Formalising Binary Linear Constraint System Games in Lean 4

Sean Perazzolo
EPFL, supervised by Prof. Thomas Vidick
sean.perazzolo@epfl.ch

June 2026

Abstract

This project presents a Lean 4 formalisation of binary Linear Constraint System (LCS) games and their quantum strategies. It provides the core definitions for LCS games over F_2, together with two complementary quantum strategy formalisms: a projector-based formalism, where strategies are described by projective measurement systems for the players, and an observable-based formalism, where strategies are described by self-adjoint involutive operators satisfying the relevant commutation relations. The development also includes a bridge between these two formalisms, allowing strategies to be translated from one description to the other.

On the analytic side, the formalisation develops local winning and local loss operators for binary LCS games and proves a sum-of-squares decomposition of the local loss operator. This is then combined with an EPR-state argument to extract row identities for observables arising from bipartite strategies. On the algebraic side, the project defines the solution group of a binary linear system and shows how these row identities induce a matrix representation of the corresponding solution group.

The Mermin-Peres Magic Square game serves as the main case study, illustrating how the abstract framework can be instantiated by an explicit quantum strategy.

Overall, the project provides a machine-checked Lean 4 development connecting binary LCS games, quantum strategies, and solution group representations.

Keywords: Lean 4, Formal verification, Linear Constraint System Games, Quantum strategies, Solution groups, Mermin-Peres magic square game

1 Introduction

1.1 Quantum Non-Local Games

In quantum information theory, non-local games are a standard formalism for studying quantum entanglement and non-locality. Non-locality refers to the ability of quantum systems to exhibit correlations that cannot be explained by local hidden-variable models, or equivalently by the principle of local realism. In a standard non-local game, two or more cooperating players are physically separated and forbidden from communicating once the game begins. They receive inputs from a referee and must produce outputs satisfying a prescribed winning condition. While classical strategies for non-local games are limited to correlations achievable by shared randomness, quantum strategies can leverage entanglement to achieve higher winning probabilities, often surpassing classical limits. This setting makes it possible to compare the correlations achievable by classical and quantum resources, providing a precise way to study the difference between the two.

1.2 Historical Context and Significance

The study of such games is rooted in the foundations of quantum mechanics. The Einstein-Podolsky-Rosen (EPR) paradox, proposed in 1935 [1], challenged the completeness of quantum theory by arguing that its predictions suggested an unacceptable form of long-range influence, famously described as "spooky action at a distance." In 1964, Bell's theorem showed [2] that no local hidden-variable theory can reproduce all quantum predictions, establishing non-locality as a central feature of quantum theory rather than a purely philosophical curiosity. Since then, non-local games have become a standard language for expressing and analysing this phenomenon. They also play an important role in quantum information theory, for instance in device-independent quantum cryptography, where security guarantees are derived from the violation of classical bounds without relying on assumptions about the internal structure of the devices used.

1.3 Linear Constraint System Games

Within this broad class, Linear Constraint System (LCS) games ([3], [4]) form a particularly simple and structured family. Instead of arbitrary input-output rules, the winning conditions in an LCS game are determined by a system of linear equations over a finite field, most often in the binary setting over F_2. In such a game, the first player, conventionally called Alice, is asked for an assignment to the variables appearing in a given equation, while the second player, Bob, is asked for the value of a single variable appearing in that equation. The players win if Alice's assignment satisfies the chosen equation and if Bob's answer agrees with Alice's value on the queried variable. Because their underlying structure is rooted in linear algebra and group theory, LCS games offer a mathematically elegant setting in which to study quantum advantage.

1.3.1 Mermin-Peres Magic Square Game

Canonical examples such as the Mermin-Peres magic square game illustrate the strength of this framework particularly well: they exhibit quantum pseudotelepathy, a phenomenon in which players sharing entanglement can satisfy the constraints with certainty even though no classical strategy can win perfectly.

1.4 Contributions and Scope

This project makes the following contributions to the formalisation of binary Linear Constraint System games in Lean 4:

- It formalises the core definitions of binary Linear Constraint System games, including layouts, assignments, games, and explicit binary linear systems over F_2.
- It develops two quantum strategy formalisms: a projector-based formalism using projective measurement systems, and an observable-based formalism using self-adjoint involutive operators.
- It implements a bridge between these two formalisms, allowing observable strategies to be translated into projector strategies.
- It defines local and global winning/loss operators for binary LCS games and proves a sum-of-squares decomposition of the local loss operator, which is then used in the EPR-state argument.
- It formalises an EPR-state argument that extracts row identities from local-loss annihilation in the bipartite setting.
- It defines the solution group of a binary linear system and constructs matrix representations of this group from the previously derived row identities.
- It instantiates the general framework on the Mermin-Peres Magic Square game as the main case study.

1.4.1 Repository and Documentation

Source

The project source code is publicly available on GitHub, accessible via
sean.perazzolo.ch/lcs/source.

Documentation

In addition, to make the project easier to navigate, API documentation is available at
sean.perazzolo.ch/lcs/documentation.

Report

A copy of this report is hosted on the project website at *sean.perazzolo.ch/lcs/report*.

2 Approach

2.1 Structure of the Lean Development

The formalisation is organised as follows:

- The foundational definitions are introduced in `LCS/Basic.lean`, which defines layouts, games, and explicit binary linear systems.
- The strategy layer is developed in `LCS/Strategy`, with separate modules for projector-based strategies, observable-based strategies, and the bridge between them.
- The main proof-oriented part of the project is then divided into several components.
  - `LCS/WinningCondition.lean` defines the winning and loss operators and proves the sum-of-squares decomposition of the local loss operator.
  - `LCS/EPR.lean` extracts matrix identities from local-loss annihilation on the EPR state.
  - `LCS/SolutionGroup.lean` together with `LCS/SolutionGroup/Representation.lean` define the binary solution group and construct its matrix representations.
- Finally, the abstract framework is instantiated in `LCS/Games/MagicSquare`, which develops the Mermin-Peres Magic Square game as the main case study.

2.2 Building the Documentation

This project documentation is built with *doc-gen4*, the standard Lean documentation tool. It processes the doc-comments attached to lemmas and sections, rendering the mathematical content while hiding the proof details. This is the same tool used for the official mathlib documentation.

2.3 Linear Constraint System Games Formalisation

We begin by formalising Linear Constraint System games.

2.3.1 The Mathematical Object

A linear constraint system over a field K consists of a finite family of variables $x_1, ..., x_s$ together with a finite family of r equations

$$\sum_{j=1}^{s} A_{ij} x_j = b_i \tag{1}$$

, where $A_{ij}, b_i \in K$.

In this project, we restrict to the binary setting over F_2. In this case, variables take values in $\{0, 1\}$ and the equations are evaluated modulo 2.

2.3.2 The Game

In the associated game, the referee selects an equation i and sends it to Alice, while Bob receives a variable j that appears in that equation.

Alice responds with an assignment of values to the variables appearing in the chosen equation, while Bob responds with a value for the queried variable.

