UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

FELIPE SOBREIRA ABRAHÃO

METABIOLOGIA, SUBCOMPUTAÇÃO E HIPERCOMPUTAÇÃO:
em direção a uma teoria geral de evolução de sistemas

Rio de Janeiro

2015

Felipe Sobreira Abrahão

METABIOLOGIA, SUBCOMPUTAÇÃO E HIPERCOMPUTAÇÃO:

em direção a uma teoria geral de evolução de sistemas

Tese de Doutorado apresentada ao Programa de Pós-Graduação em História das Ciências e das Técnicas e Epistemologia, Universidade Federal do Rio de Janeiro, como requisito parcial à obtenção do título de Doutor em História das Ciências e das Técnicas e Epistemologia.

Orientador: Prof. Dr. Francisco Antonio de Moraes Accioli Doria
Co-Orientador: Prof. Dr. Gregory J. Chaitin

Rio de Janeiro
2015

Abrahão, Felipe Sobreira

A159m Metabiologia, subcomputação e hipercomputação: em direção a uma teoria geral de evolução de sistemas / Felipe Sobreira Abrahão. – Rio de Janeiro, 2015.

176 f.

Orientador: Francisco Antônio de Moraes Accioli Doria.

Coorientador: Gregory John Chaitin.

Tese (doutorado) – Universidade Federal do Rio de Janeiro, Decania do Centro de Ciências Matemáticas e da Natureza, Programa de Pós Graduação em História das Ciências e das Técnicas e Epistemologia, 2015.

1. Metabiologia. 2. Evolução *Open-Ended* de Sistemas 3. Biologia Evolutiva. 4. Computabilidade I. Doria, Francisco Antônio de Moraes Accioli , oriente. II. Chaitin, Gregory John, coorient. III. Título.

Felipe Sobreira Abrahão

METABIOLOGIA, SUBCOMPUTAÇÃO E HIPERCOMPUTAÇÃO:
em direção a uma teoria geral de evolução de sistemas

Tese de Doutorado apresentada ao Programa de Pós-Graduação em História das Ciências e das Técnicas e Epistemologia, Universidade Federal do Rio de Janeiro, como requisito parcial à obtenção do título de Doutor em História das Ciências e das Técnicas e Epistemologia.

Aprovado em ___/___/_____.

_____
Francisco Antônio de Moraes Accioli Doria, Ph.D., UFRJ

_____
Gregory John Chaitin, Professor Colaborador Pleno, Ph.D., PEP - COPPE - UFRJ

_____
Prof. Mércio Pereira Gomes, Ph.D., HCTE - UFRJ

_____
Décio Krause, Ph.D., Universidade Federal de Santa Catarina

_____
Ítala Maria Loffredo D'Ottaviano, Ph.D., UNICAMP

_____
Marcelo Esteban Coniglio, Ph.D., Universidade de São Paulo

Aos meus amados pai e avô

AGRADECIMENTOS

Gostaria, primeiramente, de agradecer aos meus orientadores Francisco Antônio Dória e Gregory Chaitin por todos seus ensinamentos e por suas companhias durante esses anos. Esta tese é, sem dúvida, fruto de toda a ajuda e inspiração que obtive do convívio com eles.

Gostaria também de agradecer a todos os outros professores, principalmente aos professores do HCTE (Programa de Pós-Graduação em História das Ciências e das Técnicas e Epistemologia), com os quais tive o prazer e a honra de apreender e conviver. Assim como, a todos os funcionários do HCTE e da UFRJ.

Aos meus colegas do meu curso e de outros cursos que cruzaram meu caminho nessa jornada acadêmica. Principalmente, à Virginia Maria Chaitin, a qual tem sido praticamente uma terceira orientadora no meu doutorado.

À Sonia Regina Pereira Cardoso, por todo carinho e pela ajuda tanto na correção quanto na elaboração desta tese.

A todos os amigos que tecem, verdadeiramente, a pequena, mas estimada, malha social do que vos fala.

