Deterministic Isomorphic Validation of Composite Card Structures in Non-Monotonic Rank Hierarchies

Muhammad Rengga Putra Kuncoro

January 2026

Abstract

This paper presents a deterministic algorithmic framework for validating structural adherence in sequential games characterized by non-linear rank hierarchies and bounded multiplicity ($k \leq 2$). We introduce a piecewise mapping transformation, Φ, which projects a discontinuous rank space into a normalized integer domain, enabling constant-time adjacency verification for sequential structures. To ensure structural isomorphism between lead templates and follower plays, we define a validation engine that explores the symmetric group S_n. While the search space for a generalized multiset matching problem is NP-hard, we demonstrate that the physical constraints of the card domain bound the requirement set n. By leveraging greedy decomposition to minimize the cardinality of the requirement blocks, the engine achieves deterministic, real-time arbitration within the latency thresholds required for interactive gameplay. This formalization provides a rigorous foundation for rule enforcement in complex combinatorial environments such as the game of Tractor.

1 Introduction

Trick-taking games typically rely on static, linear card rankings. However, complex combinatorial games such as Tractor (Sheng Ji) introduce dynamic hierarchies that reorder the deck based on a floating Current Rank (CR). This creates a non-linear state space that complicates both the validation of legal plays and the resolution of trick outcomes.

A significant challenge in these environments is the enforcement of *isomorphic adherence*, where subsequent players must match the lead's structural composition (comprising singles, pairs, and sequential tractors) even as the underlying rank values shift. This research was inspired by the `TRACTOR` challenge on the Sphere Online Judge (SPOJ) platform, which highlights the structural ambiguities inherent in such rulesets.

This paper defines a formal method for verifying these plays by:

i. Transforming the irregular, discontinuous rank-space into a normalized, continuous integer domain (Φ) for efficient adjacency verification.

ii. Implementing a greedy structural deconstruction engine that decomposes complex plays into their constituent primitives.

iii. Applying a permutative matching algorithm to verify structural isomorphism between the lead's requirements and the follower's hand.

This integrated framework[1] allows for rigorous rule enforcement and dominance evaluation in multi-deck environments with bounded multiplicity.

[1]The complete C++ implementation and documented LaTeX source are available at: `https://github.com/tendonintendo/TRACTOR`

2 Game Preliminaries and Definitions

To provide a basis for algorithmic validation, the rules of the environment must be formalized as a set of structural constraints. We define the game state through deck composition, rank hierarchy, and the requirements for isomorphic adherence.

2.1 Deck Composition and Bounded Multiplicity

The system operates on a multi-deck environment where each card c is defined by the tuple (r, s), representing its rank and suit respectively. In a two-deck configuration, the multiplicity of any unique card is exactly two. This bounded multiplicity is a critical constraint, as it facilitates the formation of identical pairs which serve as the fundamental building blocks for higher-order structures.

2.2 The Active Hierarchy and Trump System

Unlike standard trick-taking games, the rank hierarchy is dynamic and non-linear. A Current Rank $CR \in \{2, 3, \ldots, 14\}$ and a Main Suit S_m are designated at the start of each round. For the purpose of mathematical modeling, face cards are mapped to integer values as follows:

Table 1: Face Card Rank Mapping

Rank Symbol	J	Q	K	A
Integer Value	11	12	13	14

The designation of CR and S_m partitions the deck into two primary sets:

- **Trump Set ($\mathcal{T}$):** Includes all cards of suit S_m, all cards of rank CR regardless of suit, and the supreme trumps (Jokers).
- **Natural Set ($\mathcal{N}$):** Includes all remaining cards, organized by their respective suits.

A critical aspect of this system is *Rank Displacement*. The CR is extracted from its natural sequence and elevated to a priority position within $\mathcal{T}$. This elevation creates a structural discontinuity in the remaining ranks. For instance, if $CR = 11$ (Jack), then the ranks 10 and 12 become logically adjacent.

2.3 Structural Primitives: Pairs and Tractors

The game recognizes three levels of structural complexity, each defined by its multiplicity and adjacency requirements.

