3 人の賢者の復讐

竹内郁雄 [1a]

概要　2016 年のプログラミングシンポジウムで発表した「3 人の賢者の問題」とそれを解くプログラムには致命的な論理誤りと，他の賢者の思考を推論するプログラミング手法の選択ミスがあった．ここではその誤りを正すとともに，このゲームに適合した傍目賢者シミュレーション手法と，新しく分かった興味深い数理について述べる．

キーワード　3 人の賢者の問題，他者の思考の推論，傍目賢者，フィボナッチ数列

1. はじめに

私が 2014 年 7 月号の「数学セミナー」誌に出題した「3 人の賢者の問題」[1] は，Douglas Hofstadter の凡庸（mediocre）[2] をベースにした不完全情報ゲームである．3 人の賢者が順番に，ゲームについて分かったことを発言していくという仕立てになっている．全員がずっと「分からない」と発言し続けると，そのうち誰かが「誰それの勝ち（あるいは負け）」と宣言できるようになるという不思議なゲームである．これは各賢者が他の賢者の「分からない」発言から情報が得られるという原理に基づく．一見情報がないのに，情報が次第に明らかになってくるジャンルの問題である．

2016 年冬のプログラミングシンポジウムで，私はこの問題を解くプログラムについて発表した [3]．一見簡単な問題のように見えるが，これを独立した賢者をオブジェクト（エージェント）でモデル化するというプログラミングが予想以上に困難だった．つまり，独立した 3 人の賢者の思考を忠実にモデル化することは，まさに表題のとおり，愚者には難しすぎた．プログラムはエレガントからほど遠いものになった．しかも，致命的な論理誤りまであった．

結局，発表内容はグダグダとなり，実際「知っている/知らないという論理関係は明確，かつ整数の大小関係だけなので，それだけをベースにした新しいパラダイムによる解法があるのではないか？」という結論で締めくくった．

案の定，その日のうちに，東京大学の田中哲朗氏が SMT (Satisfiability Modulo Theories) ソルバによってこの問題を解いてしまった [4]．まさに脱帽である．

これでいいかと思っていたが，やはり悔しいので，2019 年 6 月始めに気を取り直して，発想を変えてプログラムを書いてみたところ，プログラムの論理構造がすっきりして，かつ高速になった．その結果，興味深い数理も発見された．

2. 3 人の賢者の問題

「数学セミナー」誌に出題した文章をベースにより問題が明確に分かるようにしたものを紹介する．オリジナルの問題（問題 1）とその発展形（問題 2），そして任意のカードの組合せに対して正確に賢者をシミュレートするプログラミングの問題（問題 3）について述べる．

3 人の賢者 A, B, C が図 1 のように円卓の周りに時計回りに並び，カード遊びをしている．N (≥ 5) 枚の，1 から N までの数が書かれたカードが円卓上に裏向きに置いてあり，賢者たちは 1 枚ずつ取る．大小比較で中間の数を取った賢者が勝つ．しかし，一斉にカードをオープンするだけでは脳がへたれるので，次のようにルールを変えた．

賢者たちは取ったカードを見たあと，自分の右隣りに見せる．そして A から順に時計回りに（A, B, C の順に），勝敗について確定したこと，つまり「誰の勝ち」あるいは敗者しか決まらないときは「誰の負け」と**宣言**する．勝敗に関する新しい

[1] 東京大学名誉教授
[a] nue@nue.org

確定情報がないときは**ニッコリ**と会釈する．以降，宣言とニッコリを併せて**発言**と呼ぶ．

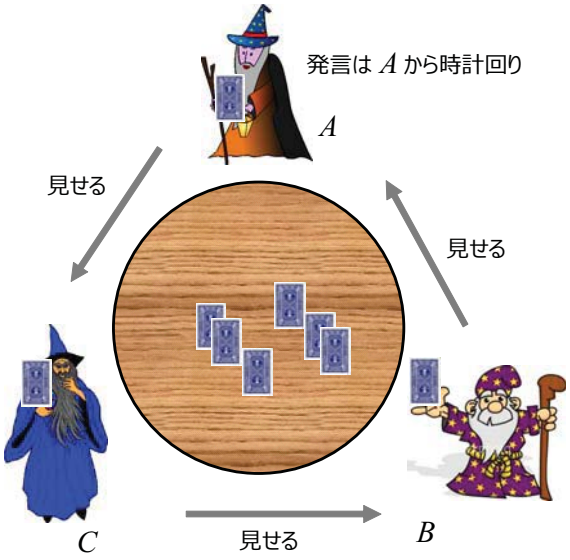

図 1. 3 人の賢者の問題

問題 1．最初からニッコリが連続したあと，ある賢者が「わしの勝ちぢゃ．最初からのニッコリがこれより長く連続することはない」と宣言した．誰がどのようなカードの組合せで勝ったか？

