Tensorial Rights Framework

A Practical Approach to Music Rights Tracking

Paul Mikulskis
ARC-H1VE

March 19, 2025

Contents

1 Introduction: Addressing the Music Rights Puzzle

Consider a producer named Aria who creates an original beat. A vocalist named Cam licenses it for a track. That track gets licensed to DJ Destiny, who remixes it with another artist's vocals that themselves came through a separate license chain. Later, a film studio wants to license the remix for a movie trailer, but only for US distribution, for 6 months.

Can they do that? Who gets paid? And how much?

This scenario, which happens regularly in today's music ecosystem, illustrates a challenge faced throughout the industry: managing rights across creative collaborations has become remarkably complex. Traditional legal frameworks have served their purpose for many years, but they seem increasingly strained when dealing with the web of interdependent rights that characterize modern creative work. As derivative works multiply exponentially, tracking these rights through manual processes grows nearly impossible.

The challenge isn't simply the volume of rights—it's their inheritance patterns. When DJ Destiny creates a remix, they don't just need to comply with their direct license from Cam, but also with all restrictions that flowed from Aria to Cam, plus any restrictions from the other vocals' rights chains. This cascade of inherited rights creates a validation problem that grows exponentially more complex with each layer of derivation.

The "Tensorial Rights Framework" represents an approach that attempts to apply mathematical principles to help organize and validate intellectual property rights. By introducing some mathematical structure to legal relationships, this framework might help address a growing problem: how to reliably validate rights chains across multiple generations of creative works.

This isn't merely a technical solution—it's potentially a tool that could assist creators, platforms, and rights-holders navigate an increasingly complex landscape. The team at ARCH1VE has been developing this system to address both current challenges and possible future developments in creative collaboration.

2 A Structured Approach: Bringing Mathematical Rigor to Rights Validation

2.1 From Simple Grids to Multi-Dimensional Tensors

Perhaps an easier way to begin understanding this approach is to think of a simple spreadsheet or X/Y grid. On one axis, you might have different types of rights (reproduce, distribute, modify). On the other axis, you have modifiers to those rights (time limits, territory restrictions). At each intersection point, you could have a simple "yes/no" or a set of specific conditions (or any primitive/basic value).

This two-dimensional grid seems adequate for simple cases—like a single license between a creator and a user—but falls short when dealing with derivative works. A grid can represent what rights a user has to a work, but not how those rights should transform when that user creates a new work based on the original. The grid can't capture the flow of rights through the creative chain.

Music rights appear to have several additional complexities that make a more structured model helpful:

1. **Multiple types of rights** exist for any creative work (distribution, modification, reproduction, etc.)

2. **Each right can have multiple modifiers** (time limits, territory restrictions, exclusivity, etc.)

3. **Rights flow across works** from originals to derivatives, each with their own constraints

4. **Multiple constraint chains** must be evaluated simultaneously for proper validation

The last two points are particularly challenging. When DJ Destiny creates a remix, their rights aren't just determined by a single grid—they're determined by how rights from multiple previous works combine and transform as they flow into the new creation, and perhaps even dynamically aggregate and change as they flow through the chain. This is where the grid analogy begins to break down.

When explored further, the grid starts to resemble something more like a multi-layered filter or sieve, with rights flowing through consecutive layers. From a mathematical perspective, tensor algebra might offer helpful tools. Tensors—mathematical objects that extend the concept of vectors to multiple dimensions—could potentially provide a useful foundation. Similar to how physicists use tensors to model complex phenomena across multiple dimensions of space and time, this approach applies them to model the multi-dimensional nature of creative rights.

In the framework, a user's rights to a creative work might form a tensor R where:

$$R = \{D_1, D_2, ..., D_n\} \times \{M_1, M_2, ..., M_m\}$$

Here, each primary dimension D_i represents a fundamental right (e.g., RECORD, REPRODUCE, MODIFY), while each secondary dimension M_j represents a modifier category (e.g., DURATION, TERRITORY). At each intersection lies a constraint function that determines whether a specific action is permitted.

