ベアメタル Xinu 上の型推論クリーン AIPL・哲学者の食事

単一クラス化による occurs check 回避、50 回食事ベンチ、GC 実行

drone-hil / xinu-rpi4 × AIPL (aipl2c)

2026 年 6 月 5 日

概要

先の分散 N-Queens に続き、古典的な**哲学者の食事**問題を**型推論クリーン**な AIPL で記述し、Raspberry Pi 4 のベアメタル Embedded Xinu の JIT ランタイムで実行した。本問題は AIPL の Hindley–Milner 型推論にとって N-Queens より難しい：フォークと哲学者を素朴に 2 クラスに分けると `parent`/参照が相互再帰し、**occurs check** が無限型を作って型付けに失敗する。N-Queens で有効だった「単一自己再帰クラス」の手法を踏襲し、フォーク・哲学者・テーブルを**単一クラス `D`**（実行時 `kind`）に畳み、さらに actor 参照を渡すメソッド引数とローカル変数を**クラス名 `:` `D` で nominal 固定**することで、型推論クリーン（`aipl2c --check` を通過）にできた。設計上**デッドロックフリー（順序付き獲得）＋ライブロックフリー（待ち行列フォーク）**で、実機で**全哲学者が厳密に同数の食事（餓死なし・完全公平）**を達成した。50 回食事を計測し、N-Queens のように suicide せず常駐する 11 個のアクターを **GC で回収**した（約 220–306 ms、対象 11 個）。

1　型推論の壁と、その回避

2 クラス（`Fork` と `Philosopher`）で素直に書くと、`Philosopher.lo_fork: Fork` かつ `Fork.waiter: Philosopher` という**相互再帰**になり、HM の単一化が $\alpha = \{\cdots : \alpha \ldots\}$ を要求して **occurs check failed** となる（フィールドや `:` `any` 注釈では回避できなかった）。N-Queens の `Node` が通ったのは**単一クラスの自己再帰**（`Node`→`Node`）だったからである。そこで:

1. **単一クラス `D`**：フォーク役・哲学者役・テーブル役を 1 クラスに畳む。全参照が `D`→`D` の自己再帰になる。
2. **`:` `D` の nominal 注釈**：actor を受け取るメソッド引数（`request(phil: D)`, `init_b(lf: D, hf: D)`）と `setup` のローカル変数（`var f0: D = new D();` 等）をクラス名で明示注釈し、推論が構造型へ展開して occurs check に落ちるのを防ぐ。

これで `aipl2c --check` を型エラーなしで通過する（型推論クリーン）。

Listing 1　`dining_xinu.abcl` の中核（待ち行列フォーク＋順序付き獲得）。

```
method request(phil: D) {                    // フォーク役: 空なら即許可、塞がっていれば待たせる
  if (held == 0) { held = 1; send phil.granted(idx); }
  else { has_waiter = 1; waiter = phil; }
```

```
}
method release(unused) {                          // 解放時、待ち哲学者へ直接フォークを手渡す
  held = 0;
  if (has_waiter == 1) { has_waiter = 0; held = 1; send waiter.granted(idx); }
}
method granted(forkIdx: int) {                    // 哲学者役: 低->高の順で取得し、両方揃ったら食事
  if (forkIdx == lo_idx) { send hi_fork.request(self); }
  else {
    send lo_fork.release(0);  send hi_fork.release(0);
    meals = meals + 1;
    if (meals < quota) { send self.dine(); }  // quota まで自己駆動で食べ続ける
  }
}
```

2　正しさ：デッドロック&ライブロックフリー

順序付き獲得：各哲学者は必ず低 index のフォークを先に取る（哲学者 4 のフォークは $\{4, 0\}$ なので lo=0, hi=4 と逆順になり循環待ちを断つ）⇒ デッドロックなし。**待ち行列フォーク**：塞がっているフォークへの要求は 1 つだけ待たせ（5 環では各フォークの競合相手は高々 1 人）、解放時に手渡す。ビジーリトライ無し ⇒ ライブロックなし。実機の結果（表 1）では**全哲学者が厳密に `quota` 回ずつ**食事しており（25=5×5, 50=10×5, 100=20×5）、**餓死ゼロ・完全公平**が確認できる。