The players win if Alice's assignment satisfies the chosen equation and if Bob's answer agrees with Alice's value on the queried variable.

2.3.3 Representation in Lean

We define an `LCSLayout` structure to represent the following data:

- the number of variables `s`,
- the number of equations `r`,
- the support of each equation, as a family of finite sets `V : Fin r → Finset (Fin s)`.

This support-based presentation records, for each equation, the set of variables that occur in it. In the binary setting, this amounts to taking the coefficients to be implicitly equal to 1 on the support of the equation, which is why the layout can be described just by these finite sets.

This structure does not capture the constants b_i on the right-hand side of the equations, since many constructions depend only on the incidence pattern of the variables in each equation.

```
structure LCSLayout where
  r : ℕ
  s : ℕ
  V : Fin r → Finset (Fin s)
```

Next, we define an `LCSGame` structure that extends `LCSLayout` by including the constants `b : Fin G.r → Fin 2`, which represent the right-hand side of the equations in the binary setting.

```
structure LCSGame (G : LCSLayout) where
  b : Fin G.r → Fin 2
```

Finally, for the group-theoretic constructions, we also define a `LinearSystem` structure as an alternative description of a binary LCS game. This structure consists of a coefficient matrix A and a right-hand side vector b.

```
structure LinearSystem where
  layout : LCSLayout
  A : Fin layout.r → Fin layout.s → Fin 2
  b : Fin layout.r → Fin 2
```

Any support-based game can be converted into such a system by taking $A_{ij} = 1$ when variable j appears in equation i, and $A_{ij} = 0$ otherwise.

The project therefore uses both a support-based description, via `LCSLayout` and `LCSGame`, and a matrix-based description, via `LinearSystem`, depending on which is more convenient for the construction at hand.

For a concrete example of these definitions, see the case study of the Mermin-Peres Magic Square game in Section 4.

2.4 Quantum Strategy Formalisms

2.4.1 Projector-Based Strategies

In a quantum strategy, a player's response to a question is not modelled as a deterministic function from questions to answers. Instead, the question determines which measurement the player performs on their share of quantum state, and the answer is then given by the outcome of that measurement. For this reason strategies are naturally described in terms of measurement systems, which are families of projective measurements.

In the binary LCS setting, Alice and Bob have different answer types. When Alice is asked an equation i, she must provide a full assignment to all variables appearing in that equation. In Lean, an assignment for equation `i` is defined simply as a function mapping each variable in V_i to a binary value:

```
def Assignment (G : LCSLayout) (i : Fin G.r) :
Type :=
  (j : G.V i) → Fin 2
```

Her measurement is therefore indexed by the set of assignments on that equation. When Bob is asked a variable j, he must provide a single bit, so his measurement is indexed by the two outcomes in F_2.

The project works with projective measurements. This means that for each question, the possible answers are indexed by a family of orthogonal self-adjoint idempotent operators summing to the identity, which represent the projectors onto the corresponding outcome subspaces.

Representation in Lean

In Lean, projective measurements are encoded by the predicate `IsMeasurementSystem`. For a finite family of operators `f : I → R`, this predicate expresses that the operators form a projective measurement: They sum to the identity, are self-adjoint, are idempotent, and are pairwise orthogonal.

```
structure IsMeasurementSystem
  {I : Type*} [Fintype I]
  (f : I → R) : Prop where
  sum_one      : ∑ x, f x = 1
  idempotent   : ∀ x, f x * f x = f x
  orthogonal   : ∀ x y, x ≠ y → f x * f y = 0
  self_adjoint : ∀ x, star (f x) = f x
```

Using this notion, the projector-based strategy formalism is defined by the structure `ProjectorStrategy`.

```
structure ProjectorStrategy
  (R : Type*) [Ring R] [StarRing R] [Algebra ℂ R]
  (G : LCSLayout) where
  E : ∀ i, (Assignment G i → R)
  F : Fin G.s → (Fin 2 → R)
  alice_ms : ∀ i, IsMeasurementSystem (E i)
  bob_ms   : ∀ j, IsMeasurementSystem (F j)
  commute  : ∀ i j α β, E i α * F j β = F j β * E i α
```

Here `E i` denotes Alice's projective measurement associated with equation i; it is the full family of operators indexed by all assignments `x : Assignment G i`. For a specific assignment x, the operator `E i x` is the projector onto the event that Alice answers exactly x when asked equation i. Similarly, `F j` denotes Bob's binary projective measurement associated with variable j, and for a bit `y : Fin 2`, the operator `F j y` is the projector onto the event that Bob answers y when asked variable j. The fields alice_ms and bob_ms assert that these families define projective measurements. Finally, the commutation condition expresses the

operator-theoretic separation between Alice and Bob: Alice's and Bob's measurement operators commute for all questions and outcomes.

Although this formalism is expressed in terms of projective measurements, the later development also makes systematic use of observables derived from these measurements. Their role is explained after the observable-based formalism has been introduced.

2.4.2 Observable-Based Strategies

While the projector-based formalism is the most direct way to describe quantum strategies, it is often more convenient to work with observables, which are self-adjoint operators whose spectral decomposition corresponds to the projective measurements. For this reason, the project also introduces an observable-based formalism.

In this setting, a strategy is described by one observable for each variable on Alice's side, and one observable for each variable on Bob's side. They must satisfy the commutation relation dictated by the structure of the game. On Alice's side, observables corresponding to variables appearing in the same equation must commute, so that their products are well defined independently of the order of multiplication. In addition Alice's observables must commute with all of Bob's observables, reflecting the spatial separation between the players.

Representation in Lean

Because the present project is restricted to binary outcomes, the observable formalism is also specialised accordingly. Rather than considering arbitrary observables, we work with self-adjoint involutions, which are exactly the operators arising from two-outcome projective measurements. This is encoded in Lean by the predicate `IsObservable`.

```
structure IsObservable (O : R) : Prop where
  involutive   : O * O = 1
  self_adjoint : star O = O
```

With this notion of binary observable in place, the project packages the observable description of a strategy into a structure `ObservableStrategy` :

```
structure ObservableStrategy
  (R : Type*) [Ring R] [StarRing R] [Algebra ℂ R]
[StarModule ℂ R]
  (G : LCSLayout) where
  alice_obs : Fin G.s → R
  bob_obs : Fin G.s → R
  alice_observable : ∀ j, IsObservable (alice_obs j)
  bob_observable : ∀ j, IsObservable (bob_obs j)
  sameEquation_comm :
    ∀ i, Pairwise (fun j k : G.V i ⇒ Commute
(alice_obs j.1) (alice_obs k.1))
  alice_bob_commute :
    ∀ j k, Commute (alice_obs j) (bob_obs k)
```

2.4.3 Bridge Between Projector and Observable-Based Strategies

The two formalisms are closely related, and the project provides translations in both directions. This bridge is essential because explicit examples are most naturally described with observables, while the loss operator and its decomposition are more naturally expressed with projectors.