問題 2．問題 1 において，そのようなカードの組合せの数が最小になる N をすべて求めよ．

問題 3．3 人の賢者 A, B, C のカードがそれぞれ a, b, c $(1 \leq a, b, c \leq N)$ だったときの賢者の発言をシミュレートするプログラムを書け．N, a, b, c が引数となる．

図 1 は $N = 9$ の場合を描いている．$N \geq 5$ としたのは，最初にニッコリが出る可能性を保証するためである．この問題の奥深さは，3 人の数のうち中間の数が勝利するという，D. Hofstadter が Scientific American に連載した Metamagical Themas で紹介した凡庸（その後，月並と改称）に由来する．これはその後私がトランプで遊べる最中限というゲームに変形し，愛好者によって分析されている [5]．

この問題の解き方の妙を説明するために，一番簡単な $N = 5$ の場合を考えよう．最初に発言をする A は自分のカードの数 a と B のカードの数 b を知っているので，a と b の組合せ (a, b) によって以下の発言をする．

(1,2) → B の勝ち	(1,3) → A の負け
(1,4) → A の負け	(1,5) → C の勝ち
(2,1) → A の勝ち	(2,3) → C の負け
(2,4) → ニッコリ	(2,5) → B の負け
(3,1) → B の負け	(3,2) → C の負け
(3,4) → C の負け	(3,5) → B の負け
(4,1) → B の負け	(4,2) → ニッコリ
(4,3) → C の負け	(4,5) → A の勝ち
(5,1) → C の勝ち	(5,2) → A の負け
(5,3) → A の負け	(5,4) → B の勝ち

このうち私が前回の発表で見逃していたのは，C の負けが宣言できる場合，a と b が連続していて，その間に c が入り得ない，つまり $|a - b| = 1$ のときである．なんとも初歩的なミス．最初から大きい N で考えていたのが敗因で，こういう小さい N から考察を始めるべきだった．

3. 問題 1 を解くプログラム

前回発表したプログラムは，シミュレーションをなるべく現実に沿ったものにする意図で，カプセル化による情報隠蔽を素直に表現した賢者というオブジェクトを定義した．つまり，賢者の推論を個々の賢者のみが知る情報をベースに組み立てようとした．これが苦労の元だった．

それぞれの賢者は自分のカード *mine* と自分の左隣りのカード *left* を知っているが，右隣りのカード *right* は知らないので，初期状態では *right* を $\{1, 2, \cdots, N\} - \{mine, left\}$ という集合（リスト）で表現する．賢者が発言するたびに *right* の集合が小さくなっていくようにする方針だった．ところがこれがやさしくない．*right* を決めるためには左隣りの賢者が内部に持つ *right*（つまり，自分を左隣りはどう見ているか）などの情報も推測しなければならない．これが，なんとも扱いにくい難しさを生んだのである．

復讐を試みた今回は，新たに第三者の賢者，**傍目** (おかめ) **賢者**の視点で個々の賢者の推論を記述することにした．傍目賢者は，3 人のカードの組合せ（以下，**配牌**と呼ぶ）は見えないが，配牌の可能性はすべて掌握できる．その前提で，右隣り

の賢者の情報を持たない個々の賢者の合理的な推論をシミュレートするのである．

よく考えると（考えなくても），問題 1 は，最初から最も長くニッコリが連続する配牌を求めるという一種のメタ問題である．だから自分と左隣りのカードを知っているとするオブジェクト指向よりも，こうしたほうがずっと簡単に書けるのは当然だった．ただし，最初にすべての可能な，すなわち $N(N-1)(N-2)$ 通りの配牌をメモリに配置する必要がある．

Lisp で書いたプログラムの構造は単純である．上述のようにまず可能なカードの配牌のリスト *All-hands* を作る．

```
((8 7 6) (8 7 5) ... (1 2 4) (1 2 3))
```

