バイナリ生物学入門

— バイナリを生物のように実行する
モデルと PoC 実装 —

［著］大神祐真

コミックマーケット 97 新刊
2023 年 5 月 21 日　ver 1.1

■免責

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、®、© などのマークは省略しています。

はじめに

本書をお手にとっていただきありがとうございます！ 本書は「バイナリ生物学」と筆者が勝手に呼んでいる学問 (?) を紹介する本です。

(大分、与太話なところはありますが、同人誌ということで面白がってもらえれば嬉しいです。)

バイナリ生物学とは何ぞや

完全に筆者の造語ですが、実行バイナリや画像ファイル等のバイナリデータを、まるで生き物のように扱ってみる、という学問 (?) です。

普段私達は、実行バイナリを生成する際はソースコードを書いてコンパイルしたり、画像を生成する際はペイントソフトで絵を描いたり、といったことを行います。ただ、これらは結局は何らかのバイナリが並んだデータに過ぎないと考えると、何らかのバイナリとバイナリを結合して作っても良いし、できるのであればバイナリエディタ等でバイナリを手で書いても生成することはできます。

そこで、バイナリ生物学ではバイナリデータの生成過程や実行過程などを生物のように扱うことで、これまでの OS などとは全く違った新しいシステムが実現できたり、そこまではいかなくとも、何か面白いことができないか、ということを考え、設計し、実装し、評価します。

それによる理想としては、バイナリが生物的な振る舞いを獲得することによる「自己進化」・「自己再生」・「自己増殖」により、エンジニアの能力を超えた機能を提供したり (進化)、バグやウイルスによる実行バイナリの問題を自己解決したり (再生)、必要に応じて自律的に並列度を増やす (増殖)、といったバイナリが生まれること、また、そのようなバイナリを処理できる処理系を提供できること、です。

なお、本書ではそのような究極のバイナリを実現するところまでは行きません。本書では実行バイナリを対象に「生きている」と言い張るための設計と、実装したサンプルプログラムの評価結果を紹介します。

本書の構成

本書は以下の 3 章構成です。

- 第 1 章 「生きている」と言い張る戦略
  - バイナリが生きていると言い張るための戦略と、そのために参考にする生

物の構造を説明します
- 第 2 章 設計とモデル
  - 生物とするための構造と生きていると言い張るための振る舞いをプログラムで表現するための設計とモデルを説明します
- 第 3 章 PoC 実装の実行結果
  - PoC として実装したものの結果を紹介します

本書の PDF 版/HTML 版やサンプルプログラムについて

これまでの当サークルの同人誌同様、本書も PDF 版/HTML 版は以下のページで無料で公開しています。また、本書で誤記や更新等があった際もこちらで公開します。

- `http://yuma.ohgami.jp`

本書で紹介する設計の実装例であるサンプルプログラムは以下の GitHub リポジトリで公開しています。

- `https://github.com/cupnes/intelligent_life_in_the_binary_samples`

実装については本書でも扱っていますが、実行方法など詳しくは上記リポジトリの Readme をご覧ください。

目次

第 1 章　「生きている」と言い張る戦略

この章では、実行バイナリを「生きている」と言い張るための戦略を紹介します。また、そのために作成するシステムの構成要素を一通り紹介します。

1.1　「生きている」と言い張る後ろ盾

まず、「何を以て『生きている』と言い張るか」ですが、その後ろ盾は広辞苑で得ることにします。『広辞苑 (第 7 版)』には「生物」について以下のように書かれています。

> せい-ぶつ【生物】[礼記楽記] 生きもの。生活しているもの。一般に栄養代謝・運動・成長・増殖など、いわゆる生活現象をあらわすものとされるが、今日では増殖を最も基本的・普遍的属性とみなす。↔無生物

この記述より、「代謝」・「運動」・「成長」・「増殖」の 4 つの振る舞いを「生活現象」と呼んでいて、「生活現象」を行えば「生物」と見なして良さそうです。

そこで、本書ではこれに「死」を加えた以下の 5 つの振る舞いを実装することで、「生きている」と言い張ることにします。

- 代謝
- 運動
- 成長
- 増殖
- 死

以降では、このような「生きたバイナリ」を「バイナリ生物」や「バイナリ生命体」と呼称することにします。

1.2　バイナリ生物の構造を考える

5 つの振る舞いを考える前に、その振る舞いを成すバイナリ生物の構造を考えます。

♣ 単細胞生物の構造から必要な要素をピックアップ

バイナリ生物の構造を考えるに当たり、本書では「単細胞生物[*1]」を参考にします。単細胞生物といえど、その細胞の中には色々な構成要素がありますが、本書では「タンパク質」と「核様体 (DNA)」、そして目に見えない要素ではありますが、属性情報として「寿命」、の 3 つをバイナリ生物の細胞の要素とすることにします。

以降、この節では、細胞を構成する要素について、バイナリ生物を作る上で参考にするポイントのみ簡単に説明します。ただ、筆者は生物学の専門家ではないため、書いている説明は厳密には間違っているかもしれません。ここでの目的はあくまでも「バイナリ生物の設計」で、それをある程度信憑性のあるものにするために現実の単細胞生物を参考にしている、という位置づけのため、生物学的な単細胞生物の定義を細かくトレースすることにはこだわらないことにします。

♣ タンパク質とは

タンパク質は細胞内の液体部分 (細胞質) に多数存在し、代謝を行ったり、運動のエネルギーを生み出すなど、機能的な役割を果たします[*2]。特に、ある物質を別の物質へ変質させる媒体となるようなタンパク質は「酵素タンパク質」と呼ばれます。酵素タンパク質を用いたこの様な物質変化が細胞における「代謝」であり、またそれにより「運動」のためのエネルギーを生み出したりします。

ここでタンパク質の構成を簡単に確認します。まず、タンパク質は、アミノ酸という「化合物」で構成されています。そして「化合物」とは「元素」が 2 つ以上結合してできた物質を指します。

細胞を含めてまとめると以下の関係です。

- 「細胞」は、複数の「タンパク質」を複数持ち、それを使って何らかの機能を果たす
  - 「タンパク質」は、複数の「化合物」でできている
    - 「化合物」は、複数の「元素」でできている

[*1] 特に DNA などの遺伝物質が核膜で覆われていない原核細胞でできている原核生物を参考にしています。

[*2] その他にも細胞の各器官を形作る構成物になるなど、ハードウェア的な役割を果たすタンパク質もあり、タンパク質で何でもやります。

♣ DNA とは

DNA は、「タンパク質の設計図」と呼ばれるもので、細胞を構成するタンパク質を生み出す役割があります。細胞分裂により「増殖」を行う上で欠かせない機能です。

「ヌクレオチド」という物質で構成されており、ヌクレオチド 3 つ分 (これを「コドン」と呼びます) が一つのアミノ酸に対応することで「タンパク質の設計図」としての機能を果たします。また、これにより DNA からタンパク質を生み出す過程が「セントラルドグマ」と呼ばれるものです。

なお、細胞における「成長」は化合物を蓄えていくことに当たります。本書で扱う単純化された細胞では、細胞分裂のために化合物を集めることが「成長」になります。

♣ 属性情報 (寿命)

物理的に目に見えるものではないですが、単細胞生物も生物である以上、いつかは死ぬため、「寿命」があります[*3]。

バイナリ生物を作成する際は、属性情報として寿命も細胞の構成要素として含めることにします。

1.3 処理系

処理系は、言わば「地球環境」のようなものです。本書では、生物を周期的に実行して生物としての振る舞いをさせたり、化合物などのリソースを管理する存在と考えます。

詳しくは次の章で説明します。

1.4 まとめ

本書では、実行バイナリに生物としての「5 つの振る舞い」をさせることで「生きている」とします。

「5 つの振る舞い」を実現する構造として「細胞」を用います。本書で扱う細胞の構成要素は単純化して「タンパク質」・「DNA」・「属性情報 (寿命)」の 3 つです。