1. **Singles ($k = 1$):** Any individual card $c \in \mathcal{C}$.
2. **Pairs ($k = 2$):** A set of two identical cards $P = \{c_1, c_2\}$ such that $rank(c_1) = rank(c_2)$ and $suit(c_1) = suit(c_2)$.
3. **Tractors ($L \geq 2, k = 2$):** A sequence of L pairs.

Formally, we define a Tractor T as a collection of cards that must satisfy a strict homogeneity constraint. Let $S(c)$ be a function returning the suit of card c, and let $\mathcal{T}$ be the Trump set. A Tractor T of length L is valid if and only if:

$$\forall c_i, c_j \in T, \quad (c_i, c_j \in \mathcal{T}) \vee (S(c_i) = S(c_j) = \mathcal{S}_n)$$

where $\mathcal{S}_n$ is a specific natural suit. Furthermore, the set of unique transformed ranks $\{\Phi(c) \mid c \in T\}$ must form a contiguous integer interval:

$$\{\Phi(c) \mid c \in T\} = \{x, x+1, \ldots, x+L-1\}$$

2.4 Composite Structural Sets (The Throw)

In a game of Tractor (Sheng Ji), a "Throw" is defined as an *atomic composite move* comprising a multi-set of cards C. Unlike standard moves, a Throw is a batch operation where the entire set must be confined to a single logical partition of the deck to maintain suit-integrity constraints. Formally, C is a valid composite structure if and only if:

$$C \subseteq \mathcal{T} \quad \vee \quad \exists \mathcal{S}_n \in \{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\} : C \subseteq \mathcal{S}_n$$

where $\mathcal{T}$ is the Trump set and $\mathcal{S}_n$ is a natural suit partition.

From a validation perspective, a Throw represents a *compound requirement template*. While it is played as a single unit, the underlying verification engine must treat it as a collection of independent structural blocks $\mathcal{B}_L$ that the follower must match isomorphically. This introduces a complexity spike, as the engine must verify not just the existence of the cards, but the adherence to the internal structural density (pairs and sequences) within the batch.

3 Mathematical Problem Formulation

The arbitration of complex card plays in a multi-deck environment requires solving two distinct, yet interleaved, mathematical problems. The system must first resolve the **Non-Monotonicity** of the rank hierarchy and then verify **Isomorphic Structural Consistency** between the lead and follower plays.

In most trick-taking games, rank adjacency is a static property. However, the introduction of a dynamic "Current Rank" (CR) and a multi-tiered Trump partition induces a structural discontinuity in the standard linear sequence. Furthermore, when multiple structures are played simultaneously (a "Throw"), the problem expands from a simple value comparison to a combinatorial search for a valid partition of the follower's hand that satisfies the lead's structural template.

3.1 The Rank-Mapping Transformation $\Phi(c)$

To facilitate constant-time adjacency verification in a discontinuous rank-space, we define a transformation function $\Phi : C \to \mathbb{Z}^+$. This mapping projects each card c into a normalized integer domain where sequential structural primitives (Tractors) can be identified via simple arithmetic.

3.1.1 Mathematical Formalization

Let $p(c)$ be the raw precedence of a card rank based on the set $\{2, 3, \ldots, A\}$, mapped to $\{1, \ldots, 13\}$. The transformation is defined piecewise based on the card's membership in the Trump ($\mathcal{T}$) or Natural ($\mathcal{N}$) partitions:

1. **Trump Set $\mathcal{T}$:**
   - Supreme Trumps: $\Phi(c) \in \{27, 28\}$.
   - CR Parity: $\Phi(c) \in \{25, 26\}$ based on $suit(c) = S_m$.
   - Main Suit Residuals: $\Phi(c) = p(c) + 12 - \mathbb{1}(p(c) > p_{CR})$.
2. **Natural Set $\mathcal{N}$:**
   - Natural Adjacency: $\Phi(c) = p(c) - \mathbb{1}(p(c) > p_{CR})$.

3.1.2 Algorithmic Implementation

The following algorithm illustrates the normalization of card values, ensuring $O(1)$ adjacency verification across suit boundaries and rank displacements.