But to handle derivative works, we need something more: a system that defines how these tensors transform and combine when flowing from one work to another. This is where propagation operators become essential—they model how rights restrictions flow across generations of creative works.

2.2 Visualizing the Constraint Network

From a visual perspective, this tensor approach could be understood as a type of sieve:

- Each primary layer represents a fundamental right (e.g., RECORD, REPRODUCE, MODIFY)
- Each section of the sieve contains different types of filters (time, territory, etc.)
- Actions can only pass through if they satisfy all constraint functions

One possible advantage of this approach is that it offers both mathematical precision and adaptability. If new types of rights or modifiers emerge in the future (as seems likely), new dimensions could be added to the tensor space without redesigning the entire system.

2.3 The Mathematics of Rights Validation

At its core, rights validation tries to answer a straightforward question: "Can this person do this thing with this work?" The framework uses formal boolean algebra to attempt to answer this question with mathematical clarity.

For example, consider a producer who wants to know if they can sample a track for a commercial project:

1. **Conjunction (AND)**: $f_1(x) \wedge f_2(x)$ - The user must have both REPRODUCTION rights AND MODIFY rights
2. **Disjunction (OR)**: $f_1(x) \vee f_2(x)$ - The territory restriction permits use in EITHER the US OR Canada

3. **Negation (NOT)**: $\neg f(x)$ - The work must NOT be used in a political campaign
4. **Implication**: $f_1(x) \implies f_2(x)$ - IF the work is used commercially, THEN attribution must be provided

These operators might allow for combining simple constraints into more complex validation rules that align with legal requirements. However, this only handles validation for a single work with a single user—it doesn't address how rights flow through derivative chains.

2.4 Formal Constraint Function Properties

To build a mathematically sound framework, we must precisely define the properties of constraint functions that operate within the tensorial space.

2.4.1 Boolean Constraint Functions

In the standard formulation, a constraint function f is defined as:

$$f : X \to \{0, 1\}$$

Where X is the domain of possible input values relevant to the constraint (e.g., dates, territories, usage types), and the output is strictly binary: 1 (permitted) or 0 (not permitted).

These functions must satisfy several mathematical properties to ensure consistent operation within the rights tensor:

- **Closure**: For any $x \in X$, $f(x) \in \{0, 1\}$ - The output is always well-defined
- **Composability**: For functions $f_1, f_2 : X \to \{0, 1\}$, the compound function $(f_1 \wedge f_2) : X \to \{0, 1\}$ is also well-defined
- **Determinism**: For any fixed $x \in X$, $f(x)$ always evaluates to the same result

Boolean constraint functions form a Boolean algebra when combined with the operations $\wedge$ (AND), $\vee$ (OR), and $\neg$ (NOT), allowing for systematic composition of complex validation rules:

$$F(x) = (f_1(x) \wedge f_2(x)) \vee (\neg f_3(x))$$

This formulation enables rigorous proofs about the validity of rights assertions. For instance, if R_1 represents the rights granted to a user and R_2 represents the rights required for an action, then the action is permitted if and only if:

$$\forall d \in D, \forall m \in M, R_2(d, m) = 1 \implies R_1(d, m) = 1$$

Which captures the containment relationship $R_2 \subseteq R_1$ in set-theoretic terms.

2.4.2 Continuous Constraint Functions

While boolean constraints provide clear yes/no validation, they fail to capture the nuanced reality of many music industry agreements. We can extend our framework to continuous constraint functions:

$$f : X \to [0, 1]$$

Where the output represents a "degree of permission" rather than a binary state. This creates a continuous spectrum of permissions with several advantageous properties:

- **Graduation of rights**: Permissions can strengthen or weaken based on contextual factors

- **Fuzzy boundaries**: The transition between permitted and non-permitted states can be gradual
- **Weighted aggregation**: Multiple partial permissions can be combined with weighted importance

Continuous functions maintain mathematical rigor through modified composition rules:

- **Fuzzy conjunction**: $(f_1 \wedge f_2)(x) = \min(f_1(x), f_2(x))$
- **Fuzzy disjunction**: $(f_1 \vee f_2)(x) = \max(f_1(x), f_2(x))$
- **Fuzzy negation**: $\neg f(x) = 1 - f(x)$
- **Weighted aggregation**: $F(x) = \sum_{i=1}^{n} w_i f_i(x)$ where $\sum_{i=1}^{n} w_i = 1$

These operations preserve the algebraic structure while allowing for more sophisticated modeling of rights scenarios.