そして、それらの構造を解釈し、周期的に振る舞いを呼び出す存在として「処理系」を用意します。

[*3] 特に、多細胞生物で細胞が組織を作るような場合は、組織によって細胞の寿命が予め決められていて、計画的に死ぬ (アポトーシス) ようにできていたりします。

第 2 章　設計とモデル

前章で説明した「5 つの振る舞い」と「細胞の構造」を設計し UML でモデル化します。

そして、本書で PoC[*1]実装するバイナリ生物とその処理系がその中でどの様に実現されるのかを説明します。

2.1　PoC 実装するバイナリ生物について

本書では、「インクリメンタ細胞」という細胞一つを体の本体とする「インクリメンタ生物」という単細胞生物を作ります。

インクリメンタ生物は以下のようにして生きている生物であるとします。(「細胞」や「化合物」等をどのようにモデル化するかは後述しますので、ここでは「そのようなものか」と思っておいてください。)

- 化合物 (64 ビット整数値) を取り込み、細胞内のタンパク質と結合して代謝を行うことで生きている
- 代謝により化合物はインクリメントされた値へ変質する
- インクリメント後の化合物は体外へ排出する

それでは、次節から、前章で説明した構造と振る舞いの設計とモデル化を説明します。

2.2　細胞の構造を掘り下げる

前章で紹介した「タンパク質」・「DNA」・「属性情報 (寿命)」について、バイナリの

[*1] 「Proof of Concept(概念実証)」の略。本書でサンプルとして紹介する実装は「生きている」と言い張る事を満たしてはいますが、それ以上のものではありません。

世界にどのように対応付けるのかを掘り下げて説明します。

そして、この節の最後に、細胞の各構成要素の機能とそれぞれの関係をクラス図で示します。

♣ タンパク質: バイナリの世界との対応付け

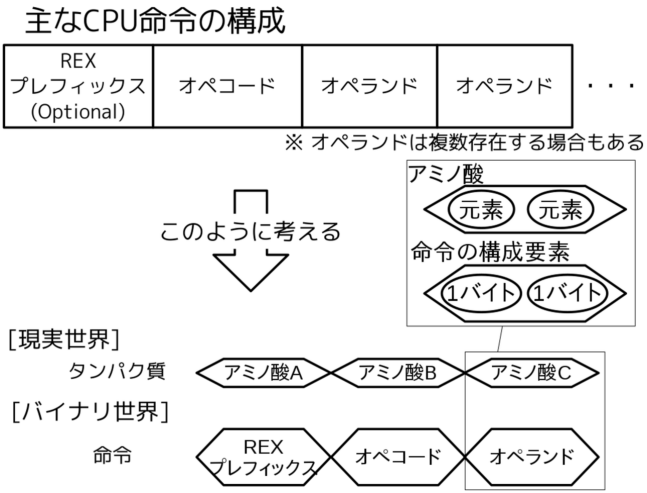
前章で、細胞とその構成要素であるタンパク質の構成を以下のように説明しました。

- 「細胞」は、複数の「タンパク質」で機能を果たす
  - 「タンパク質」は、複数の「化合物」でできている
    - 「化合物」は、複数の「元素」でできている

これを、バイナリの世界における「関数」の以下の構成と対応付けて考えます。

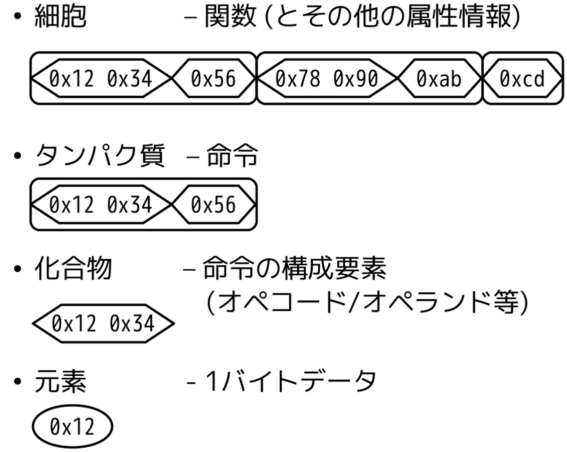
- 「関数」は、複数の「機械語命令」で機能を果たす
  - 「命令」は、1 つあるいは複数の「命令の構成要素」でできている
    - 「命令の構成要素」は、1 つあるいは複数の「1 バイトデータ」でできている

「命令の構成要素」とは、主に「オペコード」・「オペランド」等と呼ばれる 1 つの命令列を構成する要素です。タンパク質と命令を図 2.1 の様に置き換えて考えてみるわけです。



▲図 2.1: CPU 命令とタンパク質の関係

以上を踏まえて、現実世界とバイナリ世界の対応をまとめると図 2.2 の通りです。



▲図 2.2: 現実世界とバイナリ世界の対応まとめ

すなわち、バイナリ世界における「単細胞生物」は、「関数 1 つだけの実行バイナリ」となります。

そして、細胞の「タンパク質が触媒となり物質を別の物質へ変質させる (代謝)」に対して、バイナリ世界でも「関数実行により引数が戻り値へ変質している」と考え、「関数実行」を「代謝」と対応付けます。また、細胞は「代謝の際に得られたエネルギーを使って何らかの動作を行う (運動)」のですが、これも「関数実行により何らかの処理が実施される」と対応付けることで、「運動」も「関数実行」に含まれると考えます。

また、ここで、「化合物」には「命令の構成要素になるもの」と「関数の引数になるもの」の 2 種類が存在することになります。本書では、命令の構成要素になる化合物を「コード化合物」、関数の引数になる化合物を「データ化合物」と呼ぶことにします。

♣ タンパク質: インクリメンタ細胞を表現する

以上を踏まえ、インクリメンタ細胞ではどの様に表現するのかを考えます。

まず、インクリメンタ細胞は、「引数 (64 ビット整数値) をインクリメントして返す関数 (インクリメント関数)」だけで構成された実行バイナリとします。

インクリメント関数は C 言語で書くとリスト 2.1 のように書けます。

▼リスト 2.1: インクリメント関数 (C 言語)

```
unsigned long long incrementer(unsigned long long num)
{
        return num + 1;
}
```

これを GCC 等でコンパイルすると 7 命令程の機械語バイナリになるのですが、「インクリメントして返す」だけであれば 3 命令で実現できます (リスト 2.2)。

▼リスト 2.2: インクリメント関数 (アセンブラ)

```
mov     %rdi, %rax
inc     %rax
ret
```

簡単に説明すると、C 言語の関数の第 1 引数は、コンパイル後の機械語バイナリ (およびアセンブラ) では、RDI というレジスタで扱われます。そのため、RDI レジスタの内容を確認すれば、呼び出し元が第 1 引数で渡してきた値を確認できます。

ここでは、最初の「mov」命令で RDI レジスタの中身を RAX というレジスタへコピーします。

次に、「inc」命令で RAX レジスタの中身をインクリメントします。

最後に、「ret」命令で呼び出し元へジャンプします。その際、RAX レジスタが戻り値となるので、引数で渡した値がインクリメントされて返されることになります。

これはアセンブラでの表現なので、これを CPU が直接解釈する機械語に置き換える (アセンブル) とリスト 2.3 の通りです。これが機械語バイナリ (の実行コード部分) となります。

▼リスト 2.3: インクリメント関数 (機械語バイナリ)

```
0x48 0x89 0xf8
0x48 0xff 0xc0
0xc3
```

リスト 2.3 の各命令を、命令の構成要素で分解し、細胞の構造と対応付けるとリスト 2.4 の通りです。

▼リスト 2.4: インクリメント関数の各命令を分解する

```
mov命令(タンパク質):
- REXプレフィックス(化合物): 0x48
- オペコード(化合物)       : 0x89
- オペランド(化合物)       : 0xf8

inc命令(タンパク質):
```

```
- REXプレフィックス(化合物): 0x48
- オペコード(化合物)        : 0xff
- オペランド(化合物)        : 0xc0

ret命令(タンパク質):
- オペコード(化合物)        : 0xc3
```