3　手法と JIT ランタイムの特性

コンパイルは N-Queens と同じく `aipl2c --xinu-jit`（型検査は `--check` で別途、クリーン）、生成 C の `v_nil` を `v_int(0)` に 1 箇所置換、`POST /actor/load` で Pi-4 JIT へ。**重要な特性**：JIT のメッセージポンプは 1 回の `/actor/load` で**有限予算**を消化して quiesce する。N-Queens は**有限木**なので 1 回の load で全て drain したが、哲学者の食事は**持続ループ**なので、各哲学者が `quota` に達するまで `dine` を HTTP ゲートウェイ越しに**再ポーク**して駆動する。したがって計測 wall-clock は **HTTP 再ポークのラウンドトリップが支配的**で、生の食事計算時間ではない点に注意。

4　結果

5　GC の実行

N-Queens のソルバは `suicide()` で自滅するが、哲学者の食事のアクター（5 フォーク＋ 5 哲学者＋テーブル＝ **11 個**）は**常駐**する。各ランの後に `/api/actors-gc?threshold_ms=0&dry=0` を呼んでアクター GC を走らせた。GC は毎回 `total=11, targets=11`（全 11 個が回収対象）を報告し、所要は **220–306 ms**。これは「哲学者の食事は GC でアクター寿命を管理する」という、

表 1　実機・哲学者の食事（Pi 4 Xinu JIT、HTTP 駆動）と GC。

総食数	quota	各哲学者 (5 人)	合計	wall-clock (ms)	pokes	ms/食	GC (ms)
25	5	5, 5, 5, 5, 5	25	10659	20	426	227
50	10	10, 10, 10, 10, 10	50	23385	45	468	220
100	20	20, 20, 20, 20, 20	100	46094	95	461	306

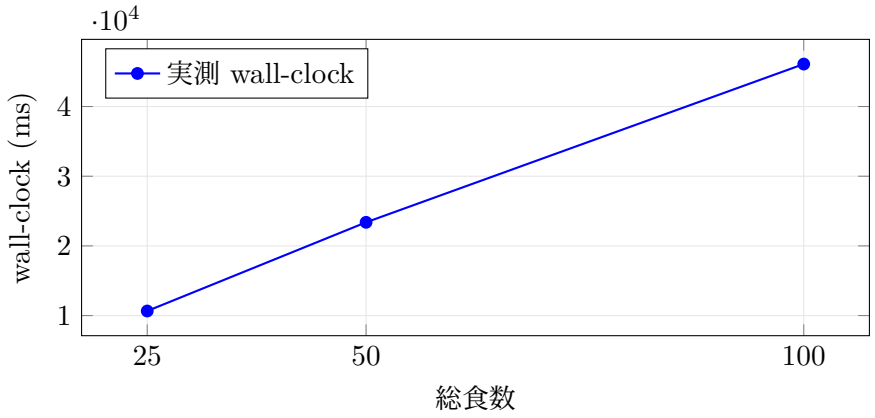

図 1　食事数に対してほぼ線形（≈ 460 ms/食）。傾きは主に HTTP 再ポークのラウンドトリップ（≈ 0.5 s/poke）に由来する。

N-Queens（自滅）との対照を具体的に示す。

6　考察と限界

- **実測**：型推論クリーン検査の通過、`--xinu-jit` の C 生成、実機での**正しさ**（各哲学者の食事数）・**完全公平**・GC の回収数と所要時間。
- **HTTP 支配**：wall-clock は再ポークのラウンドトリップが支配的で、生の食事計算時間ではない（JIT ポンプが load ごとに quiesce する特性のため）。≈ 460 ms/食はほぼ HTTP コスト。
- **単一ラン**・5 哲学者固定・1 ノード（メッシュ分散版は今後）。

7　結論

哲学者の食事は AIPL の HM 型推論にとって N-Queens より難しい（相互再帰アクターの occurs check）。**単一クラス化＋ nominal 注釈**でこれを克服し、型推論クリーンかつデッドロック&ライブロックフリーな実装が実機 Pi-4 Xinu JIT 上で**完全公平に 50 回（および 25/100 回）食事**を達成、常駐アクターを **GC で回収**した。N-Queens（計算律速・自滅アクター・1 load で drain）との対照として、哲学者の食事は**協調律速・常駐アクター・持続ループ（再駆動が必要）・GC 管理**という性格を持つ。次段は、この食事ループを WiFi メッシュ越しに分散し（境界フォークをノード間で調停）、通信律速の分散コーディネーションを実測することである。