それぞれの 3 つ組が賢者 A, B, C の配牌 (a, b, c) を表す．A の発言により，*All-hands* は短くなる．つまり，論理的に不要な配牌が削除される．初期値は賢者 A 向けの構造だが，B のターンではすべての配牌を (b, c, a) の並びにし，かつ B の思考を簡単にするために (b, c, a) の辞書式降順にソートする．以下同様である．

賢者は，自分のターンでの *All-hands* を見てどんな発言ができるかを考える．同じ (*mine,left*) に対して，見えない *right* を集めた {(*mine,left,right*)} に矛盾のない大小関係があれば，ニッコリ以外の発言が可能となる．例えば，(*mine,left*) $= (6, 4)$ の配牌の集合（グループ）が

```
{(6 4 3), (6 4 2), (6 4 1)}
```

のときは *left* の勝利を宣言できる．負けの宣言ができる場合もある．例えば，

```
{(6 4 8), (6 4 7), (6 4 5)}.
```

は *left* の負けである．これが

```
{(6 4 7), (6 4 5), (6 4 3)}
```

だと，このグループについては何も言えない．勝ち負けを宣言できた配牌をどんどん *All-hands* から削除していく．プログラムは各賢者のターンについてこれを丹念に繰り返し，誰かが勝利宣言するまで続ける．

繰り返すが，ここでは個々の賢者の札の具体的な数は知らなくてよい．可能な配牌の集合だけを考えている．

4. 問題 2 を解く数理

問題 1 のプログラムをいろいろな N で走らせると，最初からニッコリが最も長く連続したあとに自分の勝利を宣言できるときの配牌が 2 通りになる N_σ に特徴があることに気がついた．

$N_\sigma = 5, 9, 17, 29, 49, 81$ $(\sigma = 1, 2, 3, 4, 5, 6)$ となるのである．公差を取ると，$4, 8, 12, 20, 32$ である．これはまさに（一般化された）フィボナッチ数列である．

これを実験的に確かめるために，上述のプログラムのメモリ節約を行った．最もメモリを消費する *All-hands* の要素を 3 つ組のリストではなく，24 ビットの非符号整数で表現することにしたのである．プログラムの変更は容易だった．そこで，予想通りの $N_7 = 133$ を確かめることができた．

ここまで来ると，実験科学ではなく，理論的考察の対象となる．以下にその証明を紹介する．これは「数学セミナー」誌 2019 年 11 月号に紹介したものである [6]．

カード枚数は N. まず，連続ニッコリ回数 σ が 1，つまり A がニッコリした場合を考える．A には a と b が見えている．漸化式を作るために，$a < b$ と $b < a$ の場合に分けて考え，1 から N の整数区間 $[1, N]$ 上で議論する．区間の表記は，角ガッコの隣りの数はその数を含む，丸カッコの隣りの数はその数を含まないことを意味する．$a < b$ なら，区間 $[1, a)$，区間 (a, b)，区間 $(b, N]$ のいずれにも c が入り得ることがニッコリの条件である（図 2 上）．$b < a$ の場合も同様（図 2 下）．これらの情報がすべての賢者に公知になることに注意．

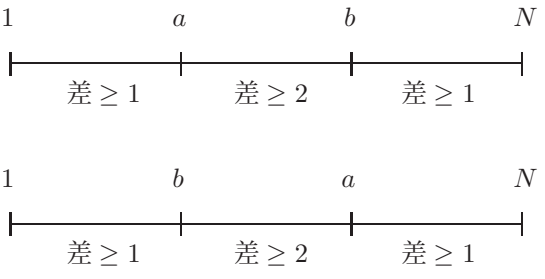

図 2. $\sigma = 1$ のときの区間の図

$\sigma = 2$，つまり次に B がニッコリしたときの整数区間を考える．B には b と c が見えている．ま

ず，$b < c$ の場合を考える．区間 $[1, b)$ に a が入るには，図 2 が示している不等式より，$b - 1 = (a-1) + (b-a) \geq 3$ が得られる．区間 (b, c) に a が入るには，同様に，$c - b = (a-b) + (c-a) \geq 3$（$c - a \geq 1$ は自明）．また，区間 $(c, N]$ に a が入るには $N - c = (a-c) + (N-a) \geq 2$．これらを総合すると図 3 上の整数区間になる．$c < b$ の場合も同様に 3 つの不等式が得られ，図 3 下の整数区間になる．差の最小値の並びが反転している．