「REX プレフィックス」は「64 ビット命令」を示す 1 バイトです。例えば、「`0x48 0x89 0xf8`」を「`0x89 0xf8`」と書いた場合、「`mov %edi,%eax`」という 32 ビットの命令になります[*2]。

以上から、本書で PoC 実装する世界には 6 種のコード化合物と、特に種別の無い 8 バイトのデータ化合物が存在することになります。

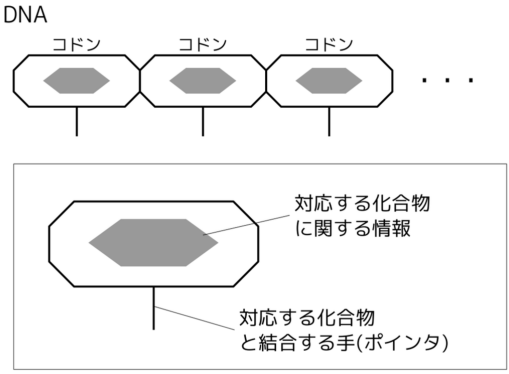
♣ DNA: バイナリの世界でどのように表現するか

本書では、DNA はコドンが連結したリストとして扱います。本書で扱うバイナリ生物の世界観では DNA の構造の最小単位をコドンとし、ヌクレオチドに相当する粒度は扱いません[*3]。

タンパク質の説明の際は、現実世界と対比させて説明しましたが、プログラムのバイナリ世界の DNA は現実世界と結構変えてしまったので、モデル化した結果のみ説明します。バイナリ世界の DNA を図示すると図 2.3 の通りです。

[*2] 詳しい説明は省きますが、先頭に"R"が付く「RDI」等のレジスタは 64 ビットのレジスタで、先頭に"E"が付く「EDI」等のレジスタは 32 ビットのレジスタです。この例では、REX プレフィックスを付けることで mov 対象のレジスタのビット幅が変わっています。

[*3] 本書では実装の簡単さ等も考えてこのようにしましたが、DNA のモデル化はもう少し良い方法があるのかもしれません。

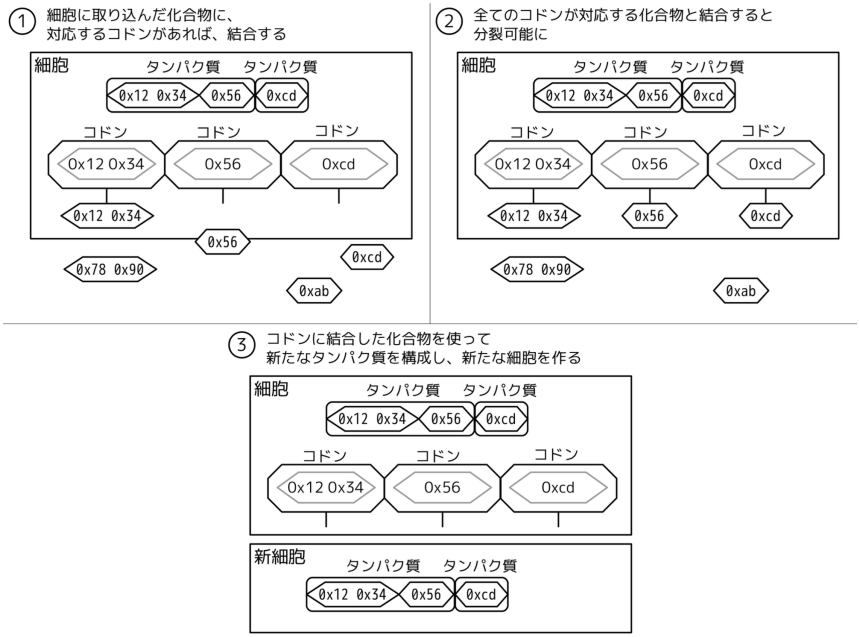


▲図 2.3: バイナリ世界の DNA

各コドンは対応するコード化合物に関する情報を持ちます。コード化合物、すなわち命令を構成するオペコード/オペランド等の要素は、それぞれの長くとも 64 ビットを越えることは無いので、コドンは対応するコード化合物を 64 ビット整数値で保持することができます。

そして、コドンに対応するコード化合物を取り込んだときに、それと結合しておくための手として化合物へのポインタも持ちます。

このポインタを使用して図 2.4 のようにセントラルドグマを行い、細胞を構成するタンパク質を生成します。



▲図 2.4: バイナリ世界のセントラルドグマ

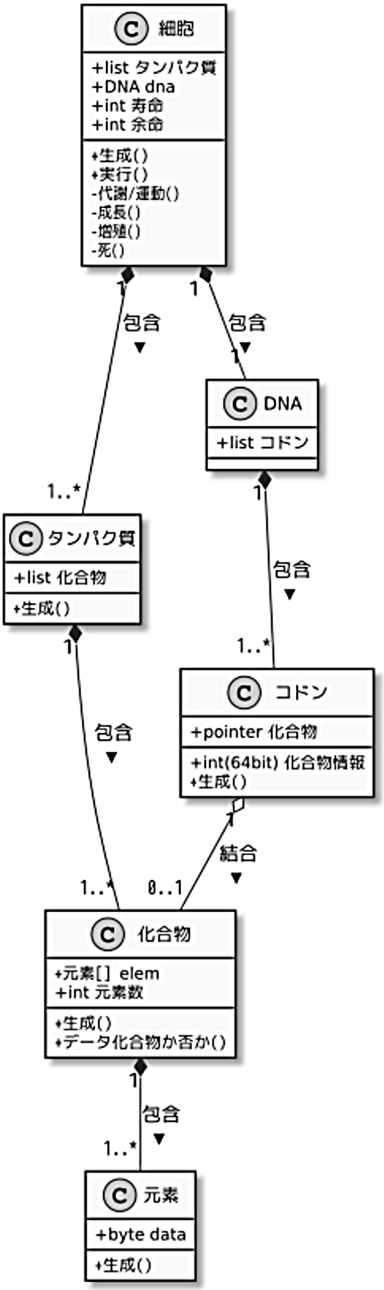
♣ 属性情報 (寿命): バイナリの世界でどのように表現するか

属性情報として寿命は、細胞のデータ構造の中に整数値として持たせておき、周期毎にデクリメントすることにします。

そして、0 になると、「死」の振る舞いを実施することにします。(振る舞いの内容は後述します。)

♣ バイナリ生物の細胞の構造まとめ

これまで説明してきたことを、それぞれの構造の関係を示すために UML のクラス図で図示してみると図 2.5 の様になります。なお、各構造が持つ関数は、各構造の役割として必須なものを挙げているだけで、これが全てというわけではありません。(特にそれぞれの構造で内部的に使う関数などは、実装の仕方により増えることもあると思います。)



▲図 2.5: 細胞から元素までの構造

以上、大分ざっくりとした説明ですが、複雑にモデル化すればするほど、それを実装する必要があるので、ひとまず簡単な単細胞生物をバイナリ生命体として実現する上では以上のモデルで十分です。

2.3 処理系

ここまで、実行バイナリを生物として対応付けて考える事を説明してきました。ただし、そのためには、実行バイナリを先述の構造に当てはめる形でロードし、後述する生物としての 5 つの振る舞いを行わせる「処理系」が必要です。

そのために、「環境に存在する細胞と化合物を管理する」、「細胞の要求に応じて環境に存在する化合物を与える/あるいは環境へ配置する」、「周期を管理し各周期で環境に存在する全ての細胞の 1 周期を実施する」などが必要です。

なお、実行バイナリをロードする部分は本書の PoC では扱いません。PoC 実装では、処理系の初期化時に実行バイナリをロード済みの状態で細胞やタンパク質の初期状態をセットアップします。

