Colophon

Title:

The Algebra Manual

Instructions for Disassembly and Reassembly

Purpose:

This document provides operational guidance for understanding algebra as a finite system of permissions rather than a collection of formulas. It is intended for readers who wish to engage algebra structurally, without reliance on rote memorization or performative cleverness.

This manual describes the syntax of algebraic transformation, not algebraic problem-solving.

Method:

The manual proceeds by isolating and examining the minimal permissions that generate algebraic activity. Each permission is treated as a lawful operation within a closed symbolic system. Examples are illustrative, not exhaustive.

Audience:

This manual assumes no special aptitude. It is written for users willing to proceed slowly and remain within the rules.

Status:

Released as a functional document.

Not a textbook.

Not a reference work.

Not a puzzle.

Date of Compilation:

2025

Archival Note:

This document is issued as part of the MPSoL instructional series concerned with symbolic systems that generate infinite structure from finite rulesets.

Contents

The Permissions (Nonlinear)

You May Name

You May Equate

You May Replace

You May Combine

You May Rearrange (when allowed)

You May Parse

You May Undo (when an inverse exists)

You May Repeat

Section 1 — You May Name

Algebra begins with permission to name.

This is not trivial.
Nothing happens before this.

To name is to designate a symbol and agree that it will stand in for something. The symbol does not need to resemble the thing it stands for. It does not need to explain it. It does not need to resolve it.

It only needs to hold a place.

When you write a letter—x, y, a, n—you are not assigning a value. You are reserving space. Naming creates a placeholder where something may later appear, or may remain absent.

This is the first act of algebra, and it is reversible only by refusal.

What Naming Allows

When you name, you gain several capabilities at once:

- You may refer to something without specifying it.
- You may speak about relationships before quantities.
- You may delay commitment.

Naming allows algebra to proceed without knowing where it will end.

For example, when you write:

Let x represent a number.

You have not said what the number is. You have not restricted it. You have not solved anything. You have simply made it possible to continue.

This is sufficient.

Naming Is Not Definition

Naming is often confused with defining. They are not the same.

A definition attempts to fix meaning.
A name merely points.

In algebra, names are intentionally thin. They carry no personality, no history, no narrative burden. A symbol should do as little as possible while remaining usable.

This is why algebra prefers letters.
Not because letters are elegant, but because they are light.

A good name does not distract.

What Naming Does Not Allow

Naming does not grant the following permissions:

- It does not assign a value.
- It does not guarantee existence.
- It does not imply equality.
- It does not authorize manipulation by itself.

If x is named, nothing may yet be done to x except referring to it.

Attempts to operate on a symbol before it is properly related to others often result in confusion. This is not algebra resisting you; it is algebra enforcing sequence.

You may name.
Then you must wait.

Common Misuse

The most common misuse of naming is premature interpretation.

Readers often look at a symbol and immediately ask:
“What is it?”
“What does it mean?”
“What number is it really?”

These questions are out of order.

Algebra is not offended by curiosity, but it is strict about timing. Meaning arrives later, through relationship.

Another common error is overnaming—introducing multiple symbols where one would suffice. This creates unnecessary bookkeeping and obscures structure.

Name sparingly.
You can always name again.

Minimal Example

Consider the statement:

Let x be a number.

At this stage, nothing further can be done. This is correct.

Now consider:

Let x be a number such that $x + 3 = 7$.

The important move here is still the first one. The equation comes later. Without naming x, there is no subject for the equation to act upon.

Naming does not solve the problem.
It makes solving possible.

Why Naming Comes First

Every algebraic system, no matter how advanced, begins by reserving symbolic space.

Groups, rings, fields, matrices, functions—each starts by naming elements before describing how they interact.

This is not tradition.
It is necessity.

You cannot relate what you cannot refer to.
You cannot transform what you have not marked.

Naming is the minimum act that makes structure visible.

A Note on Neutrality

Algebraic names are neutral by design.

They are not variables because they vary.
They vary because they are variables.

The symbol does not change itself. What changes is what is permitted to stand in its place later.

This neutrality is what allows algebra to move across domains without friction.

The name remains a name.

Summary of This Permission

You may name.

This means:
• you may introduce a symbol
• you may refer to it consistently
• you may delay commitment

You may not yet:
• assign value
• manipulate freely
• conclude anything

Naming opens the door.
It does not move you through the room.