2.4.3 Industry Applications of Continuous Rights

The continuous rights space directly addresses several complex industry scenarios that are difficult to model with boolean constraints:

- **Revenue-proportional usage**: Many music deals include royalty rates that scale based on commercial success. This can be modeled as:

$$f_{royalty}(revenue) = \begin{cases} 0.8 & \text{if } revenue < \$10,000 \\ 0.8 - 0.3 \cdot \min(1, \frac{revenue - \$10,000}{\$90,000}) & \text{otherwise} \end{cases}$$

  This creates a smooth transition from 80% artist royalty to 50% as revenue scales from \$10K to \$100K.

- **Temporal gradation**: Rights that naturally expire or transition over time:

$$f_{exclusivity}(t) = \max(0, 1 - \frac{t - t_0}{t_{end} - t_0})$$

  Where exclusivity gradually diminishes from 1 (fully exclusive) to 0 (non-exclusive) over the period from t_0 to t_{end}.

- **Contextual weighting**: Rights that strengthen or weaken based on context:

$$f_{territory}(loc) = \begin{cases} 1.0 & \text{if } loc \in \text{primary territories} \\ 0.7 & \text{if } loc \in \text{secondary territories} \\ 0.3 & \text{if } loc \in \text{tertiary territories} \\ 0 & \text{otherwise} \end{cases}$$

For LicenseGrantMeta objects, the ‘prefilledFields‘ mechanism is particularly well-suited to encoding continuous functions, allowing template-based licenses that maintain sophisticated gradations without sacrificing mathematical precision.

2.4.4 Implementation Strategy for Continuous Rights

A practical implementation approach can maintain backward compatibility while introducing continuous validation:

1. **Core validation**: Use boolean constraints for foundational rights validation
2. **Extended validation**: Apply continuous constraints for specific modifiers where gradation is beneficial
3. **Hybrid evaluation**: Combine results using a rule like:

$$R_{valid} = \bigwedge_i B_i(x) \cdot \prod_j C_j(x)$$

   Where B_i are boolean constraints and C_j are continuous constraints

This preserves hard barriers (like territory restrictions) while enabling flexible modeling of commercial terms, particularly beneficial for complex multi-party arrangements characteristic of the music industry.

2.5 Propagation Operators: Modeling Rights Inheritance

For derivative works, the framework defines what could be called a **propagation operator** $P(R)$ that transforms rights as they flow downstream. This is what solves the inheritance challenge presented in our initial example. When DJ Destiny makes a remix, propagation operators ensure that Aria's "no alcohol advertising" restriction automatically applies to all derivatives—conceptually similar to water passing through consecutive sieves, but with mathematical rigor.

Returning to our example, if the film studio wanted to use DJ Destiny's remix in a beer commercial, the system would trace restrictions through the entire chain: Aria → Cam → DJ Destiny → Film Studio. Even if DJ Destiny's direct license with Cam didn't mention alcohol advertising, the propagation operator would carry Aria's original restriction forward, flagging the potential violation.

2.5.1 Formal Definition of Propagation Operators

To mathematically formalize propagation, we define a propagation operator P as a transformation from one rights tensor to another:

$$P : \mathcal{R} \to \mathcal{R}$$

Where $\mathcal{R}$ is the space of all possible rights tensors. When a work W_2 derives from work W_1, the rights tensor R_2 of work W_2 is constrained by the propagated rights from W_1:

$$R_2 \subseteq P(R_1)$$

For a direct transformation from R_1 to R_2, this is straightforward. However, the true power of the formalism emerges with multi-generational derivatives. If work W_3 derives from W_2, which itself derives from W_1, the rights validation must satisfy:

$$R_3 \subseteq P(R_2) \subseteq P(P(R_1))$$

This implies the critical compositional property of propagation operators:

$$P_2 \circ P_1 = P_{2 \circ 1}$$

Where $\circ$ denotes operator composition. This property ensures that rights flow consistently through each generation of derivative works.

