小林研究の成果を AIPL に取り入れる場合の言語仕様

— 可能性、メリット、デメリット —

児玉工房 / AICE · AIPL プロジェクト

2026 年 07 月 30 日 13:00 JST 版

概要

本稿は二つの入力を突き合わせる。一つは小林直樹の三十余年の研究 —- 線形型、デッドロック自由な型システム、π 計算の汎用型システム、資源使用解析、高階モデル検査と交差型の等価性、述語抽象と CEGAR、高階不動点論理 HFL、制約付きホーン節への還元、サイズ型による並列計算量 —- であり、もう一つは先のレポート「AIPL の言語仕様の検討」で得た結論、すなわち**一つの中核と二つのプロファイル**（分散 AI エージェントと実時間 OS）である。

結論を先に述べる。**「全て取り入れる」ことは可能でもなく望ましくもない。**しかし小林の道具箱は、先のレポートが**未解決として残した三つの穴**にそのまま噛み合う。返信先を一級にすると「ちょうど一度」が壊れる問題は線形型が解き、`where` ガードの検証に依存型が要るという問題は述語抽象と CEGAR が解き、実時間プロファイルの応答時間上界はサイズ型と使用型が扱える。さらに π 計算の汎用型システムは、二つのプロファイルを**別々に作り込む**のではなく**一つの枠組みの二つの実例**として与える道を示す。

一方で代償は明確である。検証は還元に依存するため反例が AIPL の言葉で出なくなり、型システムを重ねるほど健全性の機械検証が現実的でなくなり、教育用言語としての可読性が落ちる。そして最も重要な非対称性として、**これらの道具はほぼすべて実時間プロファイル側に効き、AI エージェント側には効きが薄い**。非決定的なモデル出力を相手にする側では、静的に言えることが原理的に限られる。

本稿は道具ごとに対応づけを示し、採る／条件付きで採る／見送る の三分類を与える。

目次

第 I 部
何を取り入れるのか

1　はじめに

1.1　二つの入力

先のレポート「AIPL の言語仕様の検討」（29 ページ）は次の結論に達した。

- 分散 AI エージェント言語と実時間 OS 記述言語の要求は多くの点で正反対であり、一つの言語で無差別に許すとどちらの保証も得られない。
- したがって**一つの中核と二つのプロファイル**（`agent` / `realtime`）に分け、両者を分ける機構は effect/capability 系とする。
- 理想仕様として、`any` を消し、無期限に待つ構文を文法から消し、受信経路を一本化し、効果を静的に検査することを掲げた。

同時に、いくつかを**未解決として残した**。本稿の主題はここである。

残した穴	先のレポートでの扱い
返信先の一級化	`Reply`$\langle\tau\rangle$ にすれば型は付くが、「ちょうど一度返す」を保つには線形型が要る —- と述べて止まった
`where` ガードの検証	状態依存の受理条件を**証明**するには依存型・篩型が要る。現実的には動的検査と割り切る —- と述べて止まった
応答時間の上界	プロファイル R では「解析できる」と書いたが、何で解析するかは示していない
多相メソッド	ABCL/f が掲げた方向だが、ρ の量化問題で行き詰まる
セッション型	長期目標として置いただけ