That comes next.

Section 2 — You May Equate

After naming comes permission to equate.

Equating is the act of declaring that two expressions are the same. When you write an equals sign, you are asserting identity within the system.

This is a strong move. Algebra does not allow it casually.

What Equating Allows

When you equate two expressions, you gain the ability to treat them interchangeably.

If
$a = b$

then anywhere a appears, b may appear in its place.

This is the foundation of all algebraic motion. Without equality, symbols remain isolated. With equality, structure forms.

Equating turns names into participants.

The Equals Sign

The equals sign means “is the same as.”

In algebra,
$3 + 4 = 7$
is a statement of equivalence. The system records that the two sides balance.

What May Be Equated

You may equate:
• a symbol and a value
• two expressions
• two results of different operations

For example:
$x + 3 = 7$

Both sides belong to the same algebraic space.

Conditions on Equality

Equality requires that both sides belong to the same system.

Equating incompatible objects or unintroduced symbols produces statements that cannot be used. Such statements fail quietly.

Common Misuse

A frequent error is treating equality as directional.

Both sides carry equal weight.

Another error is stacking equalities without checking consistency. Carelessness here creates contradictions later.

Minimal Example

$x = 4$

$x + 3 = 7$

In both cases, a relationship has been set. Nothing has been solved yet.

Why Equating Comes Second

Naming creates a subject.
Equating creates a relationship.

Once equality exists, replacement becomes possible.

Summary of This Permission

You may equate.

This means:
• you may declare two expressions the same
• you may treat either side as interchangeable
• you may build chains of relationship

Equality establishes balance.
Movement comes next.

Section 3 — You May Replace

Replacement is the working motion of algebra.

Once two expressions have been equated, either may stand in for the other. This is what equality is for.

If something is the same, it may be used as the same.

What Replacement Allows

Replacement allows movement without change.

If
$a = b$

then anywhere a appears, b may appear instead. The reverse is also true.

Replacement lets algebra travel through an expression without altering its balance.

Replacement Is Local

Replacement does not require rewriting everything at once.

You may replace:
- a single symbol
- a part of an expression
- one side of an equation

You choose the location.

For example, if:
$x = 4$

and you are working with:
$x + 3$

you may replace x with 4 in that expression alone.

The rest of the system remains untouched.

Why Replacement Matters

Without replacement, equality would remain inert.

Replacement allows separate statements to interact. It is how constraints propagate.

This is where algebra begins to feel active.

Conditions on Replacement

Replacement must preserve equality.

You may only replace what has been equated with what it has been equated to.

Casual substitution breaks the chain.

Common Misuse

The most common error is replacing too much.

Replacing everywhere at once often produces expressions that are harder to work with.

Replacement is a permission, not an obligation.

Another error is replacing before equality has been established.

Replacement always follows equating.

Minimal Example

$x + 3 = 7$

You may replace $x + 3$ with 7 in any larger expression where it appears.

Or you may replace x with 4 after that equality has been established.

Each move is small.
Each move is reversible.

Replacement and Solving

Solving is a sequence of replacements guided by undo operations.

You replace until nothing remains to be replaced.

Summary of This Permission

You may replace.

This means:
• you may substitute equals for equals
• you may move information through expressions
• you may proceed without changing balance

Replacement is the engine.

Section 4 — You May Combine

Once symbols can be named, equated, and replaced, they may be combined.

Combination is permission to place expressions together according to agreed operations. In elementary algebra these operations are familiar. Their familiarity is not the point.

The permission is.

What Combining Allows

Combining allows multiple expressions to be treated as a single expression.

When you write:
$a + b$

you are not solving anything. You are forming a compound object that can be acted on later.

Combination creates structure without resolution.

Operations as Agreements

An operation is an agreement about how symbols may be joined.

The symbol + means combine under the rules of addition. Those rules are assumed to be stable and shared.

The same is true for multiplication, division, and exponentiation.

Combining Before Knowing

You are allowed to combine symbols even when their values are unknown.

For example:
$x + 3$

This allows work to proceed ahead of knowledge.

Order and Grouping

When combining, grouping matters.

Parentheses indicate how combination is structured.

For example:
$(a + b) + c$
$a + (b + c)$

Under addition these are equivalent. Under other operations they may not be.

Combination forms the object. Rearrangement comes later.

Common Misuse

A common error is combining without purpose.