2.5.2 Properties of Propagation Operators

Propagation operators should satisfy several mathematical properties to ensure consistent and legally sound rights inheritance:

1. **Monotonicity**: For rights tensors R_A and R_B, if $R_A \subseteq R_B$, then $P(R_A) \subseteq P(R_B)$. This ensures that adding restrictions never expands permissions in derivatives.

2. **Idempotence**: For certain classes of restrictions, $P(P(R)) = P(R)$. This property holds for "absolute" restrictions (like "no hate speech") that propagate unchanged through any number of generations.

3. **Distributivity**: For rights tensors from multiple source works $R_1, R_2, ..., R_n$ combining into a derivative, the propagated rights satisfy:

$$P(R_1 \cap R_2 \cap ... \cap R_n) = P(R_1) \cap P(R_2) \cap ... \cap P(R_n)$$

   This allows for simpler computation of propagated rights from multiple sources.

2.5.3 Types of Propagation Functions

Different types of rights require different propagation behaviors. We can formally categorize the most common propagation patterns:

- **Strict propagation**: $P_{strict}(R)(d, m) = R(d, m)$

  The constraint passes unchanged, maintaining identical boundaries through all derivative generations.

- **Territorial intersection**: For territory sets $T_1, T_2, ..., T_n$:

$$P_{territory}(R_1, R_2, ..., R_n) = \{x \in X | x \in T_1 \cap T_2 \cap ... \cap T_n\}$$

  Only territories permitted in all source works remain permitted in derivatives.

- **Duration minimization**: For time durations $d_1, d_2, ..., d_n$:

$$P_{duration}(R_1, R_2, ..., R_n) = \min(d_1, d_2, ..., d_n)$$

  The most restrictive time limit propagates forward.

- **Usage exclusion propagation**: For excluded usage sets $E_1, E_2, ..., E_n$:

$$P_{exclusion}(R_1, R_2, ..., R_n) = E_1 \cup E_2 \cup ... \cup E_n$$

  All exclusions from all source works propagate forward.

2.5.4 Propagation in Continuous Rights Spaces

When constraint functions produce continuous values rather than binary outcomes, propagation operators extend naturally using principles from fuzzy set theory:

- **Fuzzy intersection**: $P_{fuzzy-min}(R_1, R_2, ..., R_n)(x) = \min(R_1(x), R_2(x), ..., R_n(x))$
- **Fuzzy exclusion**: $P_{fuzzy-max}(R_1, R_2, ..., R_n)(x) = \max(R_1(x), R_2(x), ..., R_n(x))$
- **Weighted propagation**: $P_{weighted}(R_1, R_2, ..., R_n)(x) = \sum_{i=1}^{n} w_i R_i(x)$ where $\sum_{i=1}^{n} w_i = 1$

This extension preserves the mathematical soundness of the propagation while accommodating the more nuanced rights modeling afforded by continuous constraint functions.

2.5.5 Implementation Efficiency

The mathematical structure of propagation operators enables efficient implementation strategies:

1. **Caching**: Since $P(R)$ depends only on R, propagated results can be cached and reused.
2. **Parallel evaluation**: The distributive property allows parallel computation of propagation from multiple sources.
3. **Incremental updates**: For a derivative work with multiple sources, if one source's rights change, only that propagation path needs recalculation.

These properties are particularly valuable for maintaining validation performance as derivative chains grow in complexity.

The true value of these propagation operators lies in their practical implementation. The framework begins with a core set of propagation evaluators that handle the most common inheritance patterns:

- **Strict propagation:** Some restrictions (like "no hate speech") always flow unchanged through all derivative works
- **Territorial intersection:** If Work A allows use in North America and Work B allows use in the US and Mexico, a derivative of both works would be restricted to just the US and Mexico
- **Duration minimization:** Time limits propagate by taking the most restrictive duration from contributing works
- **Usage type filtering:** Categorical exclusions (like "no political use") flow forward to all derivatives

These base patterns solve most practical validation scenarios needed for a functioning marketplace, without requiring overly complex implementations.