1.2　小林の道具箱

もう一つの入力は、別稿「小林直樹先生の研究の流れ」に整理した五期の成果である。本稿で使う道具を列挙する。

道具	主な出典	内容
線形型	POPL 1996 (Pierce, Turner と)	チャネルをちょうど一度使う型
デッドロック自由な型	LICS 1997 / CONCUR 2006	型が付けばデッドロックしないことを保証
汎用型システム	POPL 2001 / TCS 2004	性質ごとに型システムを作るのをやめ、生成する枠組みを与える
資源使用解析	POPL 2002 / TOPLAS 2005	資源の**使用順序**を型で追う
高階モデル検査 = 交差型	POPL 2009 / LICS 2009 (Ong と)	モデル検査と型検査が同じ問題
実用アルゴリズム	FoSSaCS 2011	最悪計算量は絶望的でも実入力では線形に近い
述語抽象 + CEGAR	PLDI 2011	無限データ領域を抽象化し、反例で洗練する
HFL モデル検査	ESOP 2018 / PACMPL 2023	プログラム検証を高階不動点論理の妥当性検査へ還元
精製型 + モデル検査	PEPM 2019	精製型推論とモデル検査を組み合わせる
時相論理	POPL 2016 / SAS 2019	高階関数プログラムの時相性質を書き、検証する
CHC への還元	ESOP 2020 (RustHorn)	所有権を使ってポインタ付きプログラムをホーン節へ
所有権精製型	ESOP 2020 (ConSORT)	命令型プログラムの文脈・フロー依存な所有権型
サイズ型 + 使用型	CONCUR 2021	π 計算プロセスの**並列計算量**
停止性の還元	APLAS 2021	π 計算の停止性を逐次プログラムの停止性へ
gradual 化	ESOP 2023	テンソル形状検査を段階的型付けで

第 II 部
対応づけ —- 道具ごとに

2　線形型 —- 返信先の一級化と「ちょうど一度」

2.1　現状の何が足りないか

AIPL では `reply` が高々一度であることを**構文的に**検査している。本体を走査して回数の上界を数え、2 以上なら拒否する（ループ本体の `reply` は 2 回以上とみなす）。注釈があれば下界も課すので「ちょうど一度」になる。

これは返信先が**暗黙**である限り成立する。しかし ABCL/1 のように返信先を値として渡せるようにすると、構文的な走査は無力になる。渡された先で何回返されるかは、走査では分からない。

2.2　線形型を返信先に与える

小林らが π 計算に与えた線形型がそのまま使える。返信先を $\mathtt{Reply}\langle\tau\rangle$ 型の**線形な**値とする。

```
// r は線形。ちょうど一度使わなければならない
method fetch(k) : int {
  var r : Reply<int> = replyto;      // 返信先を値として取り出す
  send worker.compute(k, r);         // 委譲する（r を消費）
  // ここで reply(...) は書けない ---- r をすでに使った
}
```

```
class Worker {
  method compute(k, r : Reply<int>) : unit {
    reply_to(r, k * 2);              // r を消費して返す
  }
}
```

型規則は線形性の標準的な形になる。

$$\frac{\Gamma_1 \vdash r : \texttt{Reply}\langle\tau\rangle \qquad \Gamma_2 \vdash e : \tau}{\Gamma_1 \uplus \Gamma_2 \vdash \texttt{reply\_to}(r, e) : \texttt{unit}}\ \textsc{Reply-Use}$$

$$\frac{\Gamma, \mathit{replyto}{:}\texttt{Reply}\langle\tau_r\rangle \vdash \mathit{body} : \texttt{unit} \qquad \text{線形環境が本体で使い切られる}}{\Gamma \vdash \texttt{method}\ m(\overline{x}) : \tau_r\ \{\mathit{body}\}}\ \textsc{Method-Linear}$$

$\uplus$ は線形環境の分割である。これで**委譲を許しても「ちょうど一度」が保たれる**。先のレポートが「線形型が要る」と述べて止まった箇所が、これで閉じる。

所見 1. 副産物として、**現在の構文的な回数検査を捨てられる**。線形性は型で保証されるので、`max_replies` のような走査は不要になる。ループ本体の `reply` を「2 回以上」と近似する保守性も消える—- ループ内で線形な値を消費すれば型エラーになるので、より正確である。

2.3 代償

線形型は書く側に負担をかける。返信先を条件分岐の両方で使う、配列に入れる、といった素朴な操作が型で拒否される。AIPL は教育用途も持つので、**既定では暗黙の `reply` を残し、委譲したいときだけ `Reply`$\langle\tau\rangle$ を明示する**という二段構えが現実的である。

3 デッドロック自由な型システム —- 進行定理の第三の枝

3.1 現状

AIPL^{-} の進行定理は「値・簡約・`await` 待ち」の三択である。デッドロックは型では排除できないという境界を明示したものであり、「保証しないこと」の一覧にも**デッドロック自由は一般には言えない**と記した。