Expressions grow longer without becoming clearer.

Another error is mixing operations without attending to their rules.

Algebra allows combination.
It does not excuse carelessness.

Minimal Example

Suppose:
$x = 4$

You may form:
$x + 3$

You may then combine further:
$(x + 3) \cdot 2$

No solution has occurred. Structure has been built.

Combination and Structure

As algebra advances, combinations become more abstract.

You combine:
- numbers
- symbols
- functions
- vectors
- entire expressions

The permission remains the same.

Summary of This Permission

You may combine.

This means:
- you may join expressions using agreed operations
- you may build compound structures
- you may proceed without resolving values

Combination builds form.

Section 5 — You May Rearrange (when allowed)

Rearrangement is conditional permission.

You may change the order or grouping of an expression only when the rules governing that expression allow it.

What Rearrangement Allows

When permitted, rearrangement allows you to change the appearance of an expression without changing its value.

For example, under addition:
$a + b = b + a$

Rearrangement improves access.

When Rearrangement Is Allowed

Rearrangement depends on the operation in use.

Some operations permit changing order or grouping. Others permit neither.

Addition and multiplication often allow both. Subtraction and division generally do not.

Rearrangement is never assumed.
It is verified.

Order and Grouping

There are two kinds of rearrangement:
• reordering terms
• regrouping terms

Reordering requires commutativity.
Regrouping requires associativity.

These permissions are local.

Why Rearrangement Matters

Rearrangement allows structure to reveal itself.

Terms can be brought together. Factors can be exposed.

Rearrangement prepares later work.

Common Misuse

A common error is rearranging because it looks right.

Another error is rearranging across operations that do not permit it.

These errors surface later.

Minimal Example

$a + b + c$

This may be rearranged freely.

$a - b - c$

Here, order and grouping matter.

Rearrangement as Preparation

You rearrange so that:
• like terms touch

• factors become visible
• an inverse can be applied cleanly

Nothing is solved yet.

Summary of This Permission

You may rearrange.

This means:
• you may change order or grouping when allowed
• you may reshape expressions without changing value
• you may prepare structure for later work

Undo comes next.

A Negative Example: An Unpermitted Rearrangement

Consider the expression:

$$a - b - c$$

A common impulse is to rearrange this expression so that similar symbols appear together:

$$a - c - b$$

This looks harmless. The symbols are the same. The operations appear unchanged.

This rearrangement is not permitted.

What Went Wrong

Subtraction does not permit free reordering.

The original expression is structured as:

$(a - b) - c$

The rearranged expression implies:

$(a - c) - b$

These are not equivalent.

No permission has been exercised that allows this change.
No equality was declared. No inverse was applied.
Rearrangement was assumed where it was not allowed.

The move felt intuitive.

It was illegal.

How the Error Reveals Itself

Often, errors of this kind are not noticed immediately.

They surface later, when:

undo operations fail to isolate a term

replacement produces inconsistent results

a final value contradicts earlier constraints

At that point, the mistake appears distant from its cause.

This is why algebra insists on permission at each step.

Correct Handling

The correct response is restraint.

If the structure does not permit rearrangement, do not rearrange.

Instead:

introduce parentheses deliberately

apply undo operations where inverses exist

replace only what has been equated

The expression will change when it is allowed to.

Instruction

If a rearrangement feels obvious, pause.

Ask:

Which permission allows this move?

Which operation governs this structure?

If no permission applies, do not proceed.

Algebra does not punish intuition.

It simply does not recognize it as authority.

Section 6— You May Parse

Parsing is permission to recognize structure.

Before an expression can be rearranged or undone, it must be read correctly. Algebraic expressions are not flat strings of symbols. They encode operations applied in a specific order and scope.

Parsing is the act of identifying that structure.

What Parsing Allows

Parsing allows you to determine:
• which operation governs an expression
• which parts of the expression fall within its scope
• which operation was applied most recently

Parsing does not change an expression.
It makes its structure explicit.

Structure and Scope

Every algebraic expression has an outermost operation.

For example:
$3(x + 2)$

The outermost operation is multiplication.
The addition occurs within its scope.

Parentheses indicate scope.
Absence of parentheses does not imply absence of structure.

The expression:
$a - b - c$

has structure:
$(a - b) - c$

Parsing identifies this grouping even when it is not written explicitly.

