Constraint Folding

AI discovers the law from observation, and that law itself becomes the running system — the deepest mechanism of ai-desk

ai-desk methodology — Hiroyuki OKINOI · 2026-05

1 The surface face — "put branches in data"

At first glance, Constraint Folding looks like just a technique for replacing if/else with a table.

```
// the typical "branches → data" presentation
const TRANSITIONS = [
  { from: 'title',    input: 'start', to: 'playing'  },
  { from: 'playing',  input: 'pause', to: 'paused'   },
  // ...
];
function reduce(state, input) {
  return TRANSITIONS.find(t => t.from === state && t.input === input)?.to ?? state;
}
```

This is only the **entrance**. The real power lies beyond.

2 The real face — "a substrate for discovering laws from observation"

The core of Constraint Folding is that **AI infers the world (the table) from observed data, and that table runs as-is**.

> Humans hand over **intent + observation samples**
> → AI **extracts the law and generates the table**
> → that table **is itself the running state machine**

The phase called "writing code" disappears. In its place comes a new phase: "**extracting the law from observation**."

This behavior emerges when Bible §6 "Mine and Verify" combines with Constraint Folding.

3 Four phases

1
Observe
Collect (input, output) samples from existing systems or business rules

2
Generate
LLM + Monte Carlo enumerate many candidate laws

3
Verify
Run each candidate against every observation, discard whatever doesn't fit

4
Build
The surviving laws = the constraint table = a state machine that already runs

Humans only touch **1 (provide observations) and 4 (confirm behavior). 2 and 3 are entirely AI.**

4 Proof — shipping cost: 50 samples → 100% reconstruction

From a state where the source of a legacy shipping rule was lost, full reconstruction from 50 observations.

50	100%	0
observation samples (input → output pairs)	reconstruction accuracy (consistent on all 50)	if statements written by humans (implementation is just the table)

Concretely: PoC code lives in `examples/`.
From 50 (weight, distance, zone, fee) pairs, AI infers a table of "thresholds, base values, additive rules."
Feeding the reconstructed table into the reducer reproduces every observation. **The system was rebuilt without ever reading the source code**.

5 Why "folding" makes inference possible

Because it's a pure reducer, AI can **mechanically try and prune candidates**.

- **Same input → same output** (purity) → observations are reproducible, verification is reproducible
- **State and input are data** → all combinations are enumerable → candidate generation can cover the entire combinatorial space
- **Constraints are filter predicates** → "does this candidate pass the samples" is a simple filter
- **It can be run in reverse** → "what inputs would produce this output" can be enumerated → this widens the candidate space for inference

A branch tree makes all of these structurally impossible (control flow can't run backward, combinations explode, verification has side effects).
Folding is what lets AI run inference.

6 The inferred table = a running state machine

The inference result is directly executable. No "code generation" phase needed.

```
// table generated by AI inference (no human wrote this)
const SHIPPING_RULES = [
  { weight_max: 1,  zone: 'kanto',    base: 300, per_kg: 0   },
  { weight_max: 5,  zone: 'kanto',    base: 500, per_kg: 100 },
  { weight_max: 1,  zone: 'kansai',   base: 350, per_kg: 0   },
  // ...
];

// the reducer is fixed (generic). swap the table to swap the behavior
function resolve(weight, zone) {
  const rule = SHIPPING_RULES.find(r => r.zone === zone && weight <= r.weight_max);
  return rule ? rule.base + (weight * rule.per_kg) : null;
}
```

The moment AI infers it, the system already runs. **"Human transcribes inference into code" disappears**.

7 Two tiers — enumeration (basic) + inference (advanced)

BASIC: ENUMERATION

Humans declare the axes → cartesian product expands the table

```
HANDS.flatMap(a =>
  HANDS.flatMap(b =>
    HANDS.map(c => ({a,b,c}))
  )
);  // 27 worlds
```

Total coverage of a known domain

ADVANCED: INFERENCE

Observations → AI back-infers constraints → table generated automatically

```
// from 50 samples
const rules = infer(observations);
// AI generates SHIPPING_RULES
// → can be passed straight to the reducer
```

Law discovery in unknown domains

Both produce the same output: "table + reducer". **The same execution substrate is usable through two paths — humans writing it, or AI inferring it**.

8 What changes

"Writing code" is replaced by "extracting the law from observation."

TRADITIONAL DEVELOPMENT

- Read / write the spec
- Translate the spec into if/else
- Test every branch combinatorially
- Spec change → revisit every branch
- Run, find bugs, fix

CONSTRAINT FOLDING + INFERENCE

- Prepare working samples / observations
- AI infers laws → generates the table
- The table runs in the reducer as-is
- Spec change = add samples → re-infer
- Bugs = mismatch with observations, detected immediately

Humans stop doing translation work. They keep observation and confirmation.

9 Where it applies

Legacy reconstruction

Rebuild business logic with lost source from observation. Shipping fees, tax computation, discount logic.

New-domain discovery

From "I want behavior like this" samples, AI infers the rule → an immediately running implementation.

Auditing live systems

From observations of a running system, reverse-verify "is the implementation what was intended?" Detect spec/impl drift.

Automating business rules

Operator decision history → rule inference → automation. Turn "tacit know-how" into data.

Game balance tuning

From play logs, extract "fun state combinations" → adjust the rules.

General constraint solving

Any finite discrete domain. Scheduling, placement, combinatorial optimization.

10 The endpoint of ai-desk — dissolving the concept of "code"

Ultimately, the step called "writing code" disappears.

> What remains is **three things**:
> · human **intent + observation samples**
> · AI's **law inference**
> · a fixed **reducer execution substrate**

"Codebase" becomes "**a set of constraint tables**".
"Refactoring" becomes "**add samples and re-infer**".
"Bug fix" becomes "**add the failing observation and reconstruct**".

This is the final resolution of Bible §0.0 "complexity is the human's problem; concealment is the AI's problem." Humans hold **intent and observation**, AI holds **inference and construction**, the machine holds **pure execution**. The roles separate completely.

Observation → Law → Execution

Constraint Folding is not a "data-ification trick" —
it is the substrate where AI discovers the law and the law itself runs.

Enumeration is the entrance, **inference is the heart**.
On this substrate, development is no longer "writing" but "**extracting**".

github.com/AoyamaRito/ai-desk
Hiroyuki OKINOI · Aoyama Rito