小林の第 2 期（1997–2007）はまさにこの枝を型で消す仕事であった。部分的デッドロック自由（LICS 1997）から始まり、暗黙型付け（CONCUR 2000）、ロックフリー（Inf. Comput. 2002）、新しい型システム（CONCUR 2006）、割り込みを持つ計算（ESOP 2007）へと進む。

3.2 AIPL に持ち込む形

要点は、通信に**義務（obligation）と能力（capability）の時刻タグ**を与え、「先に果たすべき義務」の順序が循環しないことを型で要求する、という筋である。AIPL の `now` は同期的に待つので、これがそのまま対象になる。

```
// now の連鎖に順序を宣言する。循環すると型エラー
class Front  obligation 1 { method place(k) : string { ... now stock.take(k) ... } }
```

```
class Stock  obligation 2 { method take(k)  : int    { ... } }
```

`now` は**自分より大きい**レベルへのみ許す、という単純な規律でも相互デッドロックを構文的に排除できる。より精密には小林の型システムがレベルを推論する。

所見 2. これは先のレポートで「express mode がデッドロックからの脱出手段になる」と書いた項目と**代替関係**にある。型でデッドロックを排除できるなら、脱出手段は不要になる。express mode の主要な動機のうち一つ（固まった二者を外から叩く）が消える。残る動機は「長い計算中の状態問い合わせ」と「割り込み＝実時間の本質」である。

3.3 代償

義務レベルの宣言は設計を固定する。アクターを追加するときに既存のレベルを見直す必要が出る。推論に任せると、なぜ型が付かないのかの説明が難しくなる —- 小林の型システムの実用上の難点として知られる点である。

4 汎用型システム —- 二つのプロファイルを一つの枠組みに

4.1 先のレポートの設計を見直す

先のレポートは `agent` と `realtime` という二つのプロファイルをそれぞれ制約表として与えた。効果の許容集合、禁止する構文、必須の宣言をプロファイルごとに列挙する形である。

これは実装しやすいが、**二つの型システムを手で作り込む**ことになる。性質を一つ増やすたびに両方を触ることになり、両者の関係（あるアクターが両プロファイルの境界に立つとき何が言えるか）もその場で決めることになる。

小林の **A Generic Type System for the Pi-Calculus**, *POPL 2001 / TCS 2004* は、まさにこれを避けるための仕事である。個別の性質ごとに型システムを作るのをやめ、**型システムを生成する枠組み**を与える。

4.2 AIPL に持ち込む形

中核の型システムを一つだけ持ち、プロファイルを**パラメータ**（許す効果の集合、使用型の許容する順序、有界性の述語）として与える。

	先のレポート	汎用型システムを使う場合
プロファイルの定義	制約表を手で書く	枠組みのパラメータを与える
性質の追加	両プロファイルを触る	パラメータを一つ足す
境界の扱い	方向の規律を別に決める	パラメータの合成として導かれる
健全性	プロファイルごとに示す	枠組みで一度示す

所見 3. これが本稿で最も費用対効果が高い提案である。表現力を増やす話ではなく、**先のレポートの設計をより少ない部品で実現する**話であり、しかも健全性の証明が一度で済む。

5　資源使用解析 —- capability に順序を入れる

5.1　現状の effect は順序を持たない

先のレポートの効果系は $\varepsilon \subseteq \{\texttt{mut}, \texttt{time}, \texttt{io}, \texttt{mem}, \texttt{net}, \texttt{ai}, \texttt{fs}, \texttt{log}\}$ という**平坦な集合**である。「net を使う」は言えるが「開いてから閉じる」は言えない。

小林の **Resource Usage Analysis**, *POPL 2002 / TOPLAS 2005* は資源の**使用順序**を型で追う。これを AIPL に入れると三つが同時に片づく。

5.2　三つの効用

1. **セッションプロトコルが型になる。**現在 `protocol_*` は実行時にプロトコル状態を検査している。使用型は「$!m(T)$ のあと $?R$」という順序をそのまま表せるので、先のレポートが長期目標として置いたセッション型が、独立の機構ではなく使用型の実例として得られる。
2. **実時間側の初期化順序が言える。**「装置は初期化してから使う」「`atomic` 区間の外で開いた資源を中で閉じてはならない」といった規律が型になる。
3. **エージェント側の資源リークが防げる。**モデル呼び出しのセッション、ファイル、リモート接続を「開いたら必ず閉じる」で縛れる。

```
method report(path) : unit !{fs} uses file: open . write* . close {
  var f = open(path);
  write(f, "...");
  close(f);          // 閉じ忘れると型エラー
}
```

