonky3
• ntt_convolution       • ntt_inverse_avx2      • zkVM
• cooley_tukey          • ntt_radix4            • cualquiera
```

4.3 Qué Hace Única a Esta NTT

Aspecto	NTT Tradicional	NTT de AMO-Lean
Corrección	Tests empíricos	**Teorema probado**
Optimizaciones	"Creemos que está bien"	**Derivadas de propiedades**
Confianza	"Funciona en tests"	**Matemáticamente imposible estar mal**

4.4 Teoremas Requeridos

```
-- 1. Definición formal
def ntt (v : Vec F n) : Vec F n :=
  fun j => Σ k in range n, v[k] * ^(j*k)

-- 2. Corrección de la inversa
theorem ntt_inverse_correct (v : Vec F n) :
    intt (ntt v) = v

-- 3. Propiedad de convolución (crítica para multiplicación de polinomios)
theorem ntt_convolution (a b : Vec F n) :
    intt (ntt a * ntt b) = cyclic_conv a b

-- 4. Linealidad
theorem ntt_linear (a b : Vec F n) (c : F) :
    ntt (a + c • b) = ntt a + c • ntt b

-- 5. Factorización Cooley-Tukey (la optimización principal)
theorem cooley_tukey_correct (v : Vec F (m * n)) :
    ntt v = (I_m  DFT_n) · T · (DFT_m  I_n) · L · v
```

4.5 Entregables del Modo C

Entregable	Descripción
`ntt_forward`	NTT forward verificada (escalar + AVX2)
`ntt_inverse`	NTT inversa verificada
`ntt_coset`	NTT sobre cosets (para FRI)
Crate Rust `amolean-ntt`	Bindings para integración
Benchmarks	Comparación vs Plonky3
Paper/Docs	Publicación académica

5. Estado Actual del Proyecto

5.1 Fases Completadas

Fase	Descripción	Resultado
0	Proof of Concept	32.3x speedup Lean→C
1	Goldilocks Field	568 M elem/s throughput
2	E-Graph Optimization	91.67% reducción de ops
2.5	Verificación Formal	0 sorry en reglas core
3	AVX2 SIMD	4.00x speedup (100% eficiencia)

5.2 Inventario de Componentes

Componente	Estado	Reutilizable B	Reutilizable C
E-Graph Engine	100%		
Verified Rewrite Rules (12)	100%		
Expr/VecExpr/MatExpr AST	100%		
Goldilocks Spec Lean	100%		
Goldilocks C (escalar)	100%	Ref	
Goldilocks C (AVX2)	100%	Ref	
FRI Fold Spec	100%		
FRI Fold C	100%	Ref	
Poseidon2 Spec	60%		
NTT Spec formal	20%		
NTT Teoremas	0%	Necesario	Necesario
NTT CodeGen	0%	N/A	Necesario

5.3 Porcentajes de Completitud

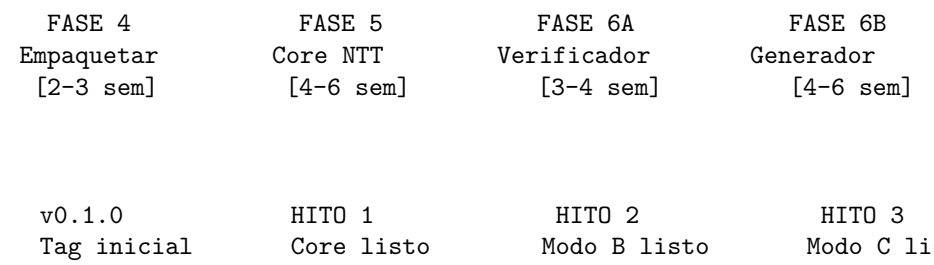
```
Estado Actual (Enero 2026)


Core Común                      55-60%
Modo B (Verificador)            25-30%
Modo C (Generador)              15-20%
```

---

6. Roadmap de Desarrollo

6.1 Diagrama de Fases



6.2 Fase 4: Empaquetar (ACTUAL)

Duración: 2-3 semanas **Objetivo:** Cerrar deuda técnica, no agregar features

Tarea	Descripción	Prioridad
Eliminar 3 sorry	`pow_one`, `one_pow`, `zero_pow`	Alta
Crear `libamolean` básica	Headers públicos vs privados	Alta
CMakeLists.txt	Build system con detección CPU	Alta
README + ejemplos	Documentación mínima de uso	Alta
Tag v0.1.0	Release inicial en GitHub	Alta

Entregable: `libamolean` v0.1.0 con FRI Fold usable

6.3 Fase 5: Core NTT (HITO 1)

Duración: 4-6 semanas **Prerequisito:** Fase 4 completada **Objetivo:** Especificación formal completa de NTT

Semana	Tarea
1-2	`NTT/Definition.lean` - DFT como suma formal
2-3	`NTT/Properties.lean` - Inversa, linealidad
3-4	`NTT/CooleyTukey.lean` - Factorización
4-5	`NTT/Butterfly.lean` - Operación atómica
5-6	Tests, documentación, refinamiento

Desbloquea: Ambos modos B y C se vuelven posibles

6.4 Fase 6A: Verificador (HITO 2)

Duración: 3-4 semanas **Prerequisito:** Hito 1 (Core NTT) **Objetivo:** AMO-Lean funciona como verificador

Semana	Tarea
1	Framework de mapping reglas → teoremas
2	Mapear Goldilocks ops de Plonky3
3	Generar certificado para Goldilocks
4	CI para re-verificar en releases Plonky3

Entregable: "Plonky3 Goldilocks verified by AMO-Lean"

6.5 Fase 6B: NTT Ejecutable (HITO 3)

Duración: 4-6 semanas **Prerequisito:** Hito 1 (Core NTT) **Objetivo:** NTT verificada usable desde Rust

Semana	Tarea
1-2	NTT CodeGen C escalar
2-3	NTT CodeGen C AVX2
3-4	Tests de equivalencia spec = impl
4-5	Benchmarks vs Plonky3 NTT
5-6	Rust bindings (`amolean-ntt` crate)

Entregable: `amolean-ntt` crate publicable

7. Estructura de Archivos Objetivo