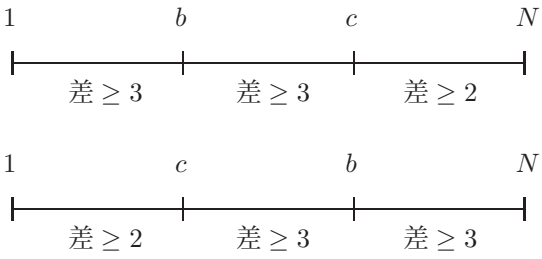

図 3. $\sigma = 2$ のときの区間の図

$\sigma = 3$ のときは，$\sigma = 2$ のときとほぼ同様だが，後述の漸化式が見えてくるように，再度説明する．

C には c と a が見えている．まず，$c < a$ の場合を考える．区間 $[1, c)$ に b が入るには，図 3 が示している不等式より，$c - 1 = (b-1) + (c-b) \geq 6$ が得られる．区間 (c, a) に b が入るには，図 2,3 より $a - c = (b-c) + (a-b) \geq 5$．区間 $(a, N]$ に b が入るには，図 2,3 より $N - a = (b-a) + (N-b) \geq 5$．これらを総合すると図 4 上の整数区間が得られる．$a < c$ の場合も同様で，図 4 下の整数区間が得られる．

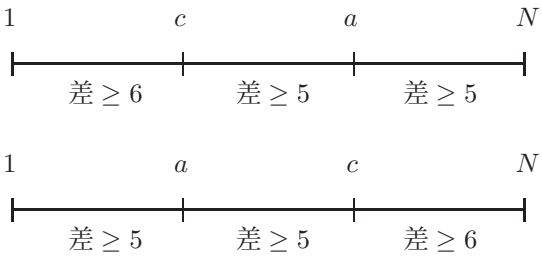

図 4. $\sigma = 3$ のときの区間の図

$\sigma = 1, 2, 3$ の整数区間において，すべての不等号が等号になるときが，配牌の可能性に関して最も「遊びがない」ことは容易に分かる．実際，この時点で次のターンの賢者が自分の勝利を宣言できる可能性は 2 通りになる．なぜなら，全部等号なので，自分の右隣りが確定し，かつ見えている左隣りも図に示した 6 つの区間いずれかのユニークな位置に入るからである．結局，最長連続ニッコリ回数 σ に対して「遊びがない最小のカード枚数 N_σ」を $\sigma = 1$ から順に書くと，$5, 9, 17$ となる．

もっと大きい σ については，a, b, c が循環的に入れ替わるので，以下の記法を用意する．ニッコリを最長 σ 回連続させた賢者を $\mathcal{S}_\sigma$，$\mathcal{S}_\sigma$ のカードの値を s_σ，$\mathcal{S}_\sigma$ の左隣りの賢者 $\mathcal{L}_\sigma$ の数値を ℓ_σ とする．もちろん，列 $\mathcal{S}_\sigma = A, B, C, A, \ldots$，列 $\mathcal{L}_\sigma = B, C, A, B, \ldots$，と変化する．そして（大小関係の対称性を考慮して）

$$
\begin{aligned}
&p_\sigma \equiv \min_{\ell_\sigma}(\ell_\sigma - 1) = \min_{\ell_\sigma}(N - \ell_\sigma),\\
&q_\sigma \equiv \min_{s_\sigma, \ell_\sigma} |s_\sigma - \ell_\sigma|,\\
&r_\sigma \equiv \min_{s_\sigma}(s_\sigma - 1) = \min_{s_\sigma}(N - s_\sigma)
\end{aligned}
$$

と定義すると，

$$
\begin{aligned}
&p_1 = 1,\ q_1 = 2,\ r_1 = 1,\\
&p_2 = 3,\ q_2 = 3,\ r_2 = 2,\text{（フィボナッチ！）}\\
&p_3 = 6,\ q_3 = 5,\ r_3 = 5.
\end{aligned}
$$

一般に $\sigma \geq 3$ において

$$
\begin{aligned}
&p_\sigma = p_{\sigma-1} + q_{\sigma-1},\\
&q_\sigma = q_{\sigma-1} + q_{\sigma-2},\\
&r_\sigma = q_{\sigma-2} + p_{\sigma-1}
\end{aligned}
$$