What makes this approach particularly valuable is how it addresses operational challenges that typically plague rights management systems. Specifically, the framework enables features that would be difficult to implement in traditional rights databases:

- **Automated chain validation:** When a DJ wants to license a remix of a track that itself contains samples, the system can trace rights inheritance through multiple generations of works, applying propagated restrictions without requiring complete manual review. For platforms handling derivatives, this could potentially reduce clearance times significantly.
- **Deterministic conflict detection:** When incompatible rights collide (e.g., a sample with "no commercial use" flowing into a work being monetized commercially), the system can identify the conflict point in the rights chain—showing which original work's restriction is being violated and where in the inheritance path the conflict occurs.
- **Marketplace rule enforcement:** Sales channels, streaming platforms, and promotional contexts each have distinct licensing requirements. The propagation approach could help filter catalogs to show works valid for specific contexts—enabling marketplaces to present appropriately licensable works for each usage type.
- **Dynamic royalty distribution:** As constraints flow through the system, corresponding royalty distribution rules can similarly propagate, helping to calculate complex splits (e.g., 5% to original sample creator, 15% to remixer, 80% to final artist), even as works are reused across generations.

For rights management operations, these capabilities suggest potential practical benefits:

- **Reduced legal review:** Automated constraint validation could help reduce manual review requirements. For a typical music catalog with multiple dependency layers, this might meaningfully decrease legal review overhead.
- **Faster clearance processes:** When an artist wants to release a track containing samples, traditional clearance processes can take weeks. A systematic validation approach could provide earlier feedback on clearance issues.
- **Licensing opportunity identification:** By efficiently validating constraint chains, the system might identify overlooked licensing opportunities—works that were thought to have conflicting constraints but are actually valid for specific use cases when analyzed systematically.
- **Manageable complexity:** As derivative works become more complex (e.g., AI-generated music incorporating multiple samples), a structured approach to rights validation becomes increasingly valuable.

Here's a concrete example to illustrate:

```
// Rights tensor for original creator (full rights)
OriginalCreator = {
  REPRODUCE: { constraints: [] },
  DISTRIBUTE: { constraints: [] },
  MODIFY: { constraints: [] }
}

// Rights tensor for licensee (limited rights)
Licensee = {
  REPRODUCE: {
    constraints: [
      // Can only reproduce for 2 years
      DURATION("P2Y"),
      // Can only reproduce in North America
      TERRITORY(["US", "CA", "MX"]),
      // Cannot use for political campaigns
      USAGE_EXCLUSION(["POLITICAL"])
    ]
  },
  DISTRIBUTE: {
    constraints: [
      // Same constraints for distribution
    ]
  }
}
```

When this licensee wants to create a derivative work and offer their own license templates, the system could potentially act as a verification sieve, perhaps helping to ensure they're only offering rights they actually possess—which might help prevent potential legal issues.

2.6 Containment and Chain Validation

One aspect of the tensorial approach that seems particularly promising is how it might handle license template validation. When a user wants to determine if they can offer a specific license template for a work, this could potentially be framed as a mathematical containment problem:

Given a user's rights tensor R_u and a license template's required rights tensor R_t, the user can offer the template if and only if:

$$R_t \subseteq R_u$$

This means that every dimension in the template's tensor must have constraints that are equal to or less restrictive than the corresponding constraints in the user's tensor. This relationship could potentially be proven mathematically and implemented computationally, though more development is needed.

For instance, if a user has a time-limited right to distribute in North America, they logically cannot offer a worldwide distribution license or one that extends beyond their time limit. The tensorial approach might reduce these complex validations to mathematical operations that could be performed systematically.

In the more complex case of multi-generation derivatives (remixes of remixes), the containment check would need to incorporate the entire chain of rights, ensuring that each generation respects the constraints of all previous generations. The mathematical formalism for this aspect of the framework is still under development.