From projectors to observables. Given a `ProjectorStrategy`, Bob's binary observables are simply the differences of his two projectors. Alice's observables are extracted by first collapsing an assignment-indexed measurement `strat.E i` to a binary one that only distinguishes the value of a single variable j (using `InducedMeasurementSystem`), and then applying the same difference formula. In both cases, the fact that the underlying family is a measurement system implies that the resulting operator is a self-adjoint involution. These derived operators are denoted `Alice_A` and `Bob_B`:

```
def ObservableOfMeasurementSystem (f : Fin 2 → R) :
R :=
  f 0 - f 1

def Alice_A
  (strat : ProjectorStrategy R G) (i : Fin G.r) (j :
G.V i) : R :=
  ObservableOfMeasurementSystem
(InducedMeasurementSystem (strat.E i) (fun x ⇒ x j))

def Bob_B (strat : ProjectorStrategy R G) (j : Fin
G.s) : R :=
  ObservableOfMeasurementSystem (strat.F j)
```

Note that Alice's derived observables are indexed by a pair (i, j) of an equation and a variable within it, whereas `ObservableStrategy` demands a single global observable A_j per variable. This mismatch is why the two formalisms do not perfectly round-trip.

From observables to projectors. Conversely, starting from an `ObservableStrategy`, the construction `ObservableStrategy_To_ProjectorStrategy` builds a `ProjectorStrategy` using the two spectral projectors of each observable. For Bob, the binary measurement is $F_{j,y} = \frac{1}{2}\big(I + (-1)^y B_j\big)$ for $y \in \{0, 1\}$. For Alice, the measurement projector attached to an equation i and assignment x is the product of the individual variable projectors along the support of that equation: $E_{i,x} = \prod_{j \in V_i} \frac{1}{2}\big(I + (-1)^{x_j} A_j\big)$. The row-wise commutation assumptions guarantee that these products are well defined, and the global commutation hypothesis ensures that all Alice–Bob commutators vanish.

```
noncomputable def
ObservableStrategy_To_ProjectorStrategy
  {R : Type*} [Ring R] [StarRing R] [Algebra ℂ R]
[StarModule ℂ R]
  {G : LCSLayout}
  (S : ObservableStrategy R G)
 :
  ProjectorStrategy R G := {
    E := AliceMeasurementFromObservables S
    F := BobMeasurementFromObservables S
    alice_ms :=
  aliceMeasurementFromObservables_isMeasurementSystem S
    bob_ms :=
  bobMeasurementFromObservables_isMeasurementSystem S
    commute := aliceMeasurement_bobMeasurement_commute S
  }
```

2.4.4 Bipartite Observable Strategies

For the final part of the project, the observable formalism is further specialised to a bipartite setting. This is the setting relevant for the EPR-state argument developed later, where Alice's and Bob's operators act on different tensor factors of a bipartite Hilbert space.

Instead of specifying two separate observable families from the start, the project begins with a single family of observables indexed by the variables of the game. These observables act on a space of the form `Matrix n n ℂ`. From this single family, one obtains Alice's and Bob's observables by lifting them to the two tensor factors: Alice's observable associated with a matrix `M` is `M ⊗ I`, while Bob's observable is `I ⊗ M`. In this way, commutation between Alice's and Bob's operators is automatic, since operators acting on different tensor factors commute.

In Lean, this data is encoded by the structure `BipartiteObservableStrategy`.

```
structure BipartiteObservableStrategy
    (n : Type*) [Fintype n] [DecidableEq n]
    (G : LCSLayout) where
  obs : Fin G.s → Matrix n n ℂ
  is_observable : ∀ j, IsObservable (obs j)
  sameEquation_comm :
    ∀ i, Pairwise (fun j k : G.V i ⇒ Commute (obs
j.1) (obs k.1))
```

Here `obs j` denotes the basic observable associated with variable `j`. The field `is_observable` asserts that each of these operators is a self-adjoint involution, while `sameEquation_comm` expresses the row-wise commutation required to form products of observables along a common equation.

From such a bipartite observable strategy, the project constructs an ordinary observable strategy by tensor lifting. Alice's observables are obtained by applying the map `M ↦ M ⊗ I`, and Bob's observables by applying `M ↦ I ⊗ M`. This construction is implemented by `toObservableStrategy`, and it provides the entry point from the bipartite setting into the general observable and projector formalisms developed earlier.

This specialisation is important because it matches the tensor-product structure of the EPR state used later in the project. In particular, it provides the framework in which local-loss annihilation on the EPR state can be turned into concrete matrix identities and, ultimately, into representations of the solution group.

2.5 Notation Used in the Sequel

From this point on, several sections use the local notation introduced in the Lean development for the operators attached to a projector-based strategy `strat` and a game `game`. This improves readability by allowing us to write concrete operator identities without having to refer to the underlying strategy and game structures at every step. The notation is as follows:

```
local notation "A[" i ", " j "]" ⇒ Alice_A strat i j
local notation "B[" j "]" ⇒ Bob_B strat j
local notation "E[" i ", " x "]" ⇒ strat.E i x
local notation "F[" j ", " y "]" ⇒ strat.F j y
local notation "b[" i "]" ⇒ game.b i
local notation "∏ₐ[" i "]" ⇒ Alice_Row_Prod strat i
```

- `A[i, j]` denotes the derived Alice observable `Alice_A strat i j`, attached to equation `i` and to a variable `j` appearing in that equation.
- `B[j]` denotes the derived Bob observable `Bob_B strat j`, attached to variable `j`.
- `E[i, x]` denotes the projector `strat.E i x` corresponding to Alice answering the assignment `x` on equation `i`.
- `F[j, y]` denotes the projector `strat.F j y` corresponding to Bob answering the bit `y` on variable `j`.
- `b[i]` denotes the right-hand side bit `game.b i` of equation `i`.

We also use the notation `∏ₐ[i]` for the product of Alice's derived observables along row `i`, implemented in Lean as `Alice_Row_Prod strat i`. Concretely, this reproduces the row product

$$\prod_{k \in V_i} A_k^{(i)} \tag{2}$$

, that is, the product of the observables attached to all variables appearing in equation `i`.

2.6 Winning Conditions and Local Loss

Once a binary LCS game and a projector-based strategy have been defined, the next step is to formalise the winning condition of the game and to derive the associated winning and loss operators. This is the point at which the right-hand side values b_i of the equations come into play, as the winning condition depends on whether an assignment satisfies the chosen constraint.

2.6.1 Winning and Loss Operators

For a fixed equation i, the set of winning assignments consists of the assignments whose parity matches b_i. This set is used to define the local winning operator for a pair (i, j) of an equation and a variable appearing in that equation. Intuitively, this operator collects exactly those outcomes for which Alice's assignment satisfies the equation and agrees with Bob's answer on variable j.