振る舞いのモデルは次節で紹介します。

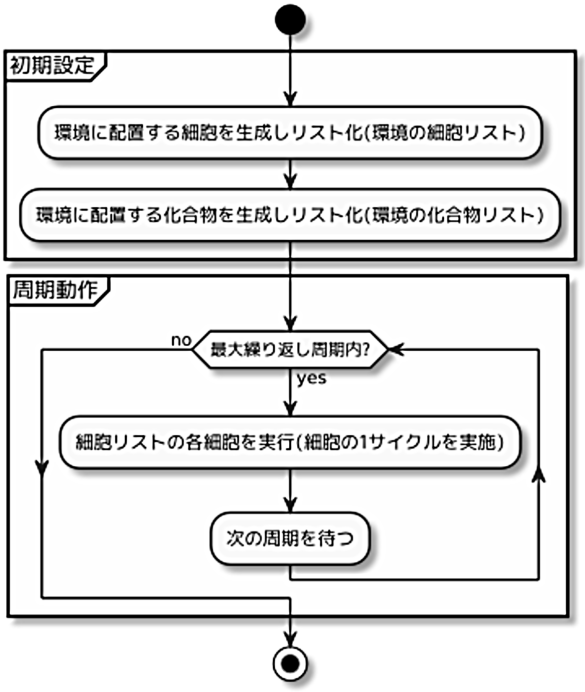
2.4 処理系と 5 つの振る舞いを考える

前節までで設計した構造を元に、バイナリ生物における細胞とその処理系の振る舞いを考えます。

♣ 処理系

前述した構造で用意された細胞を処理する処理系としては図 2.6 のような振る舞いをするものを考えます[*4]。

[*4] なお、PoC 実装の対応するソースコードの箇所はリポジトリ直下 main.c の main 関数です。適宜参照してみてください。



▲図 2.6: 処理系の振る舞い

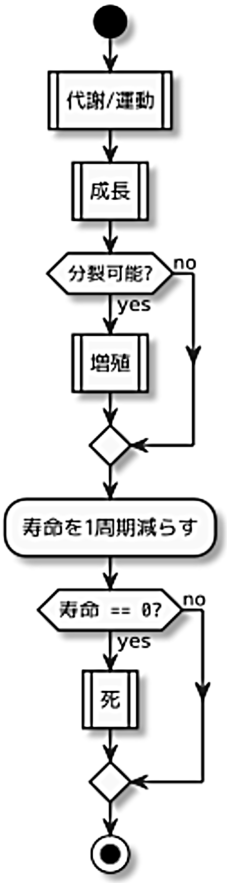
前述した通り、今回の PoC サンプルでは実行バイナリをロードする部分は扱いません。処理系の初期設定時に細胞の構造を構築します。

初期設定を終えた後は、細胞のリストに存在する細胞を毎周期で全て実行します。

♣ 細胞の 1 周期

次に、細胞の 1 周期です。1 周期の中で、「代謝/運動」・「成長」・「増殖」・「死」の振る舞いが適宜呼び出されます (図 2.7)[*5]。

[*5] PoC 実装の対応箇所はリポジトリ直下 cell.c の cell_run 関数です。



▲図 2.7: 細胞の 1 周期の振る舞い

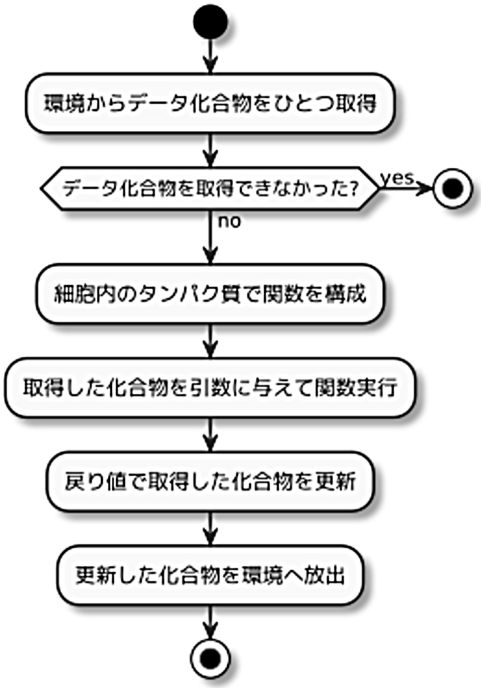
毎周期で「代謝/運動」と「成長」を実施し、成長後、分裂可能であれば「増殖」を行います。寿命も毎周期で減っていき、0 になると「死」の処理を実施します。

以降で、「代謝/運動」・「成長」・「増殖」・「死」それぞれの振る舞いを紹介します。

♣ 代謝/運動

「代謝/運動」の振る舞いを図示すると図 2.8 の通りです[*6]。

[*6] また、PoC 実装の対応箇所はリポジトリ直下 cell.c の metabolism_and_motion 関数です。



▲図 2.8: 代謝/運動の振る舞い

まず、環境から関数の引数となるデータ化合物をひとつ取得します。もし取得できなければこの周期での代謝/運動は何もしません。

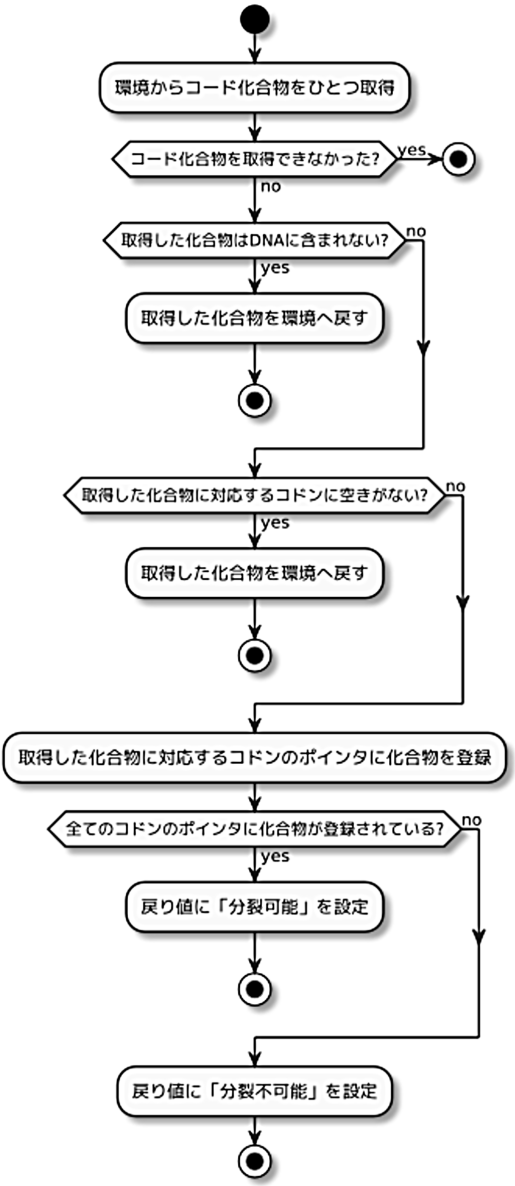
データ化合物を取得できた場合、細胞内の各タンパク質を辿って化合物を取得し、メモリ上で各化合物の内容を結合して関数を作ります。そして、取得したデータ化合物の内容を引数として関数を実行します。前述した通り、この関数実行が代謝と運動に当たります。

最後に、関数の戻り値でデータ化合物を更新し、環境へ戻します。

♣ 成長

関数を構成するオペコード/オペランド等のバイト列 (化合物) を集める作業を「成長」と考えます。「成長」の振る舞いを図示すると図 2.9 の通りです[*7]。

[*7] PoC 実装の対応箇所はリポジトリ直下 cell.c の growth 関数です。


環境からコード化合物をひとつ取得
コード化合物を取得できなかった?
yes
no
取得した化合物はDNAに含まれない?
no
yes
取得した化合物を環境へ戻す
取得した化合物に対応するコドンに空きがない?
no
yes
取得した化合物を環境へ戻す
取得した化合物に対応するコドンのポインタに化合物を登録
全てのコドンのポインタに化合物が登録されている?
no
yes
戻り値に「分裂可能」を設定
戻り値に「分裂不可能」を設定