Algorithm 1: Rank Normalization $\Phi(c)$

Input: Card c, Current Rank CR, Main Suit S_m
Output: Normalized Rank Value Φ

$p \leftarrow \text{precedence}(c.rank)$
$p_{CR} \leftarrow \text{precedence}(CR)$
if S_m *is active* **then**
  if $c.suit == \text{'R'} \vee c.suit == \text{'B'}$ **then**
    return $c.suit == \text{'R'} \ ? \ 28 : 27$
  else if $c.rank == CR$ **then**
    return $c.suit == S_m \ ? \ 26 : 25$
  else if $c.suit == S_m$ **then**
    return $p + 12 - (p > p_{CR} \ ? \ 1 : 0)$
`// Natural suit or No-Trump logic`
if $p > p_{CR}$ **then**
  return $p - 1$
return p

3.2 Permutative Isomorphism Matching

Following the lead's play P_L, the system generates a multiset of required structural blocks $\mathcal{B}_L = \{b_1, b_2, \ldots, b_n\}$ via maximal decomposition. The validation of the follower's play P_F is modeled as a search for a valid structural mapping. We define P_F as a multiset of cards which the system must attempt to partition to satisfy $\mathcal{B}_L$.

Formal Existence Criterion

A play P_F is isomorphically adherent to P_L if and only if there exists at least one permutation of the required blocks $\sigma(\mathcal{B}_L)$ that can be successfully mapped to the cards in P_F. Formally, P_F is valid if:

$$\exists \sigma \in S_n : \forall j \in \{1, \ldots, n\}, \quad \text{Match}(b_{\sigma(j)}, R_j) \neq \emptyset$$

where:

- S_n is the **Symmetric Group** of degree n, representing the set of all $n!$ possible permutations (orderings) of the lead's structural blocks.
- $b_{\sigma(j)}$ is the j-th structural block (e.g., a Tractor of length 3) under the specific permutation σ.
- $R_j = P_F \setminus \bigcup_{k=1}^{j-1} m_k$ is the **residual pool** of cards remaining in the follower's hand after the first $j-1$ blocks have been satisfied.
- m_k is the specific subset of cards consumed from P_F to satisfy block $b_{\sigma(k)}$.

By iterating through S_n, the algorithm ensures that if there is any possible way to "slice" the follower's hand to match the lead's template, it will be found. If the condition fails for all $\sigma \in S_n$, the play is non-conforming, as no structural interpretation of the follower's hand satisfies the lead.

4 Algorithmic Verification

4.1 Maximal Structural Decomposition

The validation of a composite play (a "Throw") requires reducing a multiset of cards C into its most significant structural components. This is achieved through a **Greedy Descending Decomposition** algorithm. This approach is necessitated by the **unidirectional recursive nature** of the structural matching problem: while a higher-order primitive (e.g., a Tractor) can be decomposed to satisfy lower-order requirements (Pairs or Singles) during the validation phase, the inverse is logically impossible. Therefore, the lead template must be defined by its maximal possible primitives to establish the absolute "upper bound" of structural strictness for the trick.

Formally, the decomposition partitions C into three disjoint sets:

$$C = \mathcal{P}_T \cup \mathcal{P}_P \cup \mathcal{P}_S$$

The algorithm executes the following procedural steps:

1. **Tractor Extraction ($\mathcal{P}_T$):** The system performs a greedy search for the longest possible contiguous sequence of pairs. Let $V = \{\Phi(c) \mid c \in C, \text{count}(c) = 2\}$. The algorithm seeks the largest L such that a sequence $\{v, v+1, \ldots, v+L-1\} \subseteq V$ exists. Once a Tractor is identified, those cards are removed from the pool before searching for smaller L.

2. **Pair Identification ($\mathcal{P}_P$):** Any remaining cards with multiplicity $k = 2$ that were not part of a Tractor are categorized as independent Pairs.

3. **Singular Residuals ($\mathcal{P}_S$):** All remaining cards are categorized as Singles ($k = 1$).