という漸化式が成立する．$p_0 = 0, q_0 = q_{-1} = 1$ とおくと，この漸化式は $\sigma \geq 1$ でも成立する．ここでフィボナッチ数列 $\{F_n\}_{n \geq 1}$ を使うと（$\{F_n\}$ の和の公式を使うため，p_σ の最初の項に $F_1 - 1 = 0$ を p_0 として加えている），

$$
\begin{aligned}
&q_\sigma = F_{\sigma+2},\\
&p_\sigma = (F_1 - 1) + F_2 + \cdots + F_{\sigma+1} = F_{\sigma+3} - 2,\\
&r_\sigma = F_\sigma + F_{\sigma+2} - 2.
\end{aligned}
$$

ここまで分かると，「遊び」のない N_σ は

$$
\begin{aligned}
N_\sigma &= p_\sigma + q_\sigma + r_\sigma + 1\\
&= F_{\sigma+3} - 2 + F_{\sigma+2} + F_\sigma + F_{\sigma+2} - 2 + 1\\
&= F_{\sigma+3} + 2F_{\sigma+2} + F_\sigma - 3\\
&= 3F_{\sigma+2} + F_{\sigma+1} + F_\sigma - 3\\
&= 4F_{\sigma+2} - 3
\end{aligned}
$$

となる．

フィボナッチ数列が自然界のいろいろなところに登場することは有名だが，これは意外だった．そういえば，メモリ管理の buddy システムにフィボナッチ数列を使うという方法があったことを思い出す．

ちなみにカードが N_σ 以外の枚数のときは，上に述べた整数区間にたくさんの遊びが生じる．N_σ から $N_{\sigma+1}-1$ の間の N では最初から σ 回ニッコリ連続のあとに賢者が自分の勝利を宣言できる配牌の数は，順番に並べると，中間直前までは

$$G_n = 2 + \sum_{i=1}^{n-1} i^2 + 3\sum_{i=1}^{n-1} i + 2(n-1)$$

で求まる数列 $\{G_n\}_{n\geq 1}$ の先頭部分列となる．中間直後からは G_n の各要素を

$$H_m = 4 + 2\sum_{j=1}^{m-1} j^2 + 6\sum_{j=1}^{m-1} j + 4(m-1)$$

という数列 $\{H_m\}_{m\geq 1}$ の先頭部分列の要素で減らした値になる．H_m の添字 m は中間直後を 1 として始める．例えば，$N = 13$ のときは 70 通りではなく，$H_1 = 4$ を引いた 66 通りとなる．

H_m は最初からの最長ニッコリのあと，賢者が自分の負けを宣言する場合があることに由来する．

この式は実験で求めた．理論的に導くのは面倒そうである．

5. 問題 3 を解くプログラム

問題 1 も 2 も数理のマナイタに乗った．つまり，手計算できそうな気配がある．しかし，実際に賢者がカードを引いた状態で，このゲームを紙と鉛筆でシミュレーションするほうが難しい．

例えば，$N = 8$ のとき，$(a, b, c) = (4, 5, 1)$ と配牌されたとしよう．最初のターンで賢者 A が「C の負け」と宣言するところまでは簡単である．$(a, b) = (4, 5)$ なので，c がその間に入り得ないからである．そのあと，どうニッコリが続いて，どのターンで誰が A の勝利を宣言するのだろう？これが難しいことは実際に紙と鉛筆で計算してみれば体感できる．

私は自分で試みて降参してしまった．実は，いつまでも答えが出ないニッコリの無限ループになるのではないかとすら思ってしまった．いかん，これは問題が不完全なのではないかと．

しかし，傍目賢者の視点で賢者の思考をシミュレートしてみると，あっさりと答えが出た！ 正確に言うと，配牌の可能性をすべて掌握している傍目賢者が，3 人の賢者をシミュレートするのである．3 人の配牌が見えている「神の視線」ではなく，3 人の配牌の可能性を掌握している傍目賢者というところがポイントである．

賢者の発言によって，配牌の可能性を減らしていくのは純粋に客観的な作業である．このプロセスを手計算で追うのは大変である．

問題 1 も問題 3 も *All-hands* を使って，発言の可能性を調べるのは同じなので，ここでは問題 3 のプログラムをベースにプログラムの構造を少し詳しく述べる．Lisp (TAO) で書いて，コメント行を除いて 90 行程度のプログラムである．

主たる関数は `sage-tell` で，以下の構造をしている．

(1) 初期化
　All-hands を作る（辞書式降順）.
　ターンのカウンタなどを準備する.
　賢者と配牌のリストを用意する（先頭の賢者がいま発言しようとしている賢者）.