In the codebase, the notation `S[i]` is used as a shorthand for `winning_assignments game i`, namely the set of assignments on equation i whose parity matches b_i.

```
def winning_assignments (i : Fin G.r) : Finset
(Assignment G i) :=
  Finset.univ.filter (fun α ⇒ (∑ j : G.V i, (α j :
Fin 2)) = b[i])

noncomputable def local_winning_operator (i : Fin
G.r) (j : G.V i) : R :=
  ∑ x ∈ S[i], E[i, x] * F[j, x j]

noncomputable def local_loss_operator (i : Fin G.r)
(j : G.V i) : R :=
  1 - local_winning_operator game strat i j
```

The local loss operator is defined as the complement of the local winning operator. It is the central object in the analytic part of the project, since the sum-of-square decomposition is proved for this operator.

In addition to these local quantities, the project also defines the global winning and loss operators by averaging the local ones.

```
noncomputable def winning_operator : R :=
  ∑ i : Fin G.r, ∑ j : G.V i,
  let normalization : ℂ := (G.r * (G.V i).card : ℕ)
  (1 / normalization) • local_winning_operator game
strat i j

noncomputable def loss_operator : R :=
  1 - winning_operator game strat
```

2.7 EPR Extraction Framework

2.7.1 The EPR Vector

At this point, **the formalisation is specialised from the earlier abstract algebraic setting to finite-dimensional complex matrix algebras**. More precisely, the relevant operators act on a bipartite space of the form $\mathbb{C}^n \otimes \mathbb{C}^n$, represented in Lean by matrices of type `Matrix (n × n) (n × n) ℂ`.

The distinguished vector used in the project is the unnormalised maximally entangled vector.

$$|\Omega\rangle = \sum_a e_a \otimes e_a \tag{3}$$

Its importance lies in the symmetry with which it couples the two tensor factors: it allows operators acting on one side of the tensor product to be related to operators acting on the other side:

$$(M \otimes I)|\Omega\rangle = (I \otimes M^T)|\Omega\rangle \tag{4}$$

In Lean, the EPR vector is defined as follows.

```
noncomputable def eprVec
    (n : Type*) [Fintype n] [DecidableEq n] : (n × n)
→ ℂ :=
  fun ab ⇒ if ab.1 = ab.2 then 1 else 0

local notation "Ω" ⇒ eprVec n
```

This is simply the coordinate description of the vector $|\Omega\rangle$, written in the standard basis of the bipartite space. The vector is intentionally left unnormalised, since the later arguments only use annihilation and injectivity properties rather than norm considerations.

2.7.2 EPR Identities for Bipartite Operators

The key role of the EPR vector is that it turns relations on the bipartite space into ordinary matrix identities. Concretely, if `M` and `N` are complex matrices, then the action of `M ⊗ N` on $|\Omega\rangle$ can be computed explicitly, and vanishing of this action is equivalent to a matrix equation involving `M` and `N^T`.

The fundamental identity proved in the project is that $(M \otimes N)|\Omega\rangle = 0$ if and only if $MN^T = 0$. This gives the basic extraction principle used later in the development.

```
lemma kronecker_mulVec_epr_eq_zero_iff
    (n : Type*) [Fintype n] [DecidableEq n]
    (M N : Matrix n n ℂ) :
    (M ⊗ₖ N) *ᵥ (eprVec n) = 0 ↔
      M * Nᵀ = 0
```

The project also proves the corresponding affine variant, which is the form used when extracting identities from operators of the form $1 - M \otimes N$.

```
lemma one_sub_kronecker_mulVec_epr_eq_zero_iff
    (n : Type*) [Fintype n] [DecidableEq n]
    (M N : Matrix n n ℂ) :
    (1 - M ⊗ₖ N) *ᵥ (eprVec n) = 0 ↔
      M * Nᵀ = 1
```

Several specialised versions of this identity are then derived for the bipartite lift operations introduced earlier. These lemmas make it possible to replace operator equalities on the distinguished vector $|\Omega\rangle$ by concrete matrix equalities in the underlying `n × n` space.

This is the mechanism referred to in the project as the EPR extraction argument, and it forms the bridge between bipartite operator relations and the matrix identities used later in the representation-theoretic part of the development.

2.8 Defining the Solution Group of a Binary Linear System

The final algebraic object introduced in the project is the solution group associated with a binary linear system. In the binary LCS setting, this group packages the combinatorial structure of the constraints into a presented group whose generators correspond to variables and whose relations encode the equations of the system (see Section 4.2.3 of [5]).

More precisely, let

$$\sum_{j=1}^{s} A_{ij} x_j = b_i \tag{5}$$

be a binary linear system over F_2. Its solution group is generated by symbols `g_1, ... , g_s` together with a distinguished central element `J`, subject to the following relations:

- each generator `g_j` is an involution,
- `J` is an involution,
- `J` commutes with every `g_j`,
- whenever two variables appear in a common equation, the corresponding generators commute,
- for each equation `i`, the product of the generators corresponding to the variables appearing in that equation is equal to `J^(b_i)`.

In the binary setting, each generator is an involution and the distinguished element J satisfies $J^2 = 1$. This reflects the fact that the observables appearing in this project are ± 1-valued, and the row relation simplifies to the parity condition J^{b_i} with $b_i \in \{0, 1\}$.

In Lean, the solution group is defined from the structure `LinearSystem`, which provides both the coefficient matrix `A` and the right-hand side vector `b`. The generators are represented by an inductive type containing one constructor for the variable generators and one for the distinguished element `J`. The relations are then imposed by passing to the presented group generated by these symbols.

At this stage, the project works with `LinearSystem` rather than directly with `LCSGame`. This is because the solution-group presentation is most naturally phrased in terms of explicit coefficients and equation supports extracted from a coefficient matrix. In particular, the commuting and row-product relations are defined by reading off which variables occur in each equation from the matrix `A`, while the right-hand side vector `b` determines the exponent of the distinguished generator `J`.

In Lean, the presented group machinery from Mathlib expects the relators as a set of words in the free group, a predicate of type `Set (FreeGroup G)`. The definition of `solutionRelators` below therefore takes a word `w` in the free group on `SolutionGen S` and returns a proposition stating that `w` is equal to one of the concrete relator words. Each clause in the disjunction encodes a family of relations: the first two cover the involution relators, the next two cover the commutation relators (centrality and same-equation), and the last covers the equation relators.

```
inductive SolutionGen (S : LinearSystem) where
  | var : Fin S.layout.s → SolutionGen S
  | J : SolutionGen S

def solutionRelators (S : LinearSystem) : Set (FreeGroup
(SolutionGen S)) :=
  fun w ⇒
    (∃ j, w = involutionRel (genVar (S := S) j))
  ∨
    w = involutionRel (genJ (S := S))
  ∨
    (∃ j, w = commuteRel (genVar (S := S) j) (genJ (S :=
S)))
  ∨
    (∃ j k, j < k ∧ sameEquation S j k ∧
      w = commuteRel (genVar (S := S) j) (genVar (S := S)
k))
  ∨
    (∃ i, w = equationRelator S i)


abbrev SolutionGroup (S : LinearSystem) : Type :=
  PresentedGroup (solutionRelators S)
```

Thus the solution group is defined in Lean as the presented group on the generators `SolutionGen S`, quotiented by the set of relators `solutionRelators S` encoding the involution, commutation, centrality, and row-product relations. The project therefore treats the solution group as an explicitly presented algebraic object attached to a binary linear system. This group will later serve as the target of the representation-theoretic part of the development, where matrix identities extracted from the EPR argument are used to verify its defining relations.