Parsing and Undoing

Undoing requires recognition of the operation to be reversed.

In the expression:
$3(x + 2) = 18$

Parsing reveals that multiplication by 3 governs the left-hand side. Only after this is recognized does division become a permitted undo operation.

Undoing without parsing is guesswork.

Parsing and Rearrangement

Rearrangement depends on which operation governs a structure.

To determine whether reordering or regrouping is allowed, you must know which operation applies and which subexpressions it binds together.

Parsing supplies this information.

What Parsing Does Not Allow

Parsing does not:
• suggest which move to make
• determine goals
• simplify expressions

It only identifies structure.

Common Misuse

A common error is treating adjacency as structure.

For example, assuming that:
$3x + 2$

means the same thing as:
$3(x + 2)$

These expressions have different structure.

Parsing prevents such errors by insisting on scope awareness.

Minimal Example

Consider:
$x + 3^2$

Parsing identifies exponentiation as governing 3, not the sum.

The structure is:
$x + (3^2)$

Undoing addition here is permitted.
Undoing exponentiation is not.

Summary of This Permission

You may parse.

This means:
• you may identify the governing operation of an expression

• you may determine scope and grouping
• you may recognize which undo operations are applicable

Parsing makes legality visible.

Section 7 — You May Undo (when an inverse exists)

Undoing is permission to reverse an operation.

This permission is conditional. You may only undo what has an inverse, and only within the rules of the system you are working in.

What Undoing Allows

When an operation has an inverse, you may apply it to remove the effect of the original operation.

If an expression has been altered by addition, you may undo that alteration by subtraction.
If it has been altered by multiplication, you may undo it by division.

Undoing restores balance without changing the underlying equality.

Inverses

An inverse is an operation that returns you to where you started.

Examples:
• adding 3 is undone by subtracting 3
• multiplying by 5 is undone by dividing by 5
• squaring may be undone by taking a square root, under certain conditions

The existence of an inverse is established, not assumed.

Undoing Acts on Both Sides

When undoing is applied within an equation, it must be applied symmetrically.

Undoing preserves equality.

Why Undoing Matters

Undoing allows expressions to be simplified and variables to be isolated.

Undoing clears structure so that replacement can proceed.

Conditions on Undoing

Undoing is allowed only when:
• the inverse exists
• the inverse is applied consistently
• no forbidden operation is introduced

Violating these conditions removes you from the system.

Common Misuse

A common error is undoing operations that were never applied.

Another error is undoing partially.

Undoing must correspond exactly to what was done.

Minimal Example

$x + 3 = 7$

Subtract 3 from both sides:

$x = 4$

One operation was reversed.

Undoing and Restraint

Undoing is powerful and therefore limited.

Proceed one operation at a time.

Summary of This Permission

You may undo.

This means:
• you may reverse an operation when an inverse exists
• you must apply the inverse consistently
• you may simplify structure without breaking equality

Undoing clears the path.

Section 8 — You May Repeat

Repetition is permission to apply the same lawful move again.

Algebra assumes this permission from the start. Without it, structure could not accumulate.

What Repetition Allows

When a move is allowed once, it may be allowed again.

If you may:
- replace equals with equals
- combine expressions
- rearrange under permitted operations
- undo using an inverse

then you may do so repeatedly, as long as each step remains valid.

Repetition Is Not Redundancy

Repeating a step is not a failure to progress.

Often, the same operation must be applied multiple times to different parts of an expression.

Progress in algebra is usually iterative.

Why Repetition Matters

Repetition allows small moves to accumulate into large changes.

This is how algebra scales without adding new rules.

Termination Is Optional

Algebra does not require you to stop.

You may repeat until:
• the expression reaches a desired form
• no further permitted moves apply
• you choose to stop

Completion is contextual.

Common Misuse

A common error is repeating without checking whether the move still applies.

Another is repeating after the structure has stabilized.

Minimal Example

$x + 3 + 2 = 9$

Subtract 2 from both sides:
$x + 3 = 7$

Subtract 3 from both sides:
$x = 4$

The same permission was used twice.

Repetition and Discipline

Repetition rewards patience.

The permission to repeat allows you to slow down without penalty.

Summary of This Permission

You may repeat.

This means:
• you may apply valid moves more than once
• you may proceed step by step
• you may let structure emerge gradually

Repetition turns permission into process.

What Remains When the Formulas Are Gone

When the formulas are set aside, algebra remains intact.