▲図 2.9: 成長の振る舞い

まず、環境からコード化合物を取得します。取得できなかったり、取得したが DNA に含まれていなかったり、DNA に含まれているが既に取得済みである場合は、適宜取得した化合物を環境へ戻した後、何もせずこの周期の成長を終えます。

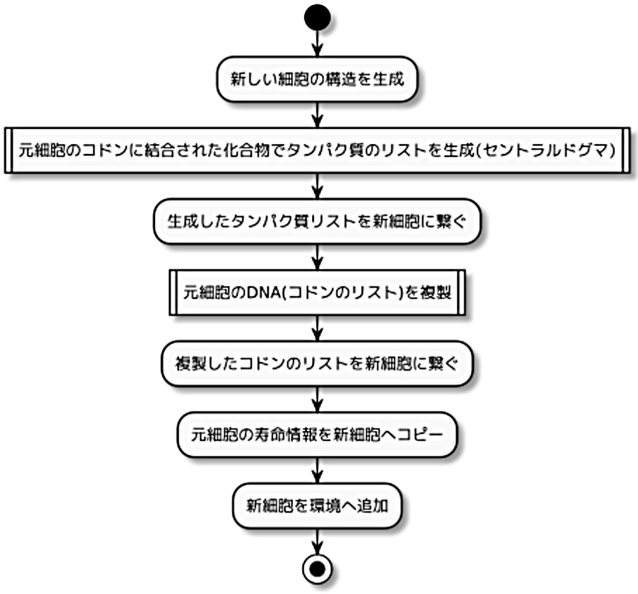
取得した化合物が DNA に含まれ、それをまだ取得していなかった場合、DNA 内の対応するコドンが持つポインタに取得したコード化合物を設定します。

その後、DNA 内の全てのコドンに化合物が設定済みか否かをチェックし、設定済みである場合は分裂可能を示す値を、未設定のコドンが一つでもあれば分裂不可能を示す値を返します。

♣ 増殖

細胞における増殖は細胞分裂です。DNA のコドンに結合された化合物を使って、新たに確保した構造へ自身のコピーを作ることに当たります。

「増殖」の振る舞いを図示すると図 2.10 の通りです[*8]。



▲図 2.10: 増殖の振る舞い

新しい細胞を中身を空で生成した後、セントラルドグマにより元細胞の DNA から

[*8] 対応する PoC 実装はリポジトリ直下 cell.c の division 関数です。

タンパク質リストを生成し、新細胞のタンパク質リストへ繋ぎます。

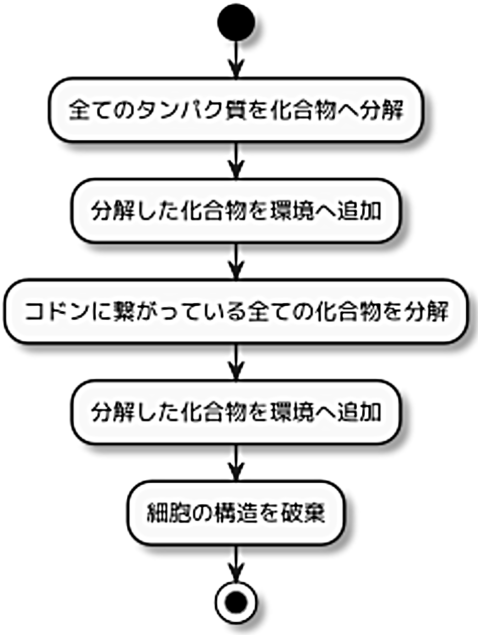
その後、コドンも中身を空で生成し元細胞の DNA のコドンをそこへコピーした後、コドンのリスト (DNA) を作成し、これも新細胞へ繋ぎます。

最後に元細胞の寿命情報を新細胞へコピーして、できあがった細胞を環境へ追加すれば終わりです。環境へ追加すれば、以降は処理系により実行されるようになります。

♣ 死

細胞においてもバイナリ世界においても、死はリソースの解放です。自身を構成している部品を環境へ解放します。

「死」の振る舞いを図示すると図 2.11 の通りです[*9]。



▲図 2.11: 死の振る舞い

死の際に細胞が分解して環境へ放出する要素は以下の 2 つです。

[*9] 対応する PoC 実装はリポジトリ直下 cell.c の death 関数です。

- タンパク質: 化合物のレベルまで分解し、環境へ放出
- DNA: コドンに結合している化合物があれば全て分離し、環境へ放出

これら 2 つの要素を環境へ放出した後は、細胞の構造を破棄します。

2.5 まとめ

この章では、バイナリの世界で細胞をどのように表現するか、その処理系も含め、設計とモデルを説明しました。ポイントをまとめると以下の通りです。

- 「細胞」はプログラムにおける「関数」に対応すると考える
  - 「タンパク質」は「命令」に対応し、「化合物」は「命令の構成要素」、「元素」は「バイト」に対応する
- この構造を用いて、生物の「5 つの振る舞い」を置き換えてモデル化した

また、本書の PoC 実装では、化合物をインクリメントして返す「インクリメンタ細胞」をもつ生物を作成します。この章で、この細胞の場合にどうなるかを適宜説明しました。

第 3 章　PoC 実装の実行結果

3.1　PoC 実装について

「代謝/運動」・「成長」・「増殖」・「死」の振る舞いを実際に確認する PoC 実装を行ってみました。

この PoC 実装では、前章のモデルを処理する処理系を実装した上で、その処理系の環境内に前章で紹介した単細胞生物「インクリメンタ」を 1 つ配置し、以下の振る舞いを確認します。

- 細胞が持つタンパク質 (命令) リストからメモリ上に関数を動的に構成し、同じく動的に取得した引数に対して処理を実行できる (代謝/運動)
- 環境内に配置したコード化合物の内、DNA に合致するものを動的に取り込む (成長)
- DNA に記載されたコード化合物を全て揃えると細胞分裂が行われる (増殖)
- 寿命を迎えるとタンパク質内の化合物と DNA のコドンが持つ化合物への参照を分解し環境へ放出する (死)

♣ 実装の特筆事項

実装の内容については本書で解説は行いません。設計は前章でモデル図も用いて紹介したとおりで、それをどのように実装するかは好み次第なところもあるので、もし興味があれば本書のサンプルリポジトリをご覧ください。

ここでは、実装に関する特筆事項のみ説明します。

まず、実装は筆者の趣味もあり、筆者の自作 OS 上のアプリケーションとして実装しています。本書で引用するスクリーンショットも自作 OS 上で動作しているものです。

ただし、前章でモデル図などで説明した設計は、Linux 等でも実装可能な内容ですので、もし (万が一、物好きにも。。) 興味があれば、ぜひご自身で実装してみてくだ

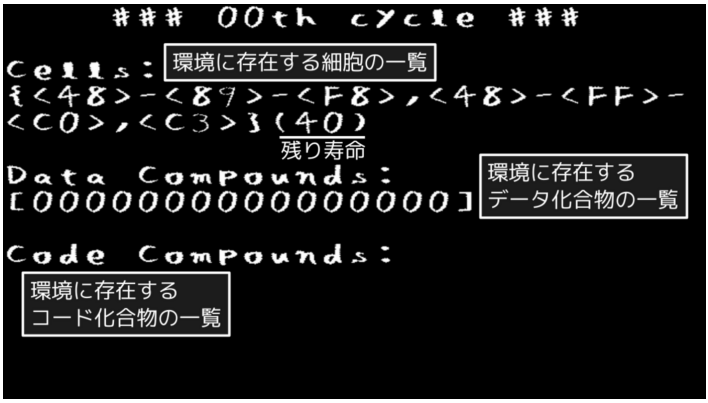
さい。

今回の PoC 実装では、主に簡単化のために以下の方針で実装しました。

- 実行バイナリはファイルからロードするのではなく、処理系の中で初期化時にメモリ上に作る
- 「DNA」という独立した構造は用意していない
  - 細胞がコドンのリストを持つ形にした
  - DNA という中間層は現状では実質機能しないためです
- 細胞/タンパク質/化合物/コドンのそれぞれは、予め中身が空の構造をグローバル変数でいくつか用意しておき、それを使い回します
  - 簡単のため (手抜きのため) です。。

