型付き協調 AI 環境における問題解決戦略

Actor、Protocol、Memory、Review による安全な AI Orchestration

Yasushi Kodama Repository Report

2026 年 5 月 4 日

概要

本稿は、型付き協調 AI 環境における問題解決戦略を論じる。大規模言語モデルを用いた AI system は、単一の推論器としてではなく、planner、solver、reviewer、memory、tool、model router など複数の役割を持つ component の協調として設計されることが多い。しかし、協調 AI は非同期性、役割間依存、外部 tool、記憶、失敗、曖昧な自然言語出力を含むため、単純な prompt chaining では信頼性を確保しにくい。

本稿では、型付き actor 言語 AIPL と actor-based intelligent computing environment である AICE を背景に、問題解決を「型付き message を交換する actor 群による protocol-guided search」として捉える。型は、値の形を検査するだけでなく、役割間 interface、capability 境界、future の結果、review feedback、session protocol を明示するために用いられる。これにより、AI の非決定性そのものを消すのではなく、非決定性の周囲に検査可能な構造を与える。

本稿の中心的主張は、型付き協調 AI 環境における有効な問題解決戦略は、分解、並列探索、検証、記憶、反復改善、protocol 管理の六つを統合する必要がある、という点である。

目次

1 はじめに

AI による問題解決は、単一の model call だけでは十分でない。現実の問題は曖昧であり、必要な情報が不足し、外部 tool の利用を要求し、途中結果の検証を必要とする。たとえば、プログラム修正、研究調査、設計案作成、数理的証明、業務計画、データ分析では、問題を分解し、情報を集め、複数案を比較し、結果を検査し、必要に応じてやり直す過程が不可欠である。

このような処理を単一の巨大 prompt に押し込むと、system は脆くなる。どの部分が計画で、どの部分が解答で、どの部分が検証で、どの外部 tool を使ったのかが曖昧になるためである。さらに、非同期に実行可能な subtask も逐次化され、失敗時の回復点も不明瞭になる。

型付き協調 AI 環境は、この問題に対し、問題解決を複数 actor の協調として構成する。planner は問題を分解し、solver は subproblem を解き、reviewer は結果を検査し、memory は文脈を保持し、tool actor は外部操作を実行し、model actor は LLM を呼ぶ。各 actor は型付き message を通じて接続される。

本稿では、この環境においてどのような問題解決戦略が有効かを整理する。

2 型付き協調 AI 環境

定義 1 (型付き協調 AI 環境)**.** 型付き協調 *AI* 環境とは、複数の *AI actor* が型付き *message* を交換しながら問題を解決する実行環境である。各 *actor* は内部状態と *method interface* を持ち、外部とは *message passing* によってのみ相互作用する。

この環境は次の要素からなる。

表 1: 型付き協調 AI 環境の構成要素

構成要素	役割
Actor	独立した実行単位。method と mailbox を持つ
Typed message	method 名、payload 型、必要なら session id を持つ通信単位
Future	非同期 subtask の結果 handle
Memory	問題、計画、途中結果、根拠、失敗履歴を保持する
Tool	外部 API、file、shell、database、search などの実行境界
Protocol	actor 間の呼び出し順序と役割を定める
Reviewer	出力の妥当性、根拠、形式、制約充足を検査する

AIPL/AICE の観点では、これらはすべて actor として表現できる。重要なのは、AI component を特別扱いしないことである。model も memory も tool も actor であり、actor interface と型によって接続される。

3　問題解決を protocol-guided search として捉える

協調 AI における問題解決は、単なる関数適用ではなく探索である。問題に対して、複数の plan、複数の tool 呼び出し、複数の解候補、複数の review 結果があり得る。そのため、問題解決は次のような protocol-guided search として捉えられる。

$$Problem \to Plan \to Subproblem^* \to Candidate^* \to Review^* \to Answer$$

ここで protocol は探索の自由度を完全には消さない。むしろ、探索を安全な範囲に閉じ込める。たとえば、solver が解候補を出す前に reviewer を呼んではならない。tool result は validator を通してから solver に渡すべきである。memory は根拠の保存と検索に使うが、古い情報をそのまま最終答案にしてはならない。

このような制約は、自然言語の指示だけでは弱い。型と protocol によって actor 間の接続を制約することで、探索過程を制御できる。

4　六つの基本戦略

型付き協調 AI 環境における問題解決戦略は、次の六つに整理できる。

1. 分解戦略
2. 並列探索戦略
3. 検証戦略
4. 記憶戦略
5. 反復改善戦略
6. protocol 管理戦略

これらは独立ではない。実際の問題解決では、planner が分解し、solver 群が並列探索し、reviewer が検証し、memory が履歴を保持し、coordinator が feedback に応じて再計画し、protocol checker が順序を管理する。