This is because formulas are not the system. They are temporary arrangements produced by the use of permissions. When the work ends, the permissions do not disappear.

This manual has described algebra as a sequence of allowed actions. Nothing more is required.

What You Have, Now

You have permission to:

- name
- equate
- parse
- replace
- combine
- rearrange (when allowed)
- undo (when an inverse exists)
- repeatWith these, any algebraic structure can be rebuilt.

You do not need to remember procedures.
You do not need to recognize familiar forms.
You do not need to move quickly.

You only need to know what is allowed next.

Why This Is Sufficient

Algebra does not depend on cleverness.
It depends on legality.

Every correct algebraic transformation is an application of one of the permissions listed here.

Mistakes are procedural, not personal.

On Solving

Solving is not a special operation.

It is the point at which no further undo, replacement, or rearrangement is required for the current purpose.

Algebra does not announce completion.
The user decides when to stop.

On Difficulty

If algebra feels difficult, it is usually because:
• too many permissions are being exercised at once
• an operation is being undone without an inverse
• rearrangement is being assumed where it is not allowed

The remedy is slowing down.

Return to the last permitted step.

On Use

Algebra travels well.

The same permissions apply whether the symbols stand for numbers, functions, vectors, matrices, or abstract elements.

The objects change.
The rules do not.

Final Note to the Reader

You are not required to like algebra.
You are not required to be fast.

You are required to remain within the permissions.

If you do, algebra will hold.
If you leave them, it will not.

That is the whole system.

The Permissions of Algebra

Permission	What It Allows
You May Name	Introduce a symbol to hold a place.
You May Equate	Declare two expressions the same.
You May Parse	Identify the governing operation, scope, and structure of an expression.
You May Replace	Substitute equals for equals within the same structure.
You May Combine	Join expressions using agreed operations.
You May Rearrange (when allowed)	Change order or grouping when the governing operation permits it.
You May Undo (when an inverse exists)	Reverse a parsed operation symmetrically.
You May Repeat	Apply any permitted move again.

A finite set of permissions generates an infinite space of valid expressions.

Read vertically. Proceed one permission at a time.
If no permission applies, stop.

On Strategy

This manual describes what moves are valid, not which move to make.

It is a grammar of manipulation, not a logic of problem-solving.

This is intentional.

Algebraic strategy—deciding which permission to apply at a given moment—depends on context, purpose, and desired form. Those factors lie outside the algebraic system itself. They belong to the user.

Algebra guarantees legality, not direction.

Why This Distinction Matters

Many failures in algebra arise from confusing these layers:

legality is mistaken for cleverness

strategy is mistaken for rule

intuition is mistaken for permission

This manual isolates the first layer so the others can be learned without confusion.

You cannot choose well among moves you do not yet recognize as lawful.

What This Manual Provides

This manual provides:

a complete account of permitted transformations

a way to audit each step for validity

a method for diagnosing error after the fact

It does not provide:

heuristics

tactics

shortcuts

problem templates

Those belong to instruction, practice, and experience.

Appendix A: Annotated Examples

This appendix provides fully annotated walkthroughs of common algebraic tasks using only the permissions from the manual. Each step explicitly names the permission(s) applied, with rationale. Annotations appear in bold italics for clarity. These examples demonstrate how the finite permissions generate standard procedures without relying on memorized formulas.

Example A1: Solving a Linear Equation ($x + 5 = 12$)

Start: $x + 5 = 12$

1. Parse the left side: outermost operation is addition (5 added to x).
Parsing identifies the governing operation for future undoing.

2. Undo addition of 5 by subtracting 5 from both sides (inverse exists for addition).
$x + 5 - 5 = 12 - 5$
Undoing preserves equality and is applied symmetrically.

3. Combine on left ($x + 0$) and on right (constants).
$x = 7$
Combination forms simpler compound expressions; here it reveals identity elements implicitly.

No further permitted moves toward isolation apply. Solved.

Example A2: Solving with Multiple Steps ($3x - 4 = 11$)

Start: $3x - 4 = 11$

1. Parse: left side has subtraction as outermost on the compound $(3x - 4)$; within, multiplication governs x.

2. Undo subtraction of 4 by adding 4 to both sides.
$3x - 4 + 4 = 11 + 4$
$3x = 15$
Undoing clears the outer operation first.