6　高階モデル検査 —- 検証器を自作しない

6.1　等価性が意味すること

小林と Ong が示したのは、高階再帰スキームのモデル検査と交差型の型検査が**同じ問題**であるということである（LICS 2009）。そしてこの等価性は実装可能なアルゴリズムを生んだ（FoSSaCS 2011）。

AIPL にとっての意味は単純である。**AIPL 用の検証器を自作する必要がない。**AIPL のアクタープログラムを高階関数のコアへ翻訳し、検証したい性質を HFL の論理式として与えれば、既存の HFL モデル検査器・CHC ソルバが答えを出す。

検証したいこと	還元先	使う道具
到達不能性（この状態にならない）	HORS / trivial automata	高階モデル検査
時相性質（いつか返信する）	HFL	HFL 妥当性検査
状態を持つアクターの不変条件	CHC	CHC ソルバ
整数を含む性質	述語抽象 + CEGAR	同上

6.2　代償 —- 反例が AIPL の言葉で出ない

これは重い。還元して解くと、反例は**還元先の言葉**で返る。HFL の論理式や CHC の導出として返ってきたものを、AIPL のどの行のどの `now` が問題なのかへ翻訳し直す仕組みが要る。

注意 1. 別稿の小林サーベイで、OOPSLA 1994 の符号化方式について「理論的な保証が得やすいがエラーメッセージが翻訳先の言葉になる」と書いた。同じ問題が三十年後の検証にも現れる。**還元は保証を得る代わりに説明可能性を失う**という構図は変わっていない。逆写像（counterexample lifting）を最初から設計に入れる必要がある。

7　述語抽象と CEGAR —- `where` ガードを検証する

7.1　先のレポートで止まった箇所

`where` による状態依存の受理条件について、先のレポートはこう述べた—-「`where` を型で扱うには依存型・篩型が要る。現実的には受理条件は動的に検査すると割り切り、型はメッセージの形だけを保証する分業が妥当」。

これは正しいが、諦めが早い。小林の **Predicate abstraction and CEGAR for higher-order model checking**, *PLDI 2011* は、まさに無限データ領域上の述語を扱うための仕事である。

7.2　AIPL に持ち込む形

ガードを**述語**として抽象化し、「このガードが永遠に真にならない」「このガードの下で本体は安全」といった性質を CEGAR で検証する。

```
method get() : int where n > 0 ensures n >= 0 {
  n = n - 1;
  reply(n);
}
```

`ensures` は事後条件であり、精製型として検査する。ガード $n > 0$ を仮定して本体が $n \geq 0$ を保つかは、述語抽象と CEGAR で自動判定できる範囲に入る。

所見 4. これで先のレポートが挙げた**「恒偽のガードを持つメソッドは決して受理されないが型検査は通る」という静かに壊れる誤り**も検出できる。ガードの充足可能性は述語の充足可能性であり、SMT で判定できる。「明らかに恒偽なガードは警告できる」と書いた箇所が、一般の述語に拡張される。

8　精製型とサイズ型 —- 期限と応答時間の上界

8.1　先のレポートの弱点

先のレポートはプロファイル R について「停止性と応答時間の上界が主張できる」と書いた。停止性は構文的な有界性（`while` 禁止、再帰禁止）から出る。しかし**応答時間の上界は、構文からは出ない**。「解析できる」と書いただけで、何で解析するかを示していない。

8.2 二つの道具

- **Sized Types with Usages for Parallel Complexity of Pi-Calculus Processes**, *CONCUR 2021* —- π 計算プロセスの**並列計算量**をサイズ型と使用型で与える。これはまさに「このメッセージ処理は何ステップで終わるか」を型で言う仕事である。
- **Termination Analysis for the π-Calculus by Reduction to Sequential Program Termination**, *APLAS 2021* —- 停止性を逐次プログラムの停止性へ還元する。構文的な禁止より**緩い**条件で停止性が言える。

```
// deadline は宣言であって主張ではない。サイズ型で検査できるようにする
periodic method tick() : unit !{io, time}
    period 1000us deadline 800us priority 10
    cost O(1) {          // ← サイズ型が検査する上界
  ...
}
```