3 From Concept to Implementation

3.1 The Grant Dimension Mapping

A critical aspect of this framework is how it maps from abstract legal concepts to a mathematical model. Each grant type in a database could become a dimension in the tensor space:

Legal Grant Type	**Tensor Dimension**	**Description**
RECORD	D_{RECORD}	Right to record the work
REPRODUCE	$D_{REPRODUCE}$	Right to reproduce the work
CREATE_DERIVATIVE_WORKS	$D_{DERIVATIVE}$	Right to create derivative works
...	...	...

Table 1: Mapping legal grant types to tensor dimensions

Similarly, each modifier type might become a secondary dimension:

Legal Modifier Type	**Tensor Dimension**	**Description**
TIME_DURATION	$M_{DURATION}$	Time limitations on the grant
EXCLUSIVITY	$M_{EXCLUSIVE}$	Whether rights are exclusive
TERRITORY	$M_{TERRITORY}$	Geographic limitations
...	...	...

Table 2: Mapping legal modifier types to tensor dimensions

This mapping could create a bridge between legal concepts and mathematical operations. In theory, when a legal team defines a new grant or modifier type, it might extend the tensor space without requiring a complete system redesign, though practical implementation would likely present several challenges.

3.2 Practical Implementation

The mathematical foundation might be elegant in theory, but early implementation attempts have revealed several challenges:

```
// Simplified validation function
function validateUserAction(
  userId: string,
```

```
  workId: string,
  action: GrantDimension,
  context: ValidationContext
): Promise<ValidationResult> {
  // 1. Calculate user's rights tensor for this work
  const rightsModel = await calculateTensorialRightsModel(userId,
      workId);

  // 2. Check if user has the requested grant dimension
  if (!rightsModel.grants[action]) {
    return { valid: false, reason: "User does not have this right" };
  }

  // 3. Evaluate all constraints for this grant
  const grant = rightsModel.grants[action];
  for (const constraint of grant.constraints) {
    // Get value to validate from context
    const valueToValidate = context[constraint.fieldKey];

    // Evaluate the constraint function
    const isValid = await constraint.evaluate(valueToValidate, context)
        ;

    // If any constraint fails, the action is not permitted
    if (!isValid) {
      return {
        valid: false,
        reason: `Failed constraint: ${constraint.description}`
      };
    }
  }

  // All constraints passed - action is permitted
  return { valid: true };
}
```

This approach could potentially validate rights queries in a systematic way—perhaps allowing for future scaling while maintaining legal compliance, though much more testing would be needed before rolling this out to users.

3.3 Combinatorial Growth Through Reusability

One insight from this development is that rights can be treated as composable, reusable components within a defined algebraic structure:

1. **Basic Boolean Algebra**: By reducing complex legal statements to boolean expressions, the framework creates building blocks that can be combined in various ways.

2. **Constraint Reusability**: Each constraint implementation (e.g., time limits, territory restrictions) becomes a reusable component applicable across different rights types.

3. **Extensible System**: New constraints and grant types can be added without modifying existing code—the tensor structure naturally accommodates additional dimensions, though this requires practical testing.

This combinatorial approach suggests that the system's capabilities might grow more rapidly than the resources invested by a small team. For example, ten new rights types combined

with ten modifier types would create a hundred possible combinations, all potentially validated through the same mathematical framework without requiring custom code for each case. However, real-world legal complexity will likely present edge cases not yet considered.

3.4 The Challenge of Dynamic Constraint Generation

In the database model for this system, modifiers to grants are stored as:

```
interface GrantModifierMeta {
  id: UUID;                // Unique identifier
  type: string;            // The type of modifier (e.g., "TIME_DURATION
     ")
  displayNote: string;     // Human-readable description
  internalNote?: string;   // Additional context for implementors
  createdAt: Date;         // Metadata
  updatedAt: Date;         // Metadata
}
```

This structure is intentionally abstract and flexible—allowing new types of modifiers to be added to the database without changing its schema. However, this flexibility creates a practical challenge: how to map from these abstract types to concrete validation logic.