3 Results

With the definitions and constructions described in the previous section, we can now formalise the results of interest, which follow the development in Arthur Mehta's thesis [5].

3.1 Sum-of-Squares Decomposition

The first main result is a sum-of-squares decomposition of the local loss operator. The proof in mathematical terms is described in section 4.7 of Mehta's thesis [5].

$$\begin{aligned} L_{i,j} &= 1 - \sum_{x,y:x\in S[i],y=x_j} E_{i,x}F_{j,y} \\ &= \frac{1}{8}\Bigg(\left(I - B_j A_j^{(i)}\right)^2 \\ &\quad + \left(I - (-1)^{b_i}\prod_{k\in V_i} A_k^{(i)}\right)^2 \\ &\quad + \left(I - (-1)^{b_i}\prod_{k\in V_i} A_k^{(i)} A_j^{(i)} B_j\right)^2\Bigg) \end{aligned} \tag{6}$$

Hence, the local loss vanishes if and only if each squared term vanishes individually, since each is positive semidefinite. This decomposition is a key step in the EPR extraction argument to produce the row identities.

In Lean, this decomposition is formalised by the following theorem:

```
theorem local_loss_sos (i : Fin G.r) (j : G.V i) :
  local_loss_operator game strat i j =
    (1/8 : ℂ) • (
      (1 - B[j] * A[i, j])^2 +
      (1 - (-1 : ℂ)^(b[i]).val • ∏ₐ[i])^2 +
      (1 - (-1 : ℂ)^(b[i]).val • (∏ₐ[i] * A[i, j] *
B[j]))^2
    )
```

This formalises that given a game and a projector-based strategy for it, the local loss operator can be decomposed as a sum of three squares.

The main challenges in the formalisation of this result were:

1) Noncommutative operator algebra. While the proof is mathematically elementary, Lean needs to be explicit about where multiplication is noncommutative and where it is safe to reorder terms. A lot of the work is proving and reusing commutation lemmas like :
   - Alice observables commute along a row.
   - Alice and Bob operators commute when needed.
   - The Row product commutes with relevant local observable.
2) Mixing scalar actions with operator multiplication A repeated source of complication was expressions involving both scalar multiplication and operator multiplication $(c \cdot X), (X * Y)$. The paper treats these transparently, but in Lean they require careful rewriting with lemmas like `smul_mul` and `mul_smul` to put the scalar factors in the right place.

3) Turning the paper sums into explicit finite sums in Lean. The proof uses sums over winning assignments and marginal slices of assignments. In Lean these are implemented using `Finset.filter`, `Finset.sum_congr` and fiberwise sums. Consequently, a significant part of the file is showing that the paper sums can be rewritten in terms of these more explicit constructions, and then manipulating them to get the desired final form.

3.2 Row Identities Extraction

The next step is to show that local-loss annihilation on the EPR state implies three local relations, which can then be extracted into identities on the underlying matrix space.

This is a two-step process.

1) First, using the sum-of-squares decomposition, we show that if the local loss operator annihilates the EPR state, then each SOS term annihilates $|\Omega\rangle$ individually. Writing

$$\begin{aligned} T_1 &= I - B_j A_j^{(i)}, \\ T_2 &= I - (-1)^{b_i} \prod_{k \in V_i} A_k^{(i)}, \\ T_3 &= I - (-1)^{b_i} \prod_{k \in V_i} A_k^{(i)} A_j^{(i)} B_j \end{aligned} \tag{7}$$

From the SOS decomposition, we get :

$$L_{i,j}|\Omega\rangle = 0 \Longrightarrow \frac{1}{8}(T_1^2 + T_2^2 + T_3^2)|\Omega\rangle = 0, \tag{8}$$

Each T_k is self-adjoint: this follows from the fact that the local Alice and Bob observables are self-adjoint, that the Alice observables appearing in the same row commute so that their product is again self-adjoint, and that the scalar factor $(-1)^{b_i}$ is real. Taking the Hermitian inner product with $|\Omega\rangle$ gives

$$\begin{aligned} \langle\Omega|(T_1^2 + T_2^2 + T_3^2)\Omega\rangle \\ = \langle T_1\Omega|T_1\Omega\rangle \\ + \langle T_2\Omega|T_2\Omega\rangle \\ + \langle T_3\Omega|T_3\Omega\rangle. \end{aligned} \tag{9}$$

Each summand is a norm square, hence a nonnegative real number. Since their sum is zero, each one must itself be zero, and therefore

$$T_1|\Omega\rangle = 0, \qquad T_2|\Omega\rangle = 0, \qquad T_3|\Omega\rangle = 0. \tag{10}$$

This is the positivity argument formalised in Lean.

```
lemma three_selfAdjoint_squares_mulVec_eq_zero
    (hT₁ : T₁ᴴ = T₁) (hT₂ : T₂ᴴ = T₂) (hT₃ : T₃ᴴ =
T₃)
    (h :
      (T₁ ^ 2 + T₂ ^ 2 + T₃ ^ 2) *ᵥ v = 0) :
    T₁ *ᵥ v = 0 ∧ T₂ *ᵥ v = 0 ∧ T₃ *ᵥ v = 0
```

2) Then, using the extraction lemmas introduced in the EPR framework, each annihilation relation is rewritten in bipartite form and converted into an ordinary matrix identity. These lemmas are applied to the three SOS terms after expressing Alice's operators as lifts $A \otimes I$ and Bob's operators as lifts $I \otimes B$.

For example, for the first term, one rewrites the consistency operator using $(I \otimes B_j)\left(A_j^{(i)} \otimes I\right) = A_j^{(i)} \otimes B_j$:

$$\begin{aligned} T_1|\Omega\rangle = 0 &\Longrightarrow \left(I - A_j^{(i)} \otimes B_j\right)|\Omega\rangle = 0 \\ &\Longrightarrow A_j^{(i)} \left(B_j\right)^T = I. \end{aligned} \tag{11}$$

Since B_j is an observable, it is an involution, and right-multiplying by $\left(B_j\right)^T$ yields:

$$A_j^{(i)} = \left(B_j\right)^T. \tag{12}$$

In Lean, these three extraction steps are packaged as separate lemmas, for the above relation the statement is as follows:

```
lemma consistency_of_epr_annihilates
    (i : Fin G.r) (j : G.V i)
    (A B : Matrix n n ℂ)
    (hCons :
      (sosConsistencyTerm strat i j) *ᵥ
        Ω = 0)
    (hAlice : Alice_A strat i j = bipartiteAliceLift
A)
    (hBob : Bob_B strat ↑j = bipartiteBobLift B)
    (hBobs : IsObservable B) :
    A = Bᵀ
```

Applying the same mechanism to the other two terms yields the three identities

$$\begin{aligned} A_j^{(i)} &= B_j^T, \\ \prod_{k \in V_i} A_k^{(i)} &= (-1)^{b_i} I, \\ (-1)^{b_i} \prod_{k \in V_i} A_k^{(i)} A_j^{(i)} B_j^T &= I. \end{aligned} \tag{13}$$

Together these give the row identities that are used in the construction of representations of the solution group.