所見 5. サイズ型を入れると、実時間プロファイルの `deadline` 宣言が**外部のスケジューリング解析器に渡す情報**から**型検査器が検査する主張**に変わる。これは先のレポートの「時間の宣言は型システムの仕事ではなくスケジューリング可能性解析の仕事である」という記述を、部分的に覆すものである。少なくとも**最悪実行時間の上界は型で扱える**。

9 CHC への還元と所有権 —- 状態を持つアクター

AIPL のアクターは可変状態を持つ。状態を持つプログラムの自動検証で近年最も成功しているのが **RustHorn**, *ESOP 2020 / TOPLAS 2021* の路線である。Rust の所有権が別名（aliasing）を排除することを利用して、ポインタを持つプログラムの検証を制約付きホーン節へ還元する。

AIPL には既に近い構造がある。アクターは状態を共有せず、外部作用を持つアクターは同時に一つしか存在してはならない —- これは別稿の DOFBOT 設計で「フェイルオーバとは Capability の移譲そのものである」と述べた性質であり、**所有権と同じ形**をしている。

所見 6. したがって AIPL は RustHorn 方式の還元に**最初から向いている**。アクターの状態は所有されており別名がないので、状態の変化を CHC の述語として直接書ける。`act` 効果（実際に駆動するアクターだけが持つ）は所有権の宣言として読み替えられる。

10 時相論理 —- 実時間仕様を書いて解く

実時間プロファイルの `deadline` や「いつか必ず応答する」は本質的に**時相性質**である。小林の第 4 期はまさにこれを扱う —- **Temporal verification of higher-order functional programs**, *POPL 2016*、**A Temporal Logic for Higher-Order Functional Programs**, *SAS 2019*、そして HFL モデル検査による検証（ESOP 2018 以降）。

AIPL に持ち込むなら、性質を書く言語を用意し、HFL へ還元する。

```
// アクターに時相仕様を付ける
class Balancer
```

```
  spec always (sensor_updated -> eventually_within 1ms motor_written)
{ ... }
```

注意 2. ここで注意すべきは、**時相論理は「書ける」ようになるだけで「守られる」保証は還元先の判定に依存する**ことである。HFL の妥当性検査は決定可能な断片に限られ、一般には近似になる。仕様を書けることと検証できることは別である。

11　gradual 化 —- 単位と形状の段階的導入

先のレポートは実時間側に物理単位（`float⟨m/s⟩`）を提案した。これは事故を減らすが、既存コードすべてに単位を書かせるのは移行が重い。

Gradual Tensor Shape Checking, *ESOP 2023* は、テンソルの形という似た問題に対して段階的型付けを使う。同じ形で、単位も**書いたところだけ検査する**にできる。

```
var v : float<m/s> = read_speed();   // 単位あり
var w = read_gyro();                 // 単位なし（dynamic）
var x = v + w;                       // 境界で実行時検査、または警告
```

移行コストが下がる代わりに、境界で実行時検査が入る。実時間側では実行時検査そのものがコストなので、**プロファイル R では gradual を許さず全面注釈を要求する**のが妥当である。

第 III 部
統合した言語仕様案

12　三層に分ける

小林の道具は性質が異なる。同じ層に並べると設計が混乱するので、三層に分けて整理する。

層	内容	由来
L1 コア型	基底型、レコード、直和、アクター型、`future`、`Reply⟨τ⟩`、戻り値型	先のレポートの理想仕様
L2 使用型	線形性、効果、資源の使用順序、義務レベル、サイズ	線形型（POPL 1996）、資源使用解析（POPL 2002）、デッドロック自由（LICS 1997）、サイズ型（CONCUR 2021）
L3 検証層	ガードの述語、事後条件、時相仕様	述語抽象 + CEGAR（PLDI 2011）、HFL（ESOP 2018）、精製型 + モデル検査（PEPM 2019）

