Prompt Engineering / Reasoning Systems

Convert natural language problems into symbolic form (logic, algebra, constraints) then solve systematically.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Logic Puzzles, Math Word Problems, Constraint Satisfaction
Updated: May 2026
Why This Prompt Exists
LLMs are bad at systematic reasoning. They skip steps, make intuitive leaps, and forget constraints. Symbolic representation forces precision.

You get:

- math word problems solved incorrectly (wrong order of operations)
- logic puzzles where constraints are forgotten mid-reasoning
- planning problems where the model violates its own plan
- no way to verify reasoning systematically
- errors that would be obvious in symbolic form

But symbolic reasoning works:

- translation: natural language → formal representation (variables, equations, logical statements)
- systematic solving: apply rules, not intuition
- verification: check each step against constraints
- back-translation: symbolic answer → natural language

Without symbolic representation, LLMs guess.

This prompt converts natural language problems to symbolic form and solves systematically.

The Prompt

```
Assume the role of a symbolic reasoning engine that formalizes natural language problems.

Your task is to convert a problem to symbolic form and solve it systematically.

Generate:

1. NATURAL LANGUAGE PROBLEM
   - Original problem statement

2. SYMBOLIC TRANSLATION
   - Variables: [define all variables with types]
   - Constraints: [list all constraints in symbolic form]
   - Goal: [what we need to find, symbolically]

3. CONSTRAINT CHECK
   - Are constraints consistent? (No contradictions)
   - Are there enough constraints to determine a unique solution?

4. SYSTEMATIC SOLUTION
   - Step 1: [apply rule/operation to constraints]
   - Step 2: [derive new constraints]
   - Step 3: [continue until goal is reached]
   - Show all intermediate derivations
```

5. SYMBOLIC ANSWER
   - Solution in symbolic form (variable assignments)

6. NATURAL LANGUAGE ANSWER
   - Translation back to natural language

7. VERIFICATION
   - Plug answer back into original constraints
   - Does it satisfy all constraints? (Yes/No)

INPUTS:

Natural language problem:
[PASTE THE PROBLEM]

Problem type:
[MATH WORD PROBLEM / LOGIC PUZZLE / CONSTRAINT SATISFACTION / PLANNING]

Symbolic formalism to use:
[FIRST-ORDER LOGIC / ALGEBRAIC EQUATIONS / PROPOSITIONAL LOGIC / OTHER]

Model:
[GPT-4 / CLAUDE / GEMINI]

RULES:
- Define all variables explicitly before using them
- Write constraints as equations or logical statements (not prose)

```
- Show every derivation step (no skipping)
- Check for consistency before solving (contradictory constraints = no solution)
- Verify solution by substituting back into constraints
- If problem can't be represented symbolically, flag as "not suitable for symbolic reasoning"
```

How To Use It

- Use for math word problems, logic puzzles, and constraint satisfaction problems.
- Define variables clearly — ambiguous variables lead to wrong solutions.
- Check for constraint consistency before solving (contradictions mean no solution).
- Verify the solution by plugging it back into the original constraints.
- If the model struggles with symbolic representation, break the problem into smaller pieces.

Example Input

Natural language problem:
“A bat and a ball cost $1.10 in total. The bat costs $1.00 more than the ball. How much does the ball cost?”

Problem type:
“MATH WORD PROBLEM”

Symbolic formalism:
“ALGEBRAIC EQUATIONS”

Why It Works
LLMs answer this classic puzzle wrong ~50% of the time when answering directly. But with symbolic translation, they get it right.

This framework improves outcomes by forcing:

- variable definition (explicit, no ambiguity)
- constraint formalization (equations, not prose)
- systematic solving (step-by-step algebra)
- consistency checking (detect contradictions)
- verification (plug answer back in)

Great symbolic reasoning doesn't replace the LLM — it structures the problem so the LLM can solve it correctly.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also Backward Reasoning Engine