A todos os meus familiares, especialmente a minha mãe Luciene Coimbra Sobreira e a minha irmã Ana Carolina Sobreira Abrahão, os quais sempre foram muito presentes em minha vida e sem os quais seria extremamente difícil a realização deste trabalho.

À minha amada companheira Raquel Cardoso Oscar, com a qual tenho passado os melhores e eternos momentos da minha vida.

Ao meu pai Guilherme José Abrahão e ao meu falecido avô José Abrahão. Não é exagero dizer que sou consequência de todo ensinamento e, principalmente, inspiração científica e artística que eles plantaram em mim.

RESUMO

ABRAHÃO, F. S. **Metabiologia, Subcomputação e Hipercomputação: em direção a uma teoria geral de evolução de sistemas**. Rio de Janeiro, 2015. Tese (Doutorado em História das Ciências e das Técnicas e Epistemologia) - Programa em História das Ciências e das Técnicas e Epistemologia, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2015.

A metabiologia é uma teoria matemática baseada, principalmente, na teoria da informação algorítmica, nos permitindo estudar a evolução *open-ended* de programas, isto é, estudar quão rápido os organismos/programas se tornam mais complexos ou mais criativos, sem nunca estagnar num nível máximo de complexidade ou criatividade. Tomamos como base os modelos metabiológicos originais chaitinianos, nos quais os organismos são sistemas computáveis (i.e., sistemas emuláveis por um computador), a natureza (ou meio-ambiente) é um sistema hipercomputável (i.e., um sistema emulável apenas por algum hipercomputador), as mutações são sistemas computáveis aleatoriamente gerados (i.e., mutações algorítmicas aleatórias) e a aptidão dos organismos é medida estritamente por quão grande é o output desses organismos. Nesse âmbito, este trabalho se foca em responder duas questões. A metabiologia seria inapropriada se os organismos não forem sistemas computáveis? Ou, na direção oposta, ela seria inapropriada se a própria Natureza for um sistema computável? Portanto, o objetivo principal é mostrar que a metabiologia caminha em direção a uma teoria geral para evolução *open-ended* de qualquer sistema, seja ele subcomputável, computável ou hipercomputável. Como objetivo secundário, mostramos que as consequências científicas e filosóficas dos nossos resultados matemáticos podem se alastrar pelas questões fundamentais da biologia e vida artificial, como também, da física e da inteligência artificial. Para satisfazer esses dois objetivos, construímos dois novos modelos metabiológicos para duas configurações, diferentes entre si, de organismos e natureza. Primeiro, mostramos que é possível a evolução metabiológica de subprogramas, dentro da qual tomamos a natureza como sendo computável, o que fez com que os organismos e as mutações passassem a sistemas subcomputáveis. Para essa demonstração, apresentamos e utilizamos o fenômeno da incomputabilidade relativa recursiva (uma versão computável/recursiva da incomputabilidade relativa que ocorre entre as diferentes ordens de hipercomputadores dentro da hierarquia dos graus de Turing). Fazemos com Turing, Radó e Chaitin o que Skolem fez com Cantor: relativizamos a incomputabilidade da função *Busy-Beaver* atrelada ao número Ω para ela existir, de modo análogo, entre um computador e qualquer subcomputador do mesmo. Em

decorrência, apresentamos o “paradoxo” da computabilidade, o qual diz que existe, pelo menos, uma função que não é computável se “olhada de dentro”, mas é computável se “olhada de fora”. Então, correlacionamos esse pseudoparadoxo com problemas da criatividade humana e inteligência artificial, da consciência, da vida artificial e da computabilidade do Universo. Segundo, construímos um modelo metabiológico para a evolução de hiperprogramas, no qual os organismos são sistemas hipercomputáveis de qualquer ordem finita (dentro da hierarquia de Turing) e a natureza é um hiperprograma de ordem ordinal ω. Nesse último modelo, mostramos que é possível uma evolução de organismos/hiperprogramas por meio de mutações algorítmicas aleatórias “cegas” (i.e., que não levam em