3.3 Matrix Representations of the Solution Group

The final step of this project is to show that these row identities can be used to construct a matrix representation of the solution group of the binary linear system associated with the game, given a perfect quantum strategy for the game.

3.3.1 The Construction

Suppose a bipartite observable strategy with observables $O_1, \ldots, O_s$ achieves perfect play, so that the local loss operators annihilate the EPR state for every equation and every support element. By the extraction argument of the previous subsection, this yields the row identities

$$\prod_{j \in \text{ supp}(i)} O_j = (-1)^{b_i} I, \qquad \forall i. \tag{14}$$

The representation is then defined by mapping the generators of the solution group to concrete unitary matrices:

- each variable generator g_j is sent to the corresponding observable O_j,

- the distinguished central generator J is sent to the scalar matrix $-I$.

This assignment respects all four families of defining relations:

1) *Involutions.* Each observable satisfies $O_j^2 = I$ by definition, and $(-I)^2 = I$.
2) *Centrality.* The scalar matrix $-I$ commutes with every matrix.
3) *Same-equation commutation.* Observables appearing in a common equation commute, which is a hypothesis of the observable strategy formalism.
4) *Equation relators.* The row identity $\prod_{j \in \mathrm{supp}(i)} O_j = (-1)^{b_i} I$ is exactly the relation $\prod g_j = J^{b_i}$ under the assignment $g_j \mapsto O_j$, $J \mapsto -I$.

Since every relator in the presentation maps to the identity under this assignment, the universal property of the presented group guarantees that the assignment extends uniquely to a group homomorphism.

3.3.2 Formalisation in Lean

The first step is to define the generator-level assignment. Each solution-group generator is mapped to a concrete element of the unitary group `unitary (Matrix n n ℂ)`: variable generators are sent to their packaged observables, and `J` is sent to the scalar matrix $-I$.

```
noncomputable def solutionGroupGeneratorImage
    (obs : Fin S.layout.s → Matrix n n ℂ)
    (obs_is_observable :
      ∀ j, IsObservable (obs j)) :
    SolutionGen S → unitary (Matrix n n ℂ)
  | .var j ⇒
      observableMatrixUnitary (obs j)
        (obs_is_observable j)
  | .J ⇒ negOneMatrixUnitary
```

This map is then lifted to the free group on the generators via `FreeGroup.lift`, and the representation is obtained by showing that every relator in the solution group maps to the identity under this lift. The generic representation constructor takes as input a family of observables, proofs that they satisfy the same-equation commutation requirement, and proofs that each equation relator maps to the identity in the unitary group. It then applies Mathlib's `PresentedGroup.toGroup` to produce the group homomorphism.

```
noncomputable def solutionGroupRepresentation
    (obs : Fin S.layout.s → Matrix n n ℂ)
    (obs_is_observable : ∀ j, IsObservable (obs j))
    (hsame : ∀ {j k}, sameEquation S j k
      → Commute (obs j) (obs k))
    (hequation : ∀ i, FreeGroup.lift
      (solutionGroupGeneratorImage obs
obs_is_observable)
      (equationRelator S i) = 1) :
    SolutionGroup S →* unitary (Matrix n n ℂ) :=
  PresentedGroup.toGroup (lift_relators_eq_one  ... )
```

The hypothesis `hequation` requires that the lift of each equation relator evaluates to the identity in the unitary group. Internally, the proof dispatches the remaining three relator families (involutions, centrality, and commutation) automatically, since these follow from the observable axioms and the commutation hypothesis.

The project then provides a second, higher-level constructor that chains the entire pipeline from local-loss annihilation on the EPR state. Starting from the hypothesis that the local loss operator annihilates the EPR vector for every equation i and every support element j, the construction first extracts the row identities via `rowObservableProduct_eq_sign_of_local_loss`, and then passes these identities to the generic representation constructor above.

```
noncomputable def solutionGroupRepresentationOfEPRLoss
    (strat : BipartiteObservableStrategy n G)
    (hNonempty : ∀ i, Nonempty (G.V i))
    (hLoss : ∀ i (j : G.V i),
      (local_loss_operator game
          strat.toProjectorStrategy i j) *ᵥ
        (eprVec n) = 0) :
    SolutionGroup game.toLinearSystem
      →* unitary (Matrix n n ℂ)
```

This is the main end-to-end result of the project. It shows that any bipartite observable strategy achieving perfect play on a binary LCS game gives rise to a unitary representation of the associated solution group. The representation lands in the unitary group (rather than just the general linear group) because observables are by definition self-adjoint involutions, and the scalar matrix $-I$ is trivially unitary.

3.3.3 Main Formalisation Challenge

The formalisation of this step is roughly 600 lines long, which may seem surprising given that the mathematical argument is short: define the generator map, check the relators, invoke the universal property. The bulk of the formalisation is devoted to bridging between two different descriptions of the same algebraic object.

On the group-theoretic side, the equation relator for row i is a word in the free group on the generators `SolutionGen S`, constructed from the explicit equation support of the linear system. On the analytic side, the row identity extracted from the EPR argument is a matrix equation involving the ordered product of observables indexed by the game's row support. These two descriptions refer to the same underlying product, but they arise from different data structures: the relator is built from the `LinearSystem`'s coefficient matrix (via `eqSupport` and `equationWord`), while the row observable product is built from the `LCSGame`'s row support (via `orderedSupportProduct`).

The key bridge lemma `lift_equationRelator_of_rowIdentity` closes this gap. It shows that evaluating the free-group lift of the equation relator under the generator map yields the same matrix as the row observable product, so that the matrix identity $\prod_{j \in \mathrm{supp}(i)} O_j = (-1)^{b_i} I$ can be used directly to verify the relator. Proving this requires a chain of intermediate steps: converting the game's support set to the linear system's equation support, showing that the sorted list product agrees with the `Finset.noncommProd` used in the strategy layer, and carefully tracking the passage between the free-group word evaluation and the matrix product.

4 Magic Square Game Case Study

The Mermin-Peres Magic Square game [6] is the main concrete example developed in this project. It is a particularly natural case study for binary LCS games: the rules are simple to state, the contradiction for classical assignments is easy to understand, and the quantum strategy can be written as an explicit 3×3 grid of Pauli observables. For this reason it is often regarded as one of the most intuitive examples of this phenomenon.

4.1 The Game

The game is built from a 3×3 array of binary variables. The referee may ask for one of the three rows or one of the three columns, so there are six possible equation questions in total. Alice receives one of these six questions and must provide values for the three cells lying in that row or column. Bob receives one cell contained in Alice's question and must provide the value of that single cell.