Decomposition as Search Space Reduction

This maximal extraction serves as a form of **Lexicographical Pre-processing**. By identifying the largest possible blocks first, the system minimizes the cardinality n of the requirement multiset $\mathcal{B}_L$. This is a critical optimization for the subsequent permutative matching phase; since the follower's search space grows at $O(n!)$, reducing n through greedy decomposition ensures that even complex 25-card throws are reduced to a small number of high-order blocks. This keeps the effective search space computationally trivial while maintaining the recursive flexibility to "degrade" these large blocks if the follower's resources are fragmented.

4.1.1 Algorithmic Implementation

The extraction process uses a state-tracking array to ensure no card is assigned to multiple blocks. The engine iterates through potential sequence lengths L in descending order, ensuring that Tractors are prioritized over independent Pairs and Singles.

Algorithm 2: Greedy Block Extraction (`getStructure`)

Input: Sorted card array C, Total cards n
Output: Structural Template $\mathcal{B}_L$ (Tractors, Pairs, Singles)

$used[1 \dots n] \leftarrow false$
`// Phase 1: Tractor Extraction (Length` $L \geq 4$`)`
for $len \leftarrow n$ ***down to*** 4 ***step*** -2 **do**
  for $i \leftarrow 0$ ***to*** $n - len$ **do**
    if $used[i \dots i + len - 1]$ *contains true* **then**
      continue
    end
    if *isSequenceOfPairs(*$C[i \dots i + len - 1]$*)* **then**
      Add Tractor of length len to $\mathcal{B}_L$
      Mark $used[i \dots i + len - 1] \leftarrow true$
    end
  end
end
`// Phase 2: Pair Identification`
for $i \leftarrow 0$ ***to*** $n - 2$ **do**
  if $\neg used[i] \wedge \neg used[i + 1] \wedge isPair(C[i], C[i + 1])$ **then**
    Increment Pair count in $\mathcal{B}_L$
    $used[i], used[i + 1] \leftarrow true$
  end
end
`// Phase 3: Singular Residuals`
Count all $used[j] == false$ as Singles in $\mathcal{B}_L$
return $\mathcal{B}_L$

4.2 Systemic Isomorphic Validation

The validation of a follower's play is executed as an exhaustive search for structural conformity. Rather than employing a predictive heuristic, the system explores the full search space of permutations $\sigma \in S_n$. This ensures that if any valid structural mapping between P_F and P_L exists, the system will identify it, effectively resolving the limitations of a standard greedy matching approach.

The Permutative Search Algorithm

The function `sameStructure` operates as a **Recursive Backtracking Search** over the search space S_n. It does not assume a fixed partition of P_F, but instead treats the follower's play as a dynamic resource pool that undergoes *Structural Degradation* as requirements are processed.

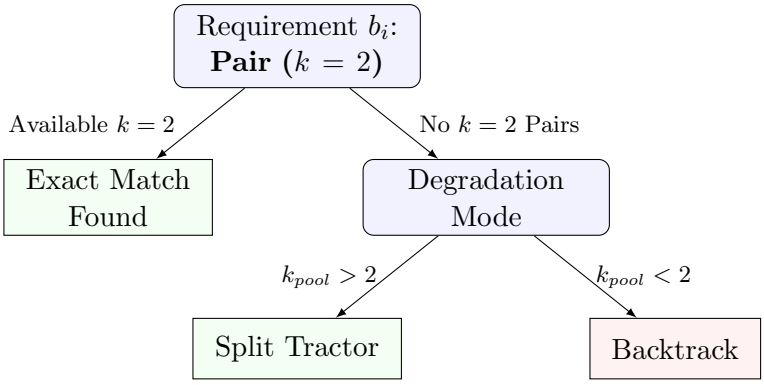

Figure 1: Recursive degradation logic for a single structural requirement b_i. If no exact match exists, the system attempts to decompose higher-order primitives (Tractors) before triggering a backtrack to explore alternative permutations of the requirements.

1. **Search Space and Permutation Reset:** The algorithm iterates through permutations of the lead's template $\mathcal{B}_L$. For each new permutation $\sigma \in S_n$, the follower's card pool is **fully restored to its original state**. This ensures that every ordering of requirements—from most specific (Tractors) to most general (Singles)—is tested against a complete resource set.