3. Parse again: now multiplication by 3 governs x.

4. Undo multiplication by 3 by dividing both sides by 3.
$3x / 3 = 15 / 3$
$x = 5$
Inverse of multiplication is division (non-zero divisor).

5. Combine simplifies both sides.

Solved.

Example A3: Combining Like Terms and Solving $(2x + 3x - 7 = 13)$

Start: $2x + 3x - 7 = 13$

1. Parse: addition and subtraction chain; structure is $((2x + 3x) - 7)$.

2. Rearrange (addition is commutative and associative): regroup like terms.
$(2x + 3x) - 7 = 13$
Rearrangement allowed because addition permits reordering and regrouping.

3. Combine coefficients of like terms (agreement on multiplication/distribution implicit).
$5x - 7 = 13$

4. Undo subtraction of 7 by adding 7 to both sides.
$5x = 20$

5. Undo multiplication by 5 by dividing both sides by 5.
$x = 4$

Solved.

Example A4: Distributive Property Emergent ($2(x + 3) = 10$)

Start: $2(x + 3) = 10$

1. Parse: outermost is multiplication by 2; within parentheses, addition.

2. Option A – Undo multiplication first:
Undo multiplication by 2 (divide both sides by 2).
$x + 3 = 5$
Then undo addition: $x = 2$
Option B – Expand first (alternative path):
Combine via distribution (multiplication over addition is agreed operation).
$2x + 6 = 10$
Then proceed as in previous examples: subtract 6, divide by 2 → $x = 2$

Both paths valid; permissions allow choice of order when multiple moves possible.

Example A5: Quadratic (Illustrating Limits) ($x^2 = 9$)

Start: $x^2 = 9$

1. Parse: exponentiation (squaring).

2. Undo squaring by square root (inverse exists, but conditional on domain).

$x = \sqrt{\pm}9$

$x = \pm 3$

Note: inverse is not single-valued over reals; permission requires acknowledging both branches or context restriction.

Appendix B: Flowcharts Mapping Permissions to Common Problems

Below are textual representations of flowcharts (decision trees) that map the eight permissions to typical algebraic tasks. These can be used as diagnostic or planning tools: start from an expression and follow branches based on "What is allowed next?"

Flowchart B1: General Strategy for Isolating a VariableStart: Equation with target variable

├── Parse: Identify outermost operation governing target

├── Addition/Subtraction? → Undo by opposite (add/subtract same to both sides) → Repeat/Parse again

├── Multiplication/Division? → Undo by inverse (multiply/divide both sides) → Repeat/Parse again

├── Exponentiation? → Undo by root (conditional) or log → Specify domain

│ └── Other (e.g., function)? → Apply agreed inverse if exists

│

├── Like terms present? → Rearrange (if commutative/associative) → Combine coefficients

│

├── Parentheses with distributable factor? → Combine via distribution

│

└── Target isolated? → Yes: Stop | No: Return to Parse

Flowchart B2: Simplifying Expressions (No Equation)Start: Expression to simplify

│

├── Parse: Identify structure/scope

│

├── Like terms? → Rearrange (if allowed) → Combine

│

├── Distributable factors? → Combine over parentheses

│

├── Redundant operations (e.g., +0, ×1)? → Undo/Replace with identity

│

└── No further moves? → Stop (form is canonical for current permissions)

Start: Expression to simplify

│

├── Parse: Identify structure/scope

│

├── Like terms? → Rearrange (if allowed) → Combine

│

├── Distributable factors? → Combine over parentheses

│

├── Redundant operations (e.g., +0, ×1)? → Undo/Replace with identity

│

└── No further moves? → Stop (form is canonical for current permissions)

Flowchart B3: Verifying or Debugging a Step

Suspected error in work

│

├── Check last move against permissions:

    ├── Was it Naming? (only introduces symbol) → Valid if no premature value

    ├── Equating? → Both sides same system? → Yes/No

    ├── Replacement? → Only equals for equals? → Yes/No

    ├── Combination? → Agreed operation? → Yes/No

    ├── Rearrangement? → Commutative/associative allowed? → Yes/No

    ├── Parsing? → Correct scope identified? → Yes/No

    ├── Undoing? → Inverse exists and applied symmetrically? → Yes/No

    └── Repeating? → Previous move still valid? → Yes/No

│

└── Invalid? → Revert to last legal step → Restart from there