そして **L1 と L2 は汎用型システムの枠組みで一つにまとめ**（POPL 2001）、プロファイルはそのパラメータとする。L3 は型検査の外側で、還元によって外部ソルバに委ねる。

	決定可能か	誤りの説明	実装
L1 コア型	決定可能・線形時間近く	AIPL の言葉で出る	自前
L2 使用型	決定可能（設計次第）	説明が難しくなる	自前
L3 検証層	一般に不完全・近似	還元先の言葉で出る	外部ソルバ

この表が本稿の実務上の結論である。**L1 は必ず自前で持ち、L3 は必ず外部に出し、L2 がどこまで自前で背負えるかが設計判断**になる。

13　改訂した仕様（先のレポートとの差分）

項目	先のレポート	本稿の改訂
返信先	暗黙。委譲できない	既定は暗黙。`Reply`$\langle\tau\rangle$ を**線形型**として明示すれば委譲可
`reply` の回数	構文的に走査して数える	**線形性で保証**。走査を廃止
デッドロック	型では言えない。express で脱出	**義務レベルで型が排除**。express の動機が一つ減る
プロファイル	制約表を二つ手で書く	**汎用型システムのパラメータ**として与える
効果	平坦な集合	**使用順序を持つ**（資源使用型）。セッション型を包含
`where`	型は形だけ、条件は動的	**述語抽象と CEGAR** で条件も検証。恒偽ガードを検出
応答時間	外部の解析器に渡す情報	**サイズ型で型検査**（上界のみ）
停止性	構文的禁止（`while` 不可）	**逐次プログラムへの還元**でより緩い条件に
時相仕様	書けない	`spec` で書き、**HFL へ還元**して解く
物理単位	全面注釈	エージェント側は **gradual**、実時間側は全面注釈
状態の検証	触れていない	所有権を利用して **CHC へ還元**
多相メソッド	行き詰まり	依然として行き詰まり（§18）

14　何が「言える」ようになるか

性質	現状	先のレポート	本稿
型健全性	言える	言える	言える（層ごと）
返信がちょうど一度	注釈時のみ	常に	常に、**委譲しても**
デッドロック自由	言えない	期限付き待ちなら有限時間	**型で排除できる**
停止性	言えない	プロファイル R で	**より広い範囲で**
応答時間の上界	言えない	解析できる（手段未定）	**型で言える**
資源の使用順序	言えない	言えない	**言える**
状態の不変条件	言えない	言えない	**CHC で検証できる**
時相性質	書けない	書けない	**書けて、近似的に検証できる**
外部流出しないこと	言えない	効果で言える	効果で言える
公平性	言えない	悪化する	悪化する（変わらず）

第 IV 部
メリットとデメリット

15　メリット

1. **先のレポートが残した穴が三つ閉じる。**返信先の一級化（線形型）、`where` の検証（述語抽象と CEGAR）、応答時間の上界（サイズ型）。いずれも「要るのは分かるが手段がない」と書いて止めた箇所である。
2. **二つのプロファイルを手で作り込まなくてよくなる。**汎用型システムの枠組みで、プロファイルはパラメータになる。健全性の証明も一度で済む。
3. **検証器を自作しなくてよい。**HFL / CHC への還元により、既存のソルバに載る。三十年の研究成果が使える。
4. **セッション型が独立の機構でなくなる。**資源使用型の実例として得られる。実装が一つ減る。
5. **デッドロックを型で排除できれば express mode の動機が減る。**実時間側で割り込みが必要なのは変わらないが、「固まった二者を外から叩く」という用途は消える。
6. **AIPL は RustHorn 方式に向いている。**アクターが状態を共有せず、外部作用が単一であるという性質は所有権と同じ形をしており、CHC への還元がしやすい。