3.2　PoC プログラムの実行結果と評価

PoC プログラムを実行すると図 3.1 の画面が表示され、自動で進んでいきます。



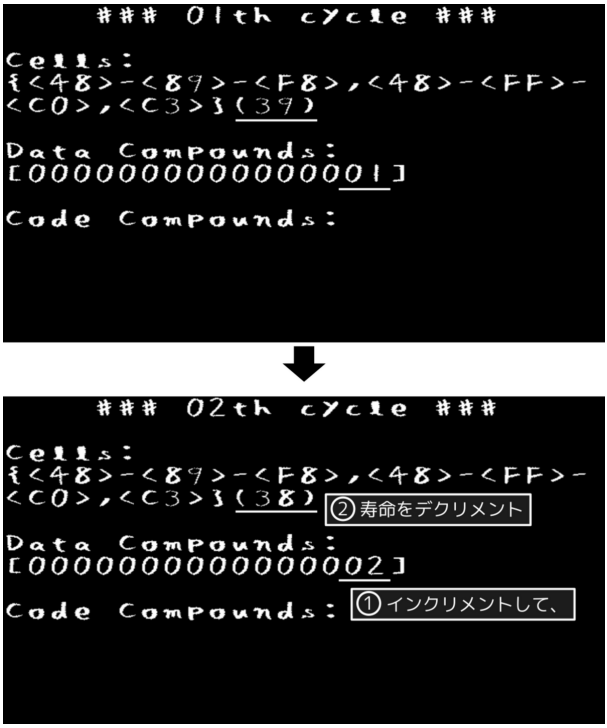
▲図 3.1: PoC プログラム起動直後の画面

画面には現在の周期 (「XXth cycle」の箇所) と、環境に存在する細胞 (「Cells:」の箇所) と、データ化合物・コード化合物 (「Data Compounds:」と「Code Compounds:」の箇所) が表示されていて、1 秒周期で進んでいきます[*1]。この画面で、以下の要素が現在の環境に存在することを示しています。

[*1] フレームバッファの制御を作り込んでいないため、特に実機で試す際はチラツキますが PoC 程度ということで気にしないでください。あるいは画面の更新箇所を減らすなど工夫してみてください。

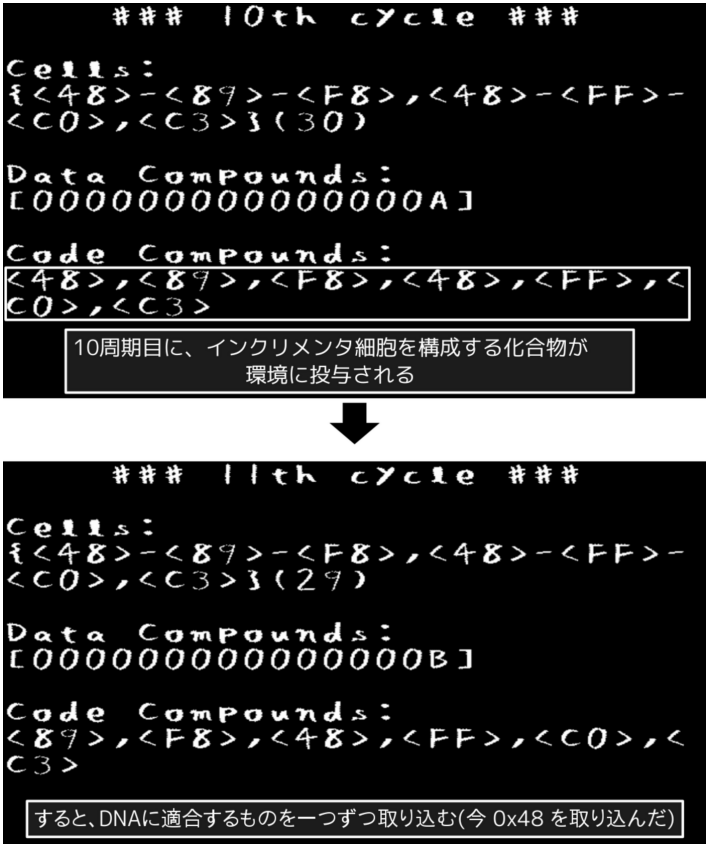
- 細胞
  - インクリメント関数の細胞 (単細胞生物)
  - 寿命は 40 周期
- 化合物
  - データ化合物
    - 8 バイト整数
    - 初期値 0
  - コード化合物
    - 無し

周期が進む毎にインクリメント細胞によりデータ化合物がインクリメントされていき (代謝/運動)、細胞自身の寿命も減っていきます (図 3.2)。



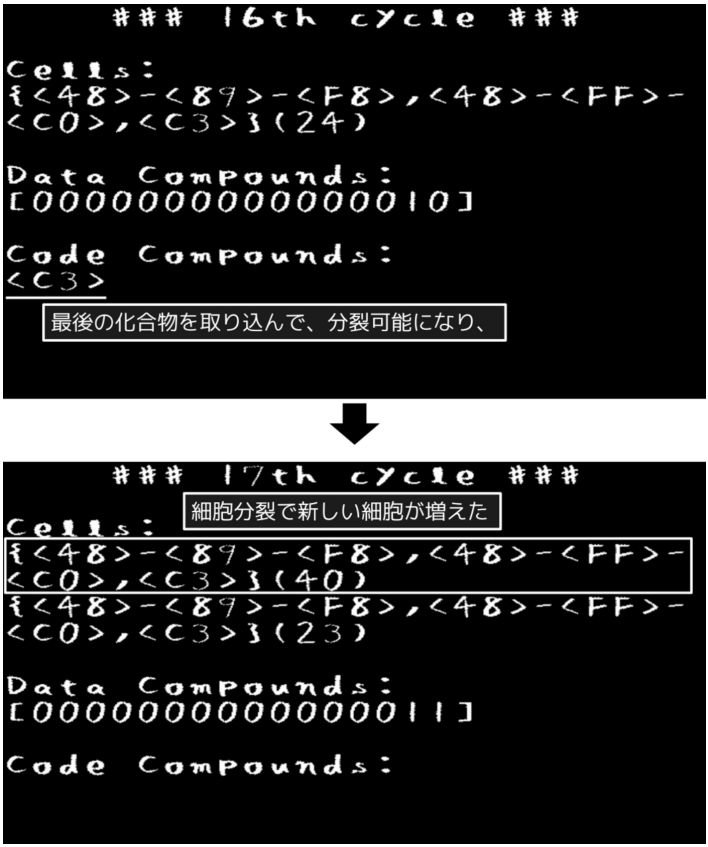
▲図 3.2: PoC プログラムの実行画面 (周期 01 - 02)

10 周期目に、インクリメント細胞のタンパク質を構成する (すなわち、細胞の DNA に記録されている) 全てのコード化合物が環境に投与されます。すると、インクリメント細胞は 1 周期に一つずつ化合物を取り込んでいきます (図 3.3)(成長)。



▲図 3.3: PoC プログラムの実行画面 (周期 10 - 11)

全ての化合物を取り込むと、DNA に記載された化合物が全て揃うので「分裂可能」となり、細胞分裂を行います (図 3.4)(増殖)。

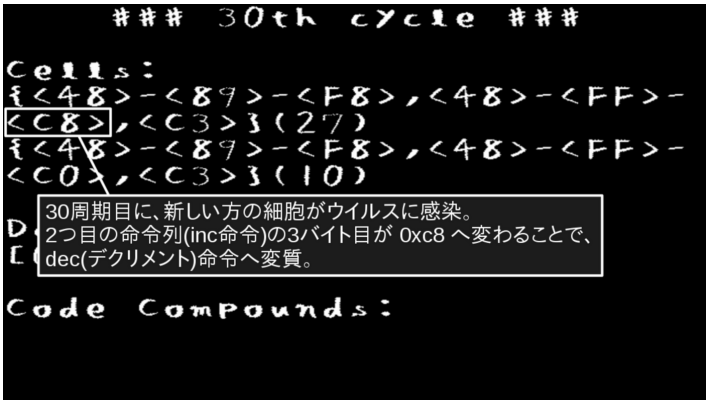


▲図 3.4: PoC プログラムの実行画面 (周期 16 - 17)