```
amo-lean/
  AmoLean/
     Core/
        Expr.lean              # [ ] AST de expresiones
        Rewrite.lean           # [ ] Reglas de reescritura
        Correctness.lean       # [ ] Teoremas de soundness

     EGraph/
        Basic.lean             # [ ] E-Graph engine
        Saturate.lean          # [ ] Equality saturation
        VerifiedRules.lean     # [ ] Reglas con pruebas

     Specs/
        Field/
           Goldilocks.lean     # [ ] Campo Goldilocks
        NTT/
           Definition.lean     # [ ] DFT formal
           Properties.lean     # [ ] Teoremas
           CooleyTukey.lean    # [ ] Factorización
        FRI/
           Fold.lean           # [ ] FRI Fold
        Poseidon/
            Spec.lean          # [ ] Poseidon2

     Verifier/                  # MODO B
        RuleMapping.lean       # [ ] Framework
        Plonky3/
           Goldilocks.lean     # [ ] Reglas Plonky3
           NTT.lean            # [ ] NTT Plonky3
        Certificate.lean       # [ ] Generación

     CodeGen/                   # MODO C
         Core.lean             # [ ] IR común
         C/
            Scalar.lean        # [ ] Backend escalar
            AVX2.lean          # [ ] Backend SIMD
         Rust/
             Basic.lean        # [ ] Backend Rust

  libamolean/                   # Librería C
     include/
        amolean.h              # API pública
     src/
        field_goldilocks.c
        fri_fold.c
        ntt.c                  # [ ]
     CMakeLists.txt

  amolean-ntt/                  # Crate Rust [ ]
     Cargo.toml
```

```
    src/
        lib.rs

docs/
    project/
        UNIFIED_PLAN.md        # Este documento
        ROADMAP.md
        PROGRESS.md
```

---

8. Métricas de Éxito

8.1 Por Fase

Fase	Métrica de Éxito
4	v0.1.0 taggeada, CI verde, 0 sorry
5	5+ teoremas NTT probados
6A	Certificado para Goldilocks de Plonky3
6B	Benchmark NTT 80% rendimiento de Plonky3

8.2 Globales

Métrica	Objetivo
Tests totales	>1500 passing
Sorry en teoremas	0
Cobertura de specs	Goldilocks + NTT + FRI
Documentación	README + ejemplos + API docs

---

9. Riesgos y Mitigaciones

Riesgo	Probabilidad	Impacto	Mitigación
Teoremas NTT difíciles	Media	Alto	Consultar literatura, usar Mathlib
Plonky3 cambia API	Media	Medio	CI detecta, re-mapear
Performance inferior	Baja	Alto	Benchmarks continuos
Scope creep	Alta	Alto	**Seguir este documento**

---

10. Decisiones de Diseño Clave

DD-007: Arquitectura Dual B+C

Decisión: AMO-Lean soporta dos modos (Verificador y Generador) con core compartido.

Razón: Maximiza reutilización (~60% compartido) y flexibilidad.

DD-008: NTT como Componente Principal de Modo C

Decisión: El primer componente verificado será NTT, no Poseidon ni Merkle.

Razón: NTT es matemáticamente rico (muchas propiedades formalizables) y crítico para rendimiento en STARKs.

DD-009: Certificados para Plonky3 Primero

Decisión: El primer target del Verificador será Plonky3, no arkworks ni otros.

Razón: Plonky3 usa Goldilocks (que ya tenemos) y tiene adopción creciente.

DD-010: No Agregar Campos Nuevos en Fase 4-5

Decisión: M31 y BabyBear se posponen hasta después de Hito 3.

Razón: Evitar scope creep. Primero completar funcionalidad core.

---

11. Glosario

Término	Definición
E-Graph	Estructura que representa múltiples expresiones equivalentes
Equality Saturation	Aplicar todas las reglas hasta saturar
NTT	Number Theoretic Transform (FFT sobre campos finitos)
Cooley-Tukey	Algoritmo de factorización para FFT/NTT
Goldilocks	Campo finito p = 2^64 - 2^32 + 1
Plonky3	Librería Rust para STARKs (Polygon)
Verificador (Modo B)	Certifica que código externo es correcto
Generador (Modo C)	Produce código optimizado con pruebas

---

12. Referencias

- SPIRAL Project: https://spiral.ece.cmu.edu/
- Plonky3: https://github.com/Plonky3/Plonky3
- Mathlib: https://leanprover-community.github.io/mathlib4_docs/
- "Term Rewriting and All That" (Baader & Nipkow)
- "Equality Saturation: A New Approach to Optimization" (Tate et al.)

---

13. Historial de Cambios

Fecha	Versión	Cambio
2026-01-29	1.0	Documento inicial con plan unificado

Aprobado por: Manuel Puebla **Fecha de aprobación:** 2026-01-29

AMO-Lean: Automatic Mathematical Optimizer in Lean