5　分解戦略

分解戦略は、入力問題をより小さい subproblem に変換する戦略である。

$$Planner : Problem \to Plan$$

Plan は単なる文字列ではなく、subtask の集合、依存関係、必要 capability、期待される出力型を含む構造であるべきである。

$$Plan = \{Task_i, Dependency_{ij}, Capability_i, OutputType_i\}$$

型付き環境では、planner の出力型を明示できる。

$$\texttt{planner.plan} : \mathit{Problem} \to \mathit{Plan}$$

分解戦略の利点は三つある。

- 大きな問題を扱いやすい単位へ分割できる。
- 並列実行可能な subtask を見つけられる。
- reviewer が検査すべき単位を明確にできる。

一方、過剰分解は overhead を増やす。したがって planner は、問題の複雑さ、tool cost、model latency、検証必要性を考慮して粒度を選ぶ必要がある。

6 並列探索戦略

AI の問題解決では、単一の解候補に早く収束することが誤りを生みやすい。型付き協調 AI 環境では、複数 solver actor を future として並列に走らせることができる。

```
var f1 = future solver1.solve(task);
var f2 = future solver2.solve(task);
var f3 = future solver3.solve(task);

var a1 = await f1;
var a2 = await f2;
var a3 = await f3;
```

型付き future は、非同期探索の結果型を失わない。

$$\frac{\Gamma \vdash e : \mathit{future}\ \tau}{\Gamma \vdash \mathit{await}\ e : \tau}$$

並列探索にはいくつかの型がある。

表 2: 並列探索の型

方式	説明
同質並列	同じ solver を複数回走らせ、多様な候補を得る
異質並列	異なる model、prompt、tool strategy を使う solver を走らせる
分割並列	subproblem ごとに solver を割り当てる
検証並列	複数 reviewer に同じ候補を検査させる
探索木並列	plan の分岐を複数 actor で探索する

並列探索では、結果の統合 actor が必要になる。

$$\mathit{Aggregator} : \mathit{Candidate}^* \to \mathit{Candidate}$$

Aggregator は単に多数決をするだけでは不十分である。根拠、制約充足、tool result、reviewer score、既存 memory との整合性を考慮する必要がある。

7 検証戦略

検証戦略は、solver の出力をそのまま採用せず、reviewer actor によって検査する戦略である。

$$Reviewer : Candidate \to Review$$

Review は、単なる OK/NG ではなく、少なくとも次の情報を持つべきである。

$$Review = \{status, issues, evidence, required\_fixes, confidence\}$$

型付き環境では、reviewer の interface を次のように定める。

$$\texttt{reviewer.review} : Candidate \to Review$$

検証戦略には次の種類がある。

- 型検証: 出力が期待型を満たすか。
- 形式検証: 指定された format、schema、LaTeX、JSON、code style を満たすか。
- 根拠検証: 主張に根拠があるか。
- 実行検証: code、test、proof、command が実際に通るか。
- 反例探索: 出力を壊す input や case がないか。

重要なのは、reviewer を solver と同じ actor にしないことである。同じ actor が自分の出力を検証すると、誤った前提を共有しやすい。AICE では solver と reviewer を別 actor として分離することで、役割の独立性を高める。

8 記憶戦略

記憶戦略は、問題解決の状態を memory actor に保存し、次の推論で利用する戦略である。

$$Memory.put : Key * Value \to Unit$$

$$Memory.get : Key \to Value$$

AI problem solving では、記憶すべきものが複数ある。

- user goal
- constraints
- plan
- subtask result
- tool result
- review issue

- accepted decision
- rejected hypothesis
- final answer

型付き memory の利点は、記憶された値の意味を区別できる点である。`Plan`、`Candidate`、`Review`、`Evidence` をすべて文字列として扱うと、古い candidate を evidence と誤用する危険がある。

記憶戦略には、短期記憶と長期記憶の区別も必要である。

表 3: 記憶の分類

種類	内容
短期記憶	現在の task 内の plan、途中結果、review issue
長期記憶	再利用可能な知識、ユーザ preference、過去の成功 strategy
作業記憶	actor 間で共有する一時的 state
監査記憶	tool 実行、model call、protocol event の log

監査記憶は特に重要である。AI system がなぜその答えに至ったかを後から検査できるようにするためである。

9 反復改善戦略

反復改善戦略は、review 結果に基づいて solver または planner に戻る戦略である。

$$Candidate \to Review \to RevisionTask \to Candidate'$$

単純な retry は同じ誤りを繰り返す可能性がある。型付き協調 AI 環境では、reviewer が返す `Review` を構造化し、修正 task に変換する。

$$ReviewIssue \to FixInstruction$$

反復改善の protocol は次のように書ける。

$$planner.plan \to solver.solve \to reviewer.review \to \begin{cases} finalize & \text{if accepted} \\ solver.revise & \text{if fixable} \\ planner.replan & \text{if plan invalid} \end{cases}$$