2. **Recursive Structural Degradation:** The core matching logic is inherently recursive. To satisfy a block b_i, the system executes a prioritized search:

   - **Exact Match:** If the current pool P_F contains a primitive identical to b_i, it is consumed.
   - **Degradation (Recursive Case):** If an exact match is missing, the system attempts to "break down" a higher-order primitive. For example, a Pair requirement may be satisfied by extracting two cards from a Tractor of any length L. The consumed cards are removed from the active pool, and the remaining fragments are returned as lower-order residuals.
   - **Base Case (Singularization):** Any requirement can ultimately be satisfied by the base case of $k = 1$ (Singles), provided the follower has sufficient card volume within the suit partition.

3. **Failure and Global Backtracking:** If a specific block b_i cannot be satisfied under the current permutation σ, the algorithm terminates that branch. It resets the follower's hand to its full initial state and proceeds to the next permutation $\sigma + 1$ in the search space.

4. **Existence Check:** If any branch of the recursion successfully satisfies all n requirements, the existence criterion is met, the play is marked **Conforming**, and the search terminates immediately.

This recursive approach ensures that the algorithm handles the "25 Singles" case and complex "Tractor" throws with equal precision. While the former involves a high n, the structural homogeneity of the blocks allows the recursion to find a valid path in the first branch, effectively reducing the computational burden to a linear pass $O(M)$.

Conformity Failure

If the algorithm iterates through every $\sigma \in S_n$ and fails to find a single valid mapping, the play is deemed **Non-Conforming**. This result implies that no structural interpretation of P_F can

satisfy the lead's template. In this state, the follower is disqualified from winning the trick, regardless of the rank values (Φ) of the cards played. This approach ensures that the "shape" of the play is treated as a hard logical constraint before any honor-based resolution occurs.

The function `sameStructure` iterates through these levels to confirm that P_F is a valid isomorphic representation of P_L. If the system determines that a follower's hand could have satisfied a higher-priority structural match but failed to do so, the play is flagged as non-conforming, and the player is excluded from the dominance resolution for that trick.

4.2.1 Algorithmic Implementation

The following procedure represents the core logic of the `sameStructure` engine. It utilizes a canonical sort of the lead blocks followed by a permutative expansion to ensure every possible structural interpretation is evaluated.

Algorithm 3: Permutative Matching Engine (`sameStructure`)

Input: Lead Template $\mathcal{B}_L$, Follower Template $\mathcal{B}_F$
Output: Boolean (Structural Conformance)

$S \leftarrow \text{Linearize}(\mathcal{B}_L)$ `// Combines Tractors, Pairs, and Singles`
Sort S in descending order to establish a canonical starting permutation
repeat
  $(T_{pool}, P_{pool}, S_{pool}) \leftarrow \text{ResetResources}(\mathcal{B}_F)$
  $match \leftarrow true$
  foreach $need \in S$ **do**
    if $need \geq 4$ **then**
      `// Attempt to satisfy requirement from follower tractors`
      if $\neg$ *ConsumeFromTractors*$(T_{pool}, need)$ **then**
        $match \leftarrow false$; **break**
      end
    else if $need == 2$ **then**
      if $P_{pool} > 0$ **then**
        $P_{pool} \leftarrow P_{pool} - 2$
      else if *ConsumeFromTractors*$(T_{pool}, 2)$ **then**
        continue
      else
        $match \leftarrow false$; **break**
    else
      if $S_{pool} > 0$ **then**
        $S_{pool} \leftarrow S_{pool} - 1$
      else if $P_{pool} > 0$ **then**
        $P_{pool} \leftarrow P_{pool} - 1$
      else if *ConsumeFromTractors*$(T_{pool}, 1)$ **then**
        continue
      else
        $match \leftarrow false$; **break**
  end
  if *match* **then**
    return true
  end
until *next permutation of S does not exist*
return false

4.2.2 Hierarchical Resource Extraction

The matching engine relies on a secondary subroutine, `ConsumeFromTractors`, to manage the degradation of high-order primitives. This function implements a First-Fit strategy to satisfy requirements that cannot be met by exact matches in the follower's pool of pairs or singles. By modifying the length of available tractors in-place within each search branch, the algorithm maintains an accurate state of the follower's "residual" resources.