16　デメリットと危険

1. **反例が AIPL の言葉で出なくなる。**最も重い代償である。還元して解く以上、反例は還元先の言葉で返る。counterexample lifting を最初から設計に入れないと、「検証器は false と言うがどこが悪いか分からない」状態になる。三十年前の符号化方式と同じ問題である。
2. **健全性の機械検証が現実的でなくなる。**現在の Coq 証明が小さく収まっているのは型システムが小さいからである。線形性・使用型・義務レベル・サイズ型を重ねると、機械検証は年単位の仕事になる。**L2 をどこまで背負うかは、証明をどこまで維持するかと直結する。**
3. **決定不能・高計算量に近づく。**交差型の型推論は一般に決定不能、高階モデル検査は n 階で n 重

指数時間完全である。実入力では実用的に振る舞うという知見はあるが、**保証**ではない。型検査が終わらない可能性を運用として受け入れる必要がある。

4. **学習コストが跳ね上がる。**AIPL は OS 記述や教材にも使われる。義務レベル、線形環境の分割、使用型の順序記述は難しい。既定を素直に保ち、必要なところだけ重い型を使わせる二段構えが不可欠である。
5. **外部ソルバへの依存。**HFL モデル検査器、CHC ソルバ、SMT ソルバの可用性・保守・性能に検証能力が縛られる。ソルバが更新されると結果が変わりうる。
6. **道具が実時間側に偏っている。**次節で述べるが、これは設計判断に直接影響する。

17 非対称性 —- 道具はエージェント側に効きにくい

所見 7. 本稿で整理した道具のほぼすべてが**実時間プロファイル側に効き、AI エージェント側には効きが薄い**。

道具	プロファイル R	プロファイル A
線形型（返信先）	○	○
デッドロック自由な型	○	△（外部呼び出しが待ちを作る）
資源使用型	○	○
サイズ型（応答時間）	○	×（モデル呼び出しの時間は型で言えない）
停止性の還元	○	×（外部応答に依存）
述語抽象 + CEGAR	○	△（モデル出力は述語で書けない）
HFL 時相検証	○	△
CHC / 所有権	○	○
gradual 単位	×（全面注釈）	○

理由は単純である。小林の道具はいずれも**プログラムの構造から性質を導く**ものであり、外部の非決定的な応答に依存する性質は扱えない。エージェント側で「モデルが 5 秒以内に返る」は型では言えない。

したがって設計上の帰結は次のようになる。

- **プロファイル R には重い型を入れる価値が高い。**フィジカル AI の安全性が型で言えるようになる。
- **プロファイル A では、型で言うのは「形」と「効果」までとし、時間や成否は失敗を型に持つことで扱う**（期限付き `now`、`Result` 型）。これは先のレポートの結論のままでよい。

18 採る／条件付き／見送る

道具	判断	理由
汎用型システムの枠組み	**採る**	最も費用対効果が高い。プロファイルを部品化し、証明が一度で済む
線形型（`Reply`）	**採る**	委譲を許しつつ「ちょうど一度」を保つ唯一の道。構文検査を廃止できる
資源使用型	**採る**	セッション型を包含する。実時間側の初期化順序にも効く
gradual 単位	**採る**	移行コストが下がる。プロファイル A のみ

道具	判断	理由
CHC への還元	**採る**	AIPL の構造が向いている。外部ソルバに載る
デッドロック自由な型	条件付き	型で排除できれば大きいが、義務レベルの推論は説明が難しい。まず**明示宣言**のみで始める
述語抽象 + CEGAR	条件付き	`where` の検証に必要だが、counterexample lifting を同時に設計できるかが条件
サイズ型（応答時間）	条件付き	実時間側の価値は高いが、型の記述が重い。`cost O(1)` 程度の粗い注釈から始める
HFL 時相検証	条件付き	仕様を書けるようにするのは安い。検証は近似であることを明示する
交差型による完全な高階モデル検査	**見送る**	型推論が一般に決定不能。AIPL の対象（アクターと通信）に対して必要な表現力を超えている
多相メソッド	**見送る**	ρ を型スキームに含めると呼び出しごとに差し替えられる問題が残る。単相のままが証明との相性で妥当
所有権の完全な型システム	**見送る**	CHC への還元に必要な範囲（別名の無さ）は既に成立している。Rust 相当の借用検査を作る必要はない