The winning condition is the same as for any binary LCS game, and it has two parts. First, Alice's assignment must satisfy the parity rule attached to the row or column she was asked about. Second, Bob's answer must agree with Alice's value on the overlapping cell.

The parity rules of this game are as follows: all three row equations have even parity, the first two column equations have even parity, and the final column has odd parity. Equivalently, if the variables are denoted by

$$x_1, x_2, ..., x_9 \in F_2, \tag{15}$$

then the six equations are

$$\begin{aligned} x_1 + x_2 + x_3 &= 0, \\ x_4 + x_5 + x_6 &= 0, \\ x_7 + x_8 + x_9 &= 0, \\ x_1 + x_4 + x_7 &= 0, \\ x_2 + x_5 + x_8 &= 0, \\ x_3 + x_6 + x_9 &= 1. \end{aligned} \tag{16}$$

To see why no perfect classical strategy exists, observe first that any deterministic perfect classical strategy would have to define a single global value for each cell of the square. Bob answers one cell at a time, so his strategy fixes a bit for each variable, and perfect consistency forces Alice to use exactly those same values whenever that variable appears in a row or column question. Thus a perfect classical strategy would induce a global assignment satisfying all six parity equations simultaneously.

This is however impossible: if one sums the three row equations over F_2, each variable appears exactly once and the total right-hand side is 0. If one instead sums the three column equations, one obtains the same left-hand side, since the same nine variables appear exactly once again, but now the total right-hand side is 1. Hence the same quantity would have to be equal to both 0 and 1, a contradiction. Therefore no deterministic classical strategy can win perfectly, and hence no classical strategy can win with certainty. Nevertheless, quantum players sharing entanglement can satisfy the game conditions perfectly.

4.2 The Observable Grid

The standard quantum strategy for the magic square is given by a 3×3 grid of commuting two-qubit observables:

$$\begin{pmatrix} X \otimes I & I \otimes X & X \otimes X \\ I \otimes Y & Y \otimes I & Y \otimes Y \\ X \otimes Y & Y \otimes X & Z \otimes Z \end{pmatrix}. \tag{17}$$

Each entry is a self-adjoint involution, hence a binary observable with eigenvalues ± 1. The crucial structural facts are:

- the three observables in each row commute pairwise,
- the three observables in each column commute pairwise,
- the product along each row is I,
- the product along the first two columns is I,
- the product along the final column is $-I$.

These identities match the parity pattern of the game exactly. Because the observables in each row or column commute, they can be measured simultaneously. Moreover, the product constraint has the correct sign in each case: a row or column whose parity bit is `0` has product I, while the final column, whose parity bit is `1`, has product $-I$. In this way, the classical parity equations are replaced by operator identities with the same sign pattern.

4.3 The Support-Based Description in Lean

In the formalisation, the game is first encoded in the support-based language introduced earlier. The layout records only which variables occur in each equation, while the right-hand side vector records the parity bits.

For the magic square, the layout has 6 equations and 9 variables. The support function lists the three cells occurring in each row and each column. In Lean this is written as follows.

```
def magic_square_layout : LCSLayout := {
  r := 6
  s := 9
  V := fun i ⇒
    match i with
    | 0 ⇒ {0, 1, 2}
    | 1 ⇒ {3, 4, 5}
    | 2 ⇒ {6, 7, 8}
    | 3 ⇒ {0, 3, 6}
    | 4 ⇒ {1, 4, 7}
    | 5 ⇒ {2, 5, 8}
}
```

The corresponding game is obtained by specifying the right-hand side bits. Only the final column has odd parity, so only the last equation is assigned the value `1`.

```
def magic_square_game : LCSGame
magic_square_layout := {
  b := fun i ⇒ if i = ⟨5, by decide⟩ then 1 else 0
}
```

This illustrates the convenience of the support-based description. At this stage one only needs to specify the incidence pattern of the variables and the parity bit attached to each constraint. The resulting object is already enough to

state the game and to instantiate the general strategy and winning-condition framework.

4.4 Formalising the Grid Strategy

The observable grid itself is encoded as a function from the nine variable indices to 4×4 matrices, obtained as Kronecker products of the Pauli matrices and the identity.

```
def magic_square_grid :
  Fin 9 → Matrix (Fin 2 × Fin 2) (Fin 2 × Fin 2) ℂ
  | 0 ⇒ X  ⊗ₖ I2
  | 1 ⇒ I2 ⊗ₖ X
  | 2 ⇒ X  ⊗ₖ X
  | 3 ⇒ I2 ⊗ₖ Y
  | 4 ⇒ Y  ⊗ₖ I2
  | 5 ⇒ Y  ⊗ₖ Y
  | 6 ⇒ X  ⊗ₖ Y
  | 7 ⇒ Y  ⊗ₖ X
  | 8 ⇒ Z  ⊗ₖ Z
```

To turn this grid into a strategy, one must verify that it satisfies the structural conditions required by the observable formalism. The first condition is that each grid entry is an observable, that is, a self-adjoint involution. This is proved in Lean by reducing each case to the corresponding facts for the Pauli matrices and using the compatibility of the Kronecker product with these properties.

The second condition is that, for every equation of the game, the observables lying in that equation commute pairwise. For the magic square this means proving pairwise commutation along each row and each column. The proof uses the familiar Pauli commutation and anticommutation relations, packaged through elementary lemmas about Kronecker products. Once these six commutativity checks are established, they are assembled into a uniform statement saying that the observables associated with any equation of the layout commute pairwise.

The resulting theorem-level object is the strategy

```
noncomputable def Strat_merminPeres :
    BipartiteObservableStrategy (Fin 2 × Fin 2)
magic_square_layout where
  obs := magic_square_grid
  is_observable := magic_square_is_observable
  sameEquation_comm := MP_sameEquation_comm
```

Notably, the case study not only defines the standard magic square strategy, but also formally verifies that the observable grid satisfies all algebraic hypotheses required by the abstract formalism. This makes the magic square a genuine verified example of the general definitions introduced earlier.

4.5 From the Game to the Associated Linear System

Although the game is described initially in support form, the later group-theoretic part of the development works with an explicit binary linear system. For this reason, the support-based game is converted to a `LinearSystem` using the generic map introduced in `LCSGame.toLinearSystem`.

For the magic square this yields the explicit system whose coefficient matrix has one row for each row or column support, and whose right-hand side vector records the parity pattern. In the codebase this is defined by

```
def magic_square_system : LinearSystem :=
  magic_square_game.toLinearSystem
```

Thus the same concrete example appears in two complementary forms:

- as an `LCSGame`, convenient for the strategy and winning-condition constructions,
- as a `LinearSystem`, convenient for the solution-group construction.

4.6 The Solution Group for the Magic Square

Once the explicit binary linear system has been recovered, the generic solution-group construction can be instantiated directly. The magic-square solution group is simply the solution group attached to `magic_square_system`.

```
abbrev MPSolutionGroup := SolutionGroup
magic_square_system
```

The formalisation then extracts inspectable data from this system: the coefficient rows, the right-hand side vector, the supports of the equations, and the resulting list of relators in presentation form. This does not yet prove any new analytic property of the concrete strategy, but it shows that the abstract algebraic machinery developed earlier can be applied to a canonical and highly nontrivial example.