(2) 賢者の発言のループ
　勝者が宣言されるまで，賢者と配牌のリストを左方向に循環シフトしながら繰り返す．1 回の繰り返しは次のように進む.

(2-1) 宣言できるかもしれない配牌･宣言内容等をターンごとに溜めるリスト *Could-tell* を初期化する（空にする）.

(2-2) *All-hands* のすべての配牌を，同じ *mine, left* を持つものでグループ化して勝敗がつくかどうか調べる．調べ方は第 3 章で述べたとおり．勝敗がつくなら，*Could-tell* にそれらの配牌を勝敗情報とともに追加し，*All-hands* から取り除いていく．なお，このとき同じグループで勝者と敗者両方について宣言可能なら，敗者については無視する.

(2-3) *Could-tell* に登録されたもののうち，実際に宣言できるのは，勝者確定の場合は，配牌の

mine, *left* が現在の賢者の *mine*, *left* と一致するもののみである．そこでゲームは終了する．敗者確定の場合も，同様の一致があり，すでに敗者が宣言済みでないものである．これらの宣言ができないときはニッコリする．

配牌の *mine*, *left* の一致に関しては，**傍目賢者ではなく**，3 人の賢者それぞれの視線で見ていることに注意．

問題 1 の (2-3) は，*All-hands* が空になったときに，*Could-tell* のうち，そのターンの賢者が自分の勝利を宣言できるものだけを抜き出す処理となる．次の (2-4) はない．

(2-4) 敗者が確定すると「敗者確定モード」に入り，変数 *loser* の値が (敗者 `lost`) になる．次のターンの *All-hands* は *Could-tell* に残された要素のうち (敗者 `lost`) を含むものから再構成する．つまり，*All-hands* の値をすべて捨て，*Could-tell* から，傍目賢者，つまり，すべての競技賢者に共通の事実となっている（敗者の負けを決定づけた）配牌候補を復活させる．

このとき復活させる配牌には，そのターンを終えた賢者にとって無駄なものを含んでいるが，その賢者から見た無駄であって，傍目賢者には削除が不可能である．

なお，ゲーム中に敗者が確定するのは 1 回だけである．もう 1 回敗者が確定するということは，勝者が確定してゲームが終了することにほかならないからである．

(2-5) 小さくなった *All-hands* の中の配牌を辞書式降順にソートし，次の賢者のターンに移る準備をして繰り返しに戻る．

付録に $N = 8$, $(a, b, c) = (4, 5, 1)$ のときの *All-hands* の変化の様子を示す．例えば，最初の A による敗者確定宣言のあと，B に与えられた *All-hands* には無駄な配牌が多いように見える．しかし，配牌が見えない傍目賢者にとっては，この時点で，これがすべての配牌の可能性である．つまり，のっけから C が負けたと A と言ったことは「a と b がそれぞれ 1 でも 8 でもない，連続した数」という事実だけを公知にする．

ここで，配牌が $(4, 5, 1)$ であったことを忘れてみよう．A が「C の負け」と言った次のターンで B が「自分の勝ち」と言えば，付録から，傍目賢者には 10 通りの配牌の可能性があることしか分からないが，3 人の賢者全員は配牌のうち 2 枚を知っているので（この場合は）B が勝ったユニークな配牌を知ることができる．

問題 1 のプログラムに対する変更は，変数の宣言やそれに伴う初期化のほかは，(2) のループの中の (2-3) の変更と (2-4) の追加である．もっとも，(2-4) での，一度消したものの「復活」は計算として無駄である．しかし，追加モジュールとして完結しているほうが嬉しい．

それにしても，最初に C の負けが宣言できたあとのニッコリの連続は，最初からニッコリが連続する場合よりかなり多い．つまり最初から連続するニッコリは実は大きな情報をもたらすのである．裏返すと，敗者が決まったあとは，一発で大きな情報は得られない．こつこつと範囲を狭めていくしかない．

最初から 2 回連続のニッコリで勝者宣言をし得る $N = 16$ なのに，具体的に $(a, b, c) = (8, 9, 1)$ という配牌を与えると，

0 A declares C lost; `(8 9 1)`
　　⋯ 途中省略 ⋯
0 A declares C lost; `(8 9 16)`
1 B smiles
2 C smiles
3 A smiles
4 B smiles
5 C smiles
6 A smiles
7 B smiles
8 C smiles
9 A smiles
10 B declares A won; `(9 1 8)`