第 V 部
段階的な道筋

19　順序

	内容	位置づけ
1	効果注釈の静的化	先のレポートの最小到達点。既存 capability 分類を使う
2	期限付き `now` / `await` のみ	同上。進行性が強まる
3	`interface` 宣言	同上。`any` が消える
4	**汎用型システムへの再編**	1–3 を枠組みのパラメータとして書き直す。ここが分岐点
5	資源使用型（順序つき効果）	セッション型を包含。4 のパラメータとして入る
6	`Reply`$\langle\tau\rangle$ の線形型	委譲が書けるようになる。構文的な回数検査を廃止
7	`where` + 述語抽象・CEGAR	counterexample lifting と同時に
8	CHC への還元と状態の不変条件	外部ソルバに載せる
9	義務レベル（明示宣言のみ）	デッドロックを型で排除
10	サイズ型と `cost` 注釈	実時間側の応答時間
11	`spec` と HFL 還元	時相仕様。近似であることを明示

段階 4 が分岐点である。ここで汎用型システムに再編しておかないと、5 以降の道具を足すたびに二つのプロファイルを手で触ることになる。

20　最小の到達点

先のレポートは四つ（効果の静的化、期限付き待ち、`interface`、有界性検査）を挙げた。本稿はそれに**二つ**を加える。

(5) **汎用型システムへの再編** —- 以後の拡張をすべて同じ枠組みに載せられるようにする。
(6) `Reply`$\langle\tau\rangle$ **の線形型** —- ABCL/1 から失った三つの機構のうち、返信先の一級性を取り戻す。しかも「ちょうど一度」を保ったまま。

この六つが揃った時点で、AIPL は *ABCL/1* **の表現力の一部を型安全に取り戻した、二つの目的に耐える言語**になる。

21　まとめ

1. **「全て取り入れる」ことは可能でもなく望ましくもない。**交差型の完全な高階モデル検査は決定不能に近く、AIPL の対象に対して過剰である。しかし**道具箱の選び方は明確に決まる** —- 先のレポートが残した穴に噛み合うものを採る。
2. **最も効くのは汎用型システムの枠組みである。**表現力を増やす話ではなく、二つのプロファイルを部品化し、健全性の証明を一度で済ませる話である。
3. **線形型が返信先の一級化を可能にする。**ABCL/1 から失った機構のうち最も本質的なものが、「ちょうど一度」を壊さずに戻る。
4. **L1 ／ L2 ／ L3 の三層に分けて考える。**L1 は自前で持ち、L3 は外部ソルバに出す。L2 をどこまで背負うかが設計判断であり、それは**健全性の機械検証をどこまで維持するか**と同じ問いである。
5. **最大の代償は説明可能性である。**還元して解くと反例が AIPL の言葉で出ない。counterexample lifting を後付けにすると、「検証器は落ちるがどこが悪いか分からない」言語になる。
6. **道具は実時間側に偏っている。**フィジカル AI の安全性は型で大きく前進できるが、AI エージェント側で時間や成否を型で言うことは原理的に難しい。エージェント側は**失敗を型に持つ**という先のレポートの結論のままでよい。

小林の三十余年は「論理を言語に入れる」ことから「論理を型に翻訳して自動で解く」ことへの移動であった。AIPL は現在、その最初の段 —- 型を言語に入れる —- を終えたところにいる。本稿が示したのは、次の段へ進む道が既に舗装されているということ、そしてその道を全部歩く必要はないということである。

参考

- 本リポジトリ `docs/aipl_abcl1_features_ja.pdf` —-「AIPL の言語仕様の検討」(29 ページ)。本稿の直接の前提。`https://kodamay.org/reports/2026-07-30_aipl_language_design.pdf`
- `~/projects/semantics/kobayashi_research_survey_ja.pdf` —-「小林直樹先生の研究の流

れ」。本稿で使う書誌の出典。

- 本リポジトリ `docs/aipl_soundness_ja.pdf` —- AIPL^{-} の型健全性・型安全性の Coq 検証。「保証しないこと」の一覧。
- 本リポジトリ `docs/aipl_vs_abcl1_ja.pdf` —- ABCL/1 との比較。失った三つの機構。
- 書誌は DBLP `https://dblp.org/pid/k/NaokiKobayashi.xml`（`Naoki Kobayashi 0001`）で確認した。