Algorithm 4: Resource Fragmentation (`ConsumeFromTractors`)

Input: Tractor Pool T, Requirement Cardinality n_{req}
Output: Boolean (Success)

```
foreach T_j ∈ T do
    if length(T_j) ≥ n_req then
        length(T_j) ← length(T_j) − n_req        // Consume and fragment
        return true
    end
end
return false      // No resource of sufficient magnitude exists
```

4.2.3 Pruning via Symmetry Breaking

By integrating a `break` statement within the requirement loop, the implementation leverages **Symmetry Breaking** to optimize the search. If a specific requirement cannot be satisfied within a given permutation, the engine immediately prunes that branch of the search tree. This ensures that the factorial search space remains computationally manageable even for maximal $M = 25$ plays.

5 Complexity and Physical Constraints

5.1 Maximal Structural Decomposition Complexity

The decomposition of a play C into its requirement multiset $\mathcal{B}_L$ is the primary preparation step for isomorphic matching. This phase is characterized by a **"complexity compression"** effect: as the number of cards M increases, the greedy extraction of Tractors typically reduces the total block count n, effectively pruning the downstream search space S_n.

Computational Overhead and Domain Sorting

The total complexity for template generation is $O(M \log M)$, dominated by the sequence sorting phase. While a naive approach to identifying maximal contiguous sequences (Tractors) in a multiset could result in a $O(2^M)$ subset search, the implementation of a standard sort in the normalized domain Φ transforms the problem into a linear scan.

Table 2: Algorithmic Complexity of Decomposition Phases

Phase	Operation	Complexity
Rank Projection	Mapping $c \to \Phi(c)$ for M cards	$O(M)$
Sequence Sorting	Ordering projected values for adjacency	$O(M \log M)$
Tractor Extraction	Greedy linear scan for $L \geq 2$ sequences	$O(M)$
Residual Partitioning	Assigning remaining $k = 2$ and $k = 1$ sets	$O(M)$

Lexicographical Optimization and Search Pruning

The $O(M \log M)$ sorting phase of the raw card data serves a dual purpose beyond simple adjacency detection. By establishing a canonical order for the input multiset P, it enables the **Maximal Greedy Extraction** of structural blocks.

This pre-processing ensures that the decomposition engine (`getStructure`) consistently identifies the largest possible primitives, such as Tractors of the greatest available length. By maximizing the structural density of each block, the algorithm effectively minimizes the total cardinality n of the requirement multiset $\mathcal{B}_L$. Since the matching engine's complexity is defined by the Symmetric Group S_n, this reduction of n is the primary mechanism for preventing combinatorial explosion. Consequently, even for maximal plays where $M = 25$, the consolidation of cards into high-order blocks ensures the $O(n!)$ permutative search remains computationally trivial within the physical constraints of the game.

5.2 Physical Bounds and Permutative Search

The search space for structural validation, $|S_n| = n!$, is governed by the number of structural blocks n. Formally, the problem of finding a valid mapping between P_F and $\mathcal{B}_L$ under structural degradation is a variation of the **Multiset Partitioning Problem**, which is known to be **NP-hard**. In a generalized environment where $M \to \infty$ and card multiplicity k is unbounded, the number of ways to partition P_F to satisfy $\mathcal{B}_L$ grows exponentially.

However, the specific implementation in Tractor operates within a **bounded parameter space**. Although the theoretical card limit per suit partition is $M = 25$, the number of *distinct* structural blocks n is physically constrained by the deck's multiplicity ($k = 2$) and the greedy compression of Tractors during decomposition.

Even in highly complex "Mixed Plays" where n might approach 10, the computational overhead remains insignificant. As shown in Table 4, the search space for $n = 10$ is roughly 3.6×10^6 permutations, a task modern CPUs resolve in milliseconds.

Table 3: Search Space Characteristics Relative to Structural Heterogeneity ($M = 25$)

Lead Composition	**Blocks (n)**	**Search Space $\lvert S_n \rvert$**	**Permutative Complexity**
25 Singles	25	1.55×10^{25}	$O(1)$†
12 Adjacent Pairs	1	1	$O(1)$
Mixed Complex Play	5–8	120 – 40,320	$O(n!)$
Theoretical Worst Case	10	3,628,800	$O(n!)$

† Note: While $n = 25$ for singles, the homogeneity of the block types collapses the search space into a single equivalence class. As the recursive base case, validation reduces to a $O(1)$ cardinality check against the follower's suit partition, effectively bypassing the permutative search.

The total algorithmic complexity $O(M \log M + n!)$ results in a deterministic validation time. By acknowledging the factorial nature of the search while demonstrating its containment within physical bounds, the engine achieves real-time $O(1)$ latency relative to human perception during game arbitration.

Appendix: Algorithmic Visualizations

A.1 Systemic Pipeline

The following flowchart illustrates the transformation of raw card data into a validated structural match through the normalized domain Φ.

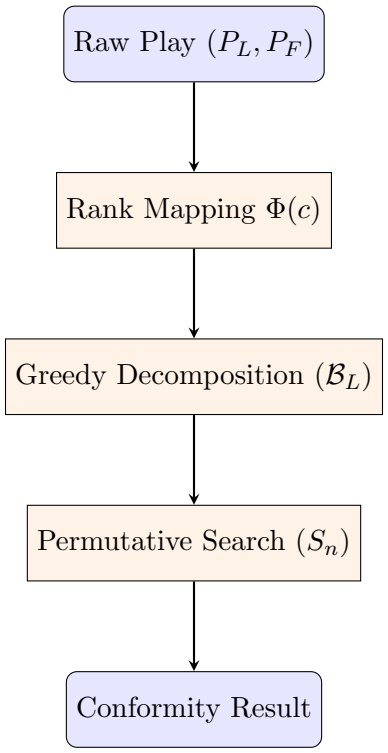

Figure 2: The verification pipeline from input to trick arbitration.

A.2 Permutative Backtracking Search Tree

This diagram represents the search space for a lead of $n = 3$ blocks. If a specific branch fails to satisfy the structural requirements, the algorithm backtracks to the previous decision node and attempts the next permutation in S_n.

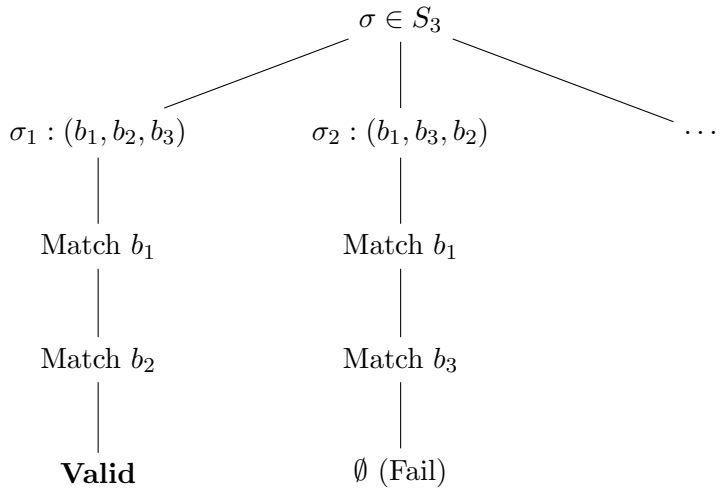

Figure 3: Visualization of the permutative existence check search tree.

References

[1] Sphere Online Judge. *TRACTOR - Game Simulator*. [Online]. Available: `https://www.spoj.com/problems/TRACTOR/`.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. *(Reference for Multiset Partitioning and Exact Cover)*.

[3] D. E. Knuth, *The Art of Computer Programming, Vol. 4, Fascicle 2: Generating All Permutations*. Addison-Wesley, 2005.

[4] S. Fortin, *The Graph Isomorphism Problem*. Technical Report, University of Alberta, 1996.

[5] T. H. Cormen et al., *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[6] M. Buro and J. R. Long, "Efficient Play and Analysis of Trick-Taking Games," in *Proc. IJCAI*, 2009.