と，B から 9 回ニッコリが連続という経過をたどる．賢者の宣言に書かれているリストは宣言した賢者の視点で根拠とした配牌候補である．

なお，配牌が $(a, b, c) = (1, 2, 7)$ のように最初から A が B の勝利を宣言できる場合など，勝利宣言でもこの配牌候補が複数表示され得る．その時点

では A にとって，c の範囲を絞れないからである．

6. むすび

田中哲朗氏の SMT ソルバしかり，「**傍目賢者プログラミング**」しかり，プログラミング方法論を変えるだけで圧倒的に問題がやさしくなるという事例は少なくない．現実の忠実なモデル化としてのオブジェクト指向プログラミングを使うのは，場合によりけりということだろうか．

また，この報告では，実際のゲーム進行をシミュレーションする問題 3 を提唱してプログラムを書いた．こちらのほうは「メタ問題」ではないので，オブジェクト指向で書いたほうがいいのかもしれないが，「傍目賢者プログラミング」をそのまま援用して書いた．計算過程に無駄な出し入れがあるが，プログラムセグメントの変更・追加のみで簡単に書けた．

傍目賢者は，個々の賢者より個別の情報が少ない分，客観的でスムーズな処理ができる．しかし，勝者や敗者の宣言の予測はできないし，宣言後の配牌推定もユニークにならない．

逆に，傍目賢者の推論は，3 人の賢者には完全な形で実行できることに注意しておこう．**傍目マイナス八目**と言ってよい．傍目賢者をシミュレートするオブジェクトとして賢者を書き，(個々の賢者目線で循環シフトやソートするなど，モデルとしてやや不自然な気もするが) *All-hands* をオブジェクトの外側において，*Could-tell* と (*mine,left*) を処理するところだけオブジェクト内部で行うという方法もありだろう．

第 57 回のプログラミングシンポジウムの予稿に書いたが，「3 人の賢者の問題」は Freudenthal の「不可能問題」[7] を頂点とする「分からないことが情報になるパズル」に属する．「3 人の賢者の問題」は，額に泥のついた子どもの問題のような論理だけの問題ではなく，数値の演算が含まれるので，より複雑な範疇に属する．それにしてもフィボナッチ数列が出てきたのには驚いた．

2015 年夏のプログラミングシンポジウムの報告集に，私は「二と三」と題した雑文を書いた [8]．そこではその後「下呂数」と呼ばれた，やはり 2 と 3 の組合せに基づく問題を急造した．このほかにも，3 引数で 2 重再帰の竹内関数など，2 と 3 が絡むとなにか怪しい面白い問題が作れる，あるいは興味深い問題を探ると奥底に 2 と 3 が絡んでいることがよくあるといった随筆だった．「3 人の賢者の問題」は，3 人が 2 人の情報を循環的に知っているという意味で「2 と 3 の問題」のジャンルに属することは間違いない．

[†] 勘違いで，数列 H_m による補正を見逃してしまった．

[参考文献]

[1] 竹内郁雄: エレガントな解答をもとむ，数学セミナー，2014 年 7 月号，10 月号，日本評論社．

[2] Douglas R. Hofstadter: Metamagical Themas, Basic Books, 1985. (邦訳) → メタマジック・ゲーム，竹内郁雄，斉藤康己，片桐泰弘，白揚社，2005（新版）．

[3] 竹内郁雄: 3 人の賢者の問題 —— 愚者は賢者をシミュレートできるか?，第 57 回プログラミングシンポジウム報告集，情報処理学会，2016 年 1 月．

[4] 田中哲朗: SMT ソルバを用いた「竹内の 3 人の賢者問題」の求解．組合せゲーム・パズルプロジェクト 第 11 回研究集会，2016 年 3 月．スライドは，http://www.alg.cei.uec.ac.jp/itohiro/Games/160307/160307-16.pdf .

[5] 副田俊介，田中哲朗: 最中限の終盤の分析，情報処理学会研究報告ゲーム情報学（GI）2003, pp.31-38, 2003-08-04 情報処理学会．

[6] 竹内郁雄: エレガントな解答をもとむ，数学セミナー，2019 年 11 月号，日本評論社 †．

[7] Axel Born, Cor A.J. Hurkens, Gerhard J. Woeginger: The Freudenthal Problem and its Ramifications (Part 1), Bulletin of the EATCS, no. 90, pp. 175-191, 2006, European Association for Theoretical Computer Science.