In this sense, the magic square plays two roles in the project. First, it provides a concrete observable strategy whose validity can be checked directly in Lean. Second, it provides a concrete binary linear system and hence a concrete solution group to which the general representation-theoretic constructions apply.

4.7 Achievements of the Case Study

The present formalisation of the Mermin-Peres game therefore establishes the following points.

- The game itself is encoded explicitly as a binary LCS game.
- The standard Pauli-grid construction is encoded explicitly as a family of observables.
- Lean verifies that this grid satisfies the observable conditions and the same-equation commutation requirements needed to define a valid strategy.
- The support-based game is converted to an explicit binary linear system.
- The associated solution group is instantiated and made inspectable in the concrete magic-square case.

4.7.1 Incomplete Aspects of the Case Study

The case study lacks a theorem stating that this particular concrete strategy is a perfect strategy for the magic square game.

More precisely, the project contains a general result showing that if a suitable strategy annihilates the EPR state through the local loss operators, then one can extract row identities

and construct a representation of the associated solution group. However, this EPR-annihilation hypothesis is not proved for the present concrete packaging of the magic-square strategy. Thus the case study currently verifies the strategy data and the associated algebraic structures, while stopping short of a complete formal proof of perfect play for this concrete example.

Even with this limitation, the magic square remains a useful and informative case study. It demonstrates that the abstract framework developed in the project is expressive enough to capture the most familiar example of quantum pseudotelepathy, and it provides a concrete benchmark against which the strategy, EPR, and solution-group layers of the formalisation can be understood.

5 Project Timeline

TABLE I
WEEK BY WEEK FORMALISATION AND DEVELOPMENT PROGRESS.

Week	Key Achievements
Weeks 1 & 2	Literature review of LCS games and the Mermin-Peres magic square; familiarisation with the Lean 4 proof assistant.
Weeks 3 & 4	Formalisation of LCS layouts and games; definition of projector-based and observable-based strategies.
Weeks 5 & 6	Formal proof of equivalence between observable and projector strategies; definition of local winning operators.
Weeks 7 & 8 *(Milestones 1 & 2)*	Formalisation of the sum-of-squares decomposition of local loss (Milestone 1) and verification of the Mermin-Peres Magic Square as a valid observable strategy (Milestone 2).
Weeks 9 & 10	Group-theoretic formalisation mapping LCS games to presented Solution Groups; derivation of global matrix identities via EPR annihilation.
Weeks 11 & 12 *(Milestone 3)*	Construction of the Solution Group representation via matrix homomorphisms.
Weeks 13 & 14	Writing the final report and preparing the codebase for submission.

6 Limitations and Future Work

6.1 Limitations

The present development has several important limitations.

- **Binary setting only.**
  The whole development is restricted to LCS games over F_2, and this choice is built in from the beginning through the support-based description of equations, which records only which variables appear and not general coefficients over an arbitrary field.

- **Finite-dimensional matrix setting for the EPR argument.**
  The extraction results are proved only after specialising to complex matrices, so the final EPR and representation arguments do not yet apply in a more abstract operator-algebraic or infinite-dimensional setting.

- **No full equivalence theorem between the two strategy formalisms.**
  The project constructs and uses the bridge from observable strategies to projector strategies, but it does not prove a complete round-trip equivalence showing that the two formalisms determine the same data in a canonical way. One concrete obstruction is that the observables recovered from a projector strategy on Alice's side are naturally indexed by a pair (i, j) of an equation and a variable in that equation, whereas `ObservableStrategy` is formulated with a single observable for each variable.

- **The symmetric bipartite lift is too restrictive to prove perfect play.**
  The `BipartiteObservableStrategy` wrapper used in the EPR part of the development starts from a single family of observables M_j and lifts it symmetrically to the two tensor factors: Alice uses $M_j \otimes I$ and Bob uses $I \otimes M_j$. However, the EPR extraction argument shows that achieving perfect play forces Alice's and Bob's observables to be transpose-related ($A_j = B_j^T$). For our symmetric lift, this would require every observable to be symmetric ($M_j = M_j^T$). Because of this, the current interface is too limited to represent perfect strategies that rely on non-symmetric observables. A more general two-family setup, where Alice applies M_j and Bob applies M_j^T, is needed to complete the perfect-play pipeline for concrete examples like the Magic Square strategy.

- **No direct computation with real or complex operator entries.**
  In Lean, real numbers are implemented in a proof-oriented way rather than as an efficient executable numeric type, so matrices with real or complex entries are well suited for exact reasoning but not for effective computation of concrete operator values.

6.2 Future Work

The project can be extended by addressing the limitations and/or adding more concrete examples. For instance, extending the formalisation to LCS games over arbitrary finite fields would be a natural next step, as well as completing the perfect-play proof for the magic square strategy.

More interestingly, the project could move toward **robust self-testing** results [4], by replacing the exact annihilation condition on the EPR state with an approximate version, and showing that this implies approximate versions of the row identities, which in turn can be used to show that the strategy is close to an ideal strategy in a suitable sense. This would require developing a robust version of the EPR extraction argument, which is a significant technical challenge but would be a very interesting direction for future work.

7 Conclusion

Overall, this project successfully formalises the core mathematical framework of binary Linear Constraint System games in Lean 4. By providing these foundational definitions and formalising several key results, including quantum strategy frameworks and matrix representations of the solution group, this work opens the door to verifying more advanced LCS game theory theorems in Lean.

8 AI Use Disclosure

The focus of this project is on the high-level formalisation architecture and theorem statements rather than the cleanliness of the proof bodies. Accordingly, the detailed Lean 4 proof steps and their execution were left to an AI coding assistant.

References

[1] A. Einstein, B. Podolsky, and N. Rosen, "Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?," *Physical Review*, vol. 47, no. 10, pp. 777–780, 1935, doi: 10.1103/PhysRev.47.777.

[2] J. S. Bell, "On the Einstein Podolsky Rosen Paradox," *Physics Physique Fizika*, vol. 1, no. 3, pp. 195–200, 1964, doi: 10.1103/PhysicsPhysiqueFizika.1.195.

[3] R. Cleve and R. Mittal, "Characterization of Binary Constraint System Games." [Online]. Available: https://arxiv.org/abs/1209.2729

[4] A. Coladangelo and J. Stark, "Robust self-testing for linear constraint system games." [Online]. Available: https://arxiv.org/abs/1709.09267

[5] A. Mehta, "Entanglement and Non-locality in Games and Graphs," Doctoral dissertation, 2021. [Online]. Available: http://hdl.handle.net/1807/106342

[6] N. D. Mermin, "Simple Unified Form for the Major No-Hidden-Variables Theorems," *Physical Review Letters*, vol. 65, no. 27, pp. 3373–3376, 1990, doi: 10.1103/PhysRevLett.65.3373.