細胞分裂すると、インクリメンタ細胞が 2 つになるため、データ化合物は 2 ずつ増加するようになります (図 3.5)。

▲図 3.5: PoC プログラムの実行画面 (周期 17 - 18)

実験として、30 周期目に片方の細胞がウイルスに感染するように処理系へ仕掛けを入れてみました。30 周期目に若い方の細胞の 2 つ目のタンパク質 (inc 命令) の一番うしろの化合物を「0xc0」から「0xc8」へ変更してみました (図 3.6)。



▲図 3.6: PoC プログラムの実行画面 (周期 30)

【コラム】宣伝: 「作って分かる！ x86_64 機械語入門」

図を順に紹介していく都合上、余白ができてしまったので宣伝を。

本書をもし試す際は機械語の知識があると良いです。当サークルの既刊で、一つ一つ命令や構文を理解してくことで簡単なプログラムを作る機械語の入門本を出しています (図 3.7)。こちらも筆者のウェブサイト (「はじめに」参照) で公開していますので興味があれば見てみてください。

▲図 3.7: 作って分かる！ x86_64 機械語入門

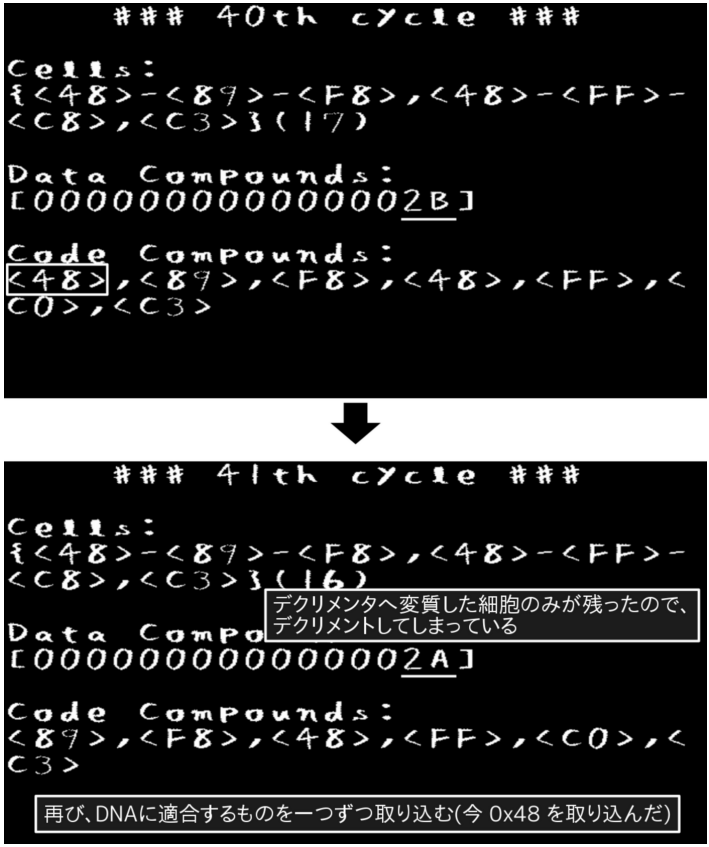
inc 命令の最後のバイトが「0xc8」になっていると、「デクリメント (dec) 命令」となるため、片方の細胞のインクリメントと相殺してデータ化合物が増減しなくなります (図 3.8)。これは、インクリメンタのみの世界観であるはずのこの環境としては異常な状態 (異物あるいは異常者が存在している状態) です。

▲図 3.8: PoC プログラムの実行画面 (周期 30 - 31)

そこから進むと、古い細胞 (ウイルス感染していない方) が寿命を迎え、死滅します。細胞は死ぬ際、自身を構成する要素を化合物のレベルまで分解して環境へ放出します (図 3.9)。

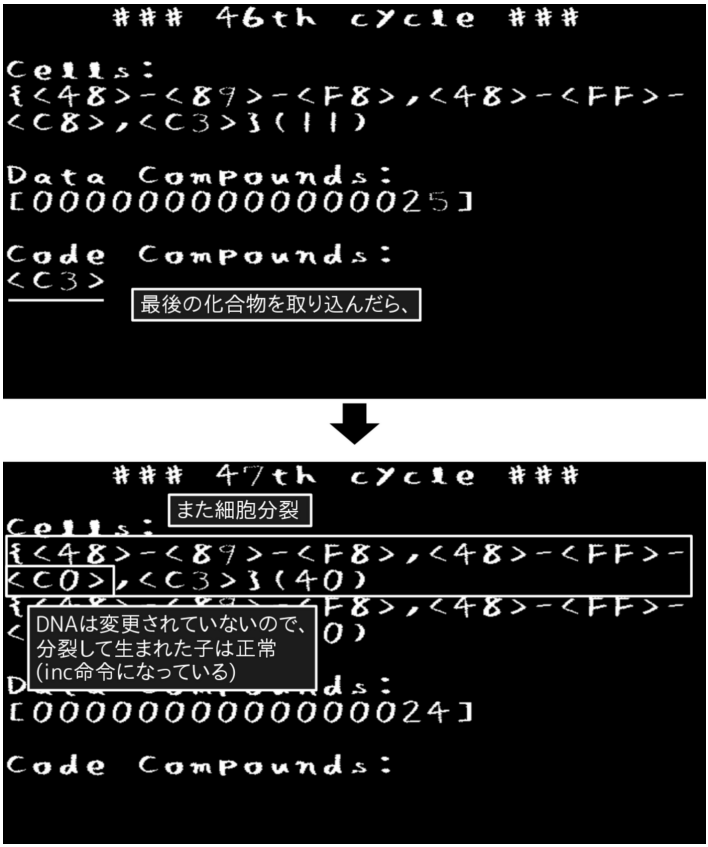
▲図 3.9: PoC プログラムの実行画面 (周期 39 - 40)

すると、先ほどと同様に、環境に放出された化合物を現在生きている細胞が取り込んでいきます。デクリメンタへ変質してしまっているので、その間、データ化合物はデクリメントしていきます。ただし、DNA のコドンは変質していないため、inc 命令の「0xc0」も取り込みます (DNA は表示させていないため画面上は変質していない事が分からないですが)(図 3.10)。



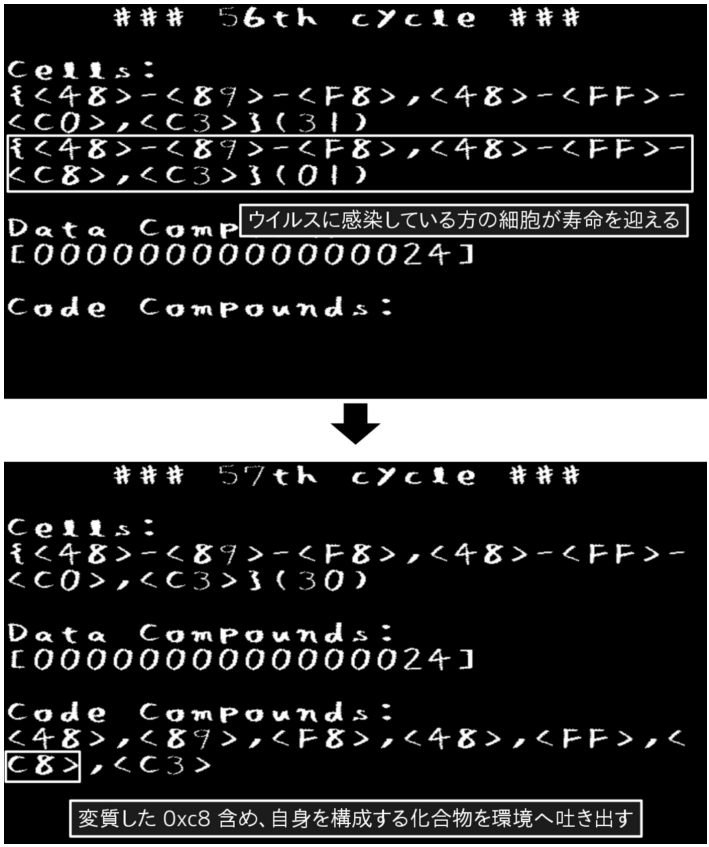
▲図 3.10: PoC プログラムの実行画面 (周期 40 - 41)