ここで重要なのは、どの段階に戻るかを review の型で決めることである。

$$ReviewStatus ::= Accepted \mid NeedsRevision \mid NeedsReplan \mid Rejected$$

`NeedsRevision` なら solver に戻り、`NeedsReplan` なら planner に戻る。これにより、反復改善は ad hoc な retry ではなく、型付き状態遷移になる。

10 protocol 管理戦略

protocol 管理戦略は、actor 間の順序を制御する。型付き message が正しくても、順序が誤れば問題解決は失敗する。

たとえば、次の protocol を考える。

$$planner.plan \rightarrow solver.solve \rightarrow reviewer.review \rightarrow finalizer.finish$$

この protocol では、reviewer は solver の後でなければならない。solver が複数並列に走る場合は、aggregator が候補をまとめてから reviewer に渡す必要がある。

runtime protocol checker は、現在位置の expected action と実際の action を照合する。

$$expected(pos) = actor.method$$

一致すれば position を進め、一致しなければ message を mailbox に入れる前に拒否する。

静的 session type を導入すれば、この検査を実行前に行える。

$$\Gamma; \Delta(s) = a.m; P \vdash send\ a.m(args) \triangleright \Delta[s := P]$$

protocol 管理は、AI workflow の安全性に直結する。特に、tool call、memory write、final answer のような副作用を持つ action は、protocol によって順序を固定する必要がある。

11 戦略選択の型

協調 AI における戦略そのものも型を持つと考えられる。問題の性質に応じて、どの strategy を選ぶべきかは異なる。

$$Strategy : ProblemProfile \rightarrow Plan$$

ProblemProfile は次の情報を含む。

- 問題の曖昧性
- 必要な外部情報
- 検証可能性
- 失敗時の cost
- 並列化可能性
- 応答時間制約

たとえば、検証可能性が高い code task では、tool 実行と test を含む戦略を選ぶ。調査 task では、source gathering と citation review を含む戦略を選ぶ。創造的 task では、複数案生成と preference-based review を含む戦略を選ぶ。

表 4: 問題 profile と推奨戦略

問題 profile	推奨戦略
形式検証可能	solver の後に test/proof actor を必ず置く
情報不足	research actor と memory actor を先に使う
高リスク	複数 reviewer と audit log を使う
低遅延要求	分解を浅くし、future 並列を制限する
探索空間大	planner による分岐生成と pruning を使う
曖昧な要求	clarifier actor または assumption reviewer を使う

12 安全性と有効性

型付き協調 AI 環境における安全性は、次の三層に分けられる。

1. 型安全性: well-typed message は payload shape mismatch を起こさない。
2. protocol 安全性: workflow は定義された順序を破らない。
3. 検証安全性: final answer は reviewer または tool による検証を通過している。

有効性は、単に安全であるだけでは不十分である。問題解決 system は、適切な探索を行い、必要な情報を取得し、誤りを修正し、最終的に user goal を満たす必要がある。

したがって、有効な協調 AI strategy は次の条件を満たす。

- 問題を適切な粒度に分解する。
- 並列化できるところは future で並列化する。
- 外部 tool を capability として明示的に使う。
- 各候補を reviewer または executable check に通す。
- 失敗を memory に残し、同じ失敗を繰り返さない。
- final answer に至る protocol trace を監査可能にする。

13 限界

型付き協調 AI 環境にも限界がある。

第一に、型は意味的正しさを完全には保証しない。文字列型の answer が正しいとは限らない。そのため reviewer、tool execution、proof checker、test runner が必要である。

第二に、LLM の自然言語出力は構造化されていない。型付き環境では、自然言語出力を parser/validator actor に渡し、構造化型へ変換する必要がある。

第三に、protocol は柔軟性を制限する。探索的問題解決では、予期しない有効な経路が必要になることがある。そのため protocol は固定列だけでなく、分岐、反復、例外処理を表現できる必要がある。

第四に、過度な actor 分割は overhead を増やす。型付き協調 AI の設計では、安全性と実行効率の trade-off を考慮しなければならない。

14 結論

型付き協調 AI 環境における問題解決は、単一の model call ではなく、型付き actor 群による protocol-guided search として設計すべきである。本稿では、そのための基本戦略を分解、並列探索、検証、記憶、反復改善、protocol 管理の六つに整理した。

型は、値の誤用を防ぐだけでなく、actor interface、future result、capability boundary、review feedback、session protocol を構造化する。これにより、AI の非決定性を完全に消すのではなく、非決定性の周囲に検査可能な実行構造を与えることができる。

今後の課題は、strategy selection 自体を型付きにし、problem profile に基づいて協調 protocol を自動生成・検証することである。これが実現すれば、AICE/AIPL のような型付き actor AIOS は、複雑な問題解決を安全かつ監査可能に実行する基盤となる。