Rather than using a rigid one-to-one mapping from modifier types to constraint functions, the framework implements a registry system that might better handle the complexity of real-world legal constraints:

```
/**
 * Maps from GrantModifierMeta types to constraint functions
 * with sophisticated matching logic
 */
const constraintRegistry: ConstraintRegistryEntry[] = [
  {
    // Base TIME_DURATION constraint
    grantModifierMetaType: "TIME_DURATION",
    implementation: {
      tier: ConstraintTier.PRIMITIVE,
      generate: generateTimeDurationConstraint,
    },
    description: "Basic time duration constraint",
  },
  {
    // Specialized TIME_DURATION for music releases
    grantModifierMetaType: "TIME_DURATION",
    fieldMatchers: [
      {
        key: "useCase",
        type: "ENUMERATED",
        priority: 10,
      },
    ],
    implementation: {
      tier: ConstraintTier.PRIMITIVE,
      generate: generateMusicReleaseDurationConstraint,
    },
    description:
      "Time duration for music releases with use case restrictions",
  },
  // Additional specialized implementations...
];
```

This registry might allow the system to handle some of the complexity of legal contracts, where the same modifier type may require different validation logic depending on context. This approach is still experimental and would benefit from feedback from legal experts.

4 Future Directions

While the current approach addresses some common rights scenarios, several promising directions for further development exist:

4.1 Context-Aware Constraints

Currently, the rights tensor primarily operates in two dimensions: grant types and modifiers. A potentially valuable extension might be to incorporate context as a third dimension, creating what could be described as a third-order tensor:

$$R = \{D_1, D_2, ..., D_n\} \times \{M_1, M_2, ..., M_m\} \times \{C_1, C_2, ..., C_p\}$$

Where each C_k could represent a context dimension such as:

- The purpose of use (educational, commercial, personal)
- The medium of distribution (streaming, physical, broadcast)
- The relationship between parties (label-artist, artist-collaborator)

This approach might allow the same set of rights to evaluate differently based on specific situations, somewhat similar to how certain legal doctrines (like fair use) apply differently depending on context. However, this would significantly increase the mathematical complexity, and the practical benefits might not outweigh the additional implementation challenges.

4.2 Transparent Validation Pathways

To improve user experience, what might be called "Validation Pathways" could show why a particular action is or isn't permitted:

- **Permission visualizations**: Visual representations of the tensor space showing which dimensions a user has access to and which they don't
- **Alternative options**: Identifying possible ways to achieve similar goals within current rights constraints
- **Path suggestions**: Computing equivalent tensor configurations that might be permitted under current rights

This could potentially transform rights management from a simple yes/no decision to a more nuanced exploration of possibilities within legal boundaries. These are preliminary ideas, and much more work would be needed to determine feasibility.

4.3 Collective Rights Management

Perhaps most importantly, this type of framework might eventually help multiple parties collaborate on complex rights scenarios:

- **Collaborative validation**: Allowing different rights administrators to maintain separate parts of the tensor space that could be combined through defined operations

- **Rights modeling**: Providing ways for parties to model different rights configurations before legal agreements
- **Efficient licensing**: Creating channels where standardized rights packages could be exchanged with more clarity

5 Computational Complexity and Scaling Properties

A critical aspect of any rights management framework is how it scales with increasing complexity. The tensorial approach offers several advantageous scaling properties that make it viable for large-scale music rights ecosystems.

5.1 Complexity Analysis

5.1.1 Time Complexity

Let's define the key variables affecting computational complexity:

- n = number of works in a derivative chain
- d = number of grant dimensions (rights types)
- m = number of modifier dimensions
- c = average number of constraints per grant-modifier pair

For a single rights validation query, the time complexity would be:

$$O(d \cdot m \cdot c)$$

This represents the cost of evaluating all constraint functions across all dimensions of the tensor. Importantly, this complexity is independent of the chain length n, assuming the propagated rights have been precomputed and cached.

For propagation computation in a linear chain of derivatives, the time complexity becomes:

$$O(n \cdot d \cdot m \cdot c)$$

This linear scaling with chain length n could offer advantages over traditional rights clearance approaches, where validation often becomes more complex with derivative depth due to the increasing number of possible conflicts.