[8] 竹内郁雄: 二と三，2015 年夏のプログラミングシンポジウム報告集，情報処理学会，2016 年 1 月．

[付録] (sage-tell 8 4 5 1) のトレース

最初に A が C の負けを宣言，敗者確定モードに入る．次に B のターン．C 以外の勝ちとなっても（B の）*mine* と *left* が配牌にマッチしないかぎり棄却（# で表す）．+ の記号は同じ *mine*, *left* の配牌グループを意味する．配牌の第 3 項 *right* が前後していて勝者が決まらない？ のグループは次の *All-hands* に残す．ターンが変わるたびに，配牌が左循環シフトされることに注意.

- B のターン，*All-hands* には 60 個の配牌．ニッコリ.

```
(7 8 6) B won#     (7 5 6) A won#     (7 4 6) A won#     (7 3 6) A won#
(7 2 6) A won#     (7 1 6) A won#
(6 8 7)+(6 8 5)?
(6 7 5) B won#     (6 5 7) B won#
(6 4 7)+(6 4 5)?   (6 3 7)+(6 3 5)?   (6 2 7)+(6 2 5)?
(6 1 7)+(6 1 5)?   (5 8 6)+(5 8 4)?   (5 7 6)+(5 7 4)?
(5 6 4) B won#     (5 4 6) B won#
(5 3 6)+(5 3 4)?   (5 2 6)+(5 2 4)?   (5 1 6)+(5 1 4)?
(4 8 5)+(4 8 3)?   (4 7 5)+(4 7 3)?   (4 6 5)+(4 6 3)?
(4 5 3) B won#     (4 3 5) B won#
(4 2 5)+(4 2 3)?   (4 1 5)+(4 1 3)?   (3 8 4)+(3 8 2)?
(3 7 4)+(3 7 2)?   (3 6 4)+(3 6 2)?   (3 5 4)+(3 5 2)?
(3 4 2) B won#     (3 2 4) B won#
(3 1 4)+(3 1 2)?
(2 8 3) A won#     (2 7 3) A won#     (2 6 3) A won#     (2 5 3) A won#
(2 4 3) A won#     (2 1 3) B won#
```

- C のターン，*All-hands* には 40 個の配牌．ニッコリ.

```
(8 7 6) A won#     (8 6 5) A won#
(8 5 6)+(8 5 4)?   (8 4 5)+(8 4 3)?
(8 3 4) B won#     (8 2 3) B won#     (7 6 5) A won#     (7 5 4) A won#
(7 4 5)+(7 4 3)?
(7 3 4) B won#     (7 2 3) B won#     (6 5 4) A won#     (6 4 3) A won#
(6 3 4) B won#     (6 2 3) B won#     (5 4 3) A won#     (5 2 3) B won#
(4 7 6) B won#     (4 5 6) A won#     (3 7 6) B won#     (3 6 5) B won#
(3 5 6) A won#     (3 4 5) A won#     (2 7 6) B won#     (2 6 5) B won#
(2 5 6)+(2 5 4)?
(2 4 5) A won#     (2 3 4) A won#     (1 7 6) B won#     (1 6 5) B won#
(1 5 6)+(1 5 4)?   (1 4 5)+(1 4 3)?
(1 3 4) A won#     (1 2 3) A won#
```

- A のターン，*All-hands* には 12 個の配牌．以下のグループは 1 個も捨てられない．ニッコリ.

```
(5 6 8)+(5 6 2)+(5 6 1)?   (5 4 8)+(5 4 2)+(5 4 1)?
(4 5 8)+(4 5 7)+(4 5 1)?   (4 3 8)+(4 3 7)+(4 3 1)?
```

- B のターン，12 個の配牌が *All-hands* に残っている．1 個だけ B の *mine*, *left* と一致する勝ちがある．ここで B は A の勝利を宣言し，ゲームは終了する．傍目賢者は 6 通りまでしか配牌を絞りきれない.

```
(6 8 5) B won#     (6 2 5) A won#     (6 1 5) A won#     (5 8 4) B won#
(5 7 4) B won#
(5 1 4) A won **** Game End! ****
(4 8 5) A won#     (4 2 5) B won#     (4 1 5) B won#     (3 8 4) A won#
(3 7 4) A won#     (3 1 4) B won#
```