全ての化合物を取り込むと、DNA に記載された化合物が全て揃うため分裂可能となり、細胞分裂を起こします。その際、先ほども説明した通り、DNA は変質していないため、分裂して生まれた子は正常なインクリメンタ細胞です (図 3.11)。



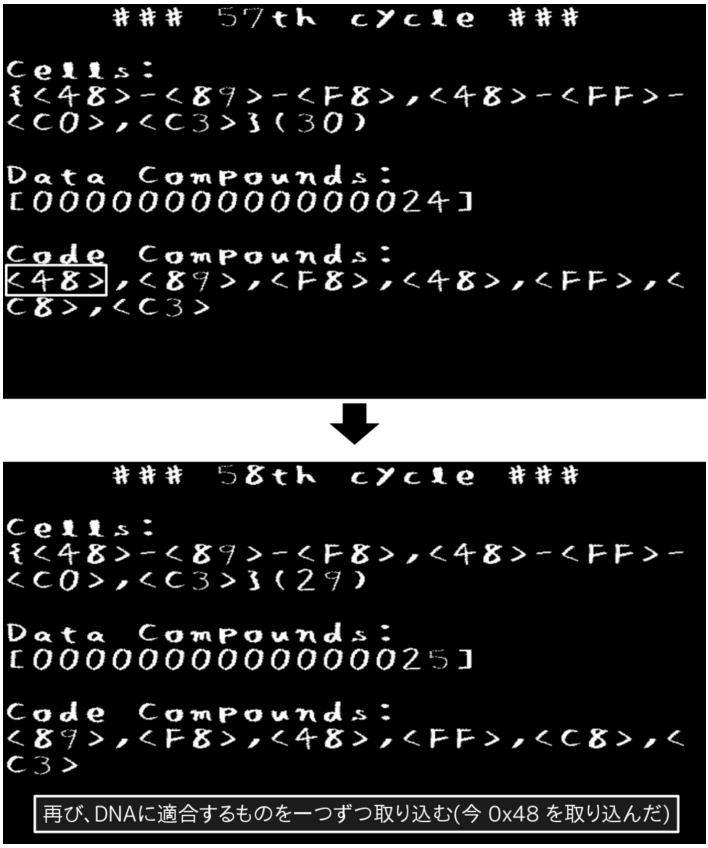
▲図 3.11: PoC プログラムの実行画面 (周期 46 - 47)

そして、今度はウイルスに感染している方が寿命を迎えます。細胞を構成する inc 命令の化合物が一箇所「0xc8」に変質しているため、それが環境へ放出されています (図 3.12)。



▲図 3.12: PoC プログラムの実行画面 (周期 56 - 57)

すると、再び残された細胞が放出された化合物を取り込んでいきます (図 3.13)。

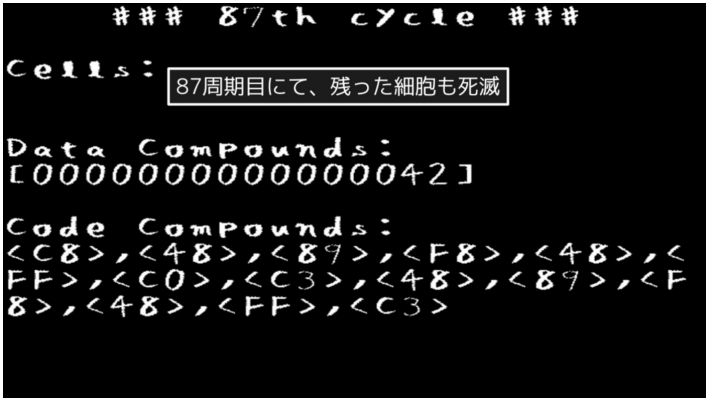


▲図 3.13: PoC プログラムの実行画面 (周期 57 - 58)

ただし、変質した「0xc8」の化合物だけは正常なインクリメンタ細胞の DNA には無いため、取り込まれずに残ります (図 3.14)。「デクリメンタが存在する」という異常状態は、「細胞はいずれ死ぬ」という機能により、異常状態から回復する事ができました。

▲図 3.14: PoC プログラムの実行画面 (周期 64 - 65)

なお、inc 命令を構成する化合物である「0xc0」が足りないため、残った細胞は細胞分裂を行うことができず、寿命を迎えて死滅します。このインクリメンタ細胞が繁栄してから滅亡するまでの結果、データ化合物に最終的に記録された数字は「0x42」でした (図 3.15)。



▲図 3.15: PoC プログラムの実行画面 (周期 87)

3.3 まとめ

この章では、前章で説明した単細胞生物と処理系の実装について特筆事項を紹介し、PoC 実装の実行結果を紹介しました。

実行結果から、本書で目的としている「生きていると言い張る」ための「5 つの振る舞い」が実際に動作していることを確認できました。

おわりに

ここまで、本書をお読みいただきありがとうございました！

本書では「生きたバイナリ」とどのように言い張るか (笑) を設計・モデル化し、PoC 実装で実現してみました。

ただ、これだけでは何にも使えず、本当にただ「生きている」という定義を満たしたに過ぎません。今後これをどうするのか、何に使えるのか、などもまだ研究中なところです。

「はじめに」で「究極のバイナリ」と説明した「自己進化」・「自己再生」・「自己増殖」の内、「自己再生」と「自己増殖」は一応実現できています。では、「自己進化」をどのように実現するか、ですが、例えば、「どのように自身のタンパク質と DNA を変質あるいは増減させるか」と「タンパク質と DNA の変質あるいは増減をシステムとしてどのようにコントロールするか」を解決できれば良さそうです。

前者は、「環境に存在するコード化合物が勝手にくっついて増大する」とか「細胞分裂のタイミングである確率で DNA が変質する」等が考えられますが、「ちゃんと実行できる命令列になるのか」が問題です。ただ、それは「実行してみて例外が発生したら『結合できないバイナリ列だった』と判断する」等の方法で判別可能です。

より問題なのは後者で、「実行できるがデタラメな命令列」に対してどの様に優劣を付けるのか、どの様に進化の方向性を制御するのか、等はまだ上手いアイディアがありません。ただ、ひとまずはシステムとしてはコントロールしないという手もあります。毎日一回、オペレーターが環境内の細胞をチェックし、ダメなものが居たら手動で削除する、とかでも良いのかなと。

やはり、現状はまだ「頭のおかしい与太話」に過ぎませんが、このようなシステムでしか解けない、あるいは、このシステムだからこそできる何かもあるんじゃないかなと思っています。それは普段私達が使っている「OS」とは UI から何から全く違ったものになるのかもしれませんが、そんな、「今までの OS の考え方と全く違ってしまうもの」を作ってしまえる (まあ趣味なら) のも、自作 OS の醍醐味ですね。

バイナリ生物学入門

バイナリを生物のように実行する
モデルと PoC 実装

2019 年 12 月 31 日　ver 1.0 （コミックマーケット 97 新刊）
2023 年 5 月 21 日　ver 1.1 （技書博指定の奥付を追加）

著　者　大神祐真
発行者　大神祐真
連絡先　yuma@ohgami.jp
　　http://yuma.ohgami.jp
　　@yohgami (https://twitter.com/yohgami)
印刷所　日光企画

(powered by Re:VIEW Starter)