5.1.2 Space Complexity

The space required to represent a full rights tensor is:

$$O(d \cdot m \cdot c)$$

However, in practice, the tensor is typically sparse—most grant-modifier pairs don't have constraints. Using sparse tensor representations reduces this to:

$$O(k)$$

Where k is the number of non-zero entries in the tensor (i.e., the number of actual constraints). Empirically, $k \ll d \cdot m$ for most real-world licenses, often by several orders of magnitude.

For caching propagated rights through a derivative chain, the worst-case space complexity is:

$$O(n \cdot k)$$

Again, this linear scaling with chain length enables practical implementation even for complex multi-generational derivative works.

5.2 Algorithmic Optimization

Several mathematical properties of the tensor model enable significant algorithmic optimizations:

1. **Pruning**: Since $R_2 \subseteq P(R_1) \implies P(R_2) \subseteq P(P(R_1))$, we can prune validation paths once we determine a constraint must propagate.

2. **Memoization**: For any specific $(work, user)$ pair, the rights tensor R needs to be computed only once and can be cached since:

$$validateRights(work, user, action) = F(R_{work,user}, action)$$

3. **Incremental validation**: When a new constraint is added to a work in the middle of a derivative chain, we need only recompute propagation forward from that point, not the entire chain.

These optimizations are mathematically guaranteed by the properties of constraint functions and propagation operators defined earlier.

5.3 Scaling with Business Growth

The framework's mathematical structure provides scaling advantages beyond just computational complexity:

5.3.1 Growing Rights Dimensions

As the business expands to new rights types or modifiers, the tensor model scales through dimensional expansion without requiring architectural changes. Adding a new grant dimension D_{n+1} or modifier dimension M_{m+1} is an independent operation:

$$R_{new} = R_{old} \times \{D_{n+1}\} \text{ or } R_{new} = R_{old} \times \{M_{m+1}\}$$

This dimensional independence means existing code paths remain valid when new rights types are introduced.

5.3.2 Parallel Validation

The framework's algebraic structure enables efficient parallelization. Since constraint evaluation across dimensions is independent:

$$F(R)(d, m) = \bigwedge_i f_i(x)$$

Evaluation of each f_i can be performed in parallel, with only the final conjunction requiring synchronization. This allows horizontal scaling across computational resources as validation load increases.

5.3.3 Convergence Guarantees

For iterative propagation algorithms, we can prove bounded convergence using the monotonicity of propagation operators. Let $R^0 = R$ and $R^{i+1} = P(R^i)$. Since P is monotonic and the constraint space is finite, this sequence converges in at most $d \cdot m \cdot c$ iterations:

$$R^0 \supseteq R^1 \supseteq R^2 \supseteq ... \supseteq R^{d \cdot m \cdot c} = R^{d \cdot m \cdot c+1}$$

This bounded convergence guarantee ensures that even complex propagation rules stabilize efficiently.

5.4 Empirical Performance Analysis

Initial prototyping suggests promising performance characteristics:

- **License validation**: Millisecond-range operations for typical structures
- **Chain validation**: Appears to scale linearly with chain length
- **Memory usage**: Compact representation for typical licenses
- **Propagation computation**: Most test cases converge rapidly

These early results suggest the framework could potentially scale to practical deployment, handling substantial catalogs with derivative chains while maintaining reasonable validation performance.

6 Conclusion

The Tensorial Rights Framework represents an attempt to bring mathematical structure to the complex problem of rights management in the music industry. By applying mathematical concepts to rights validation, we're exploring approaches to handle the growing complexity of creative collaboration.

This approach is not intended as a replacement for legal expertise—rights management will always require human judgment for edge cases and nuanced interpretation. However, a formal mathematical foundation might help simplify routine validation, potentially reduce legal risks, and create new opportunities for collaboration in creative industries.

The hope for this framework at ARC-H1VE is to contribute to a more transparent and creator-friendly rights ecosystem. There are still many challenges to overcome, and feedback from both technical and legal experts will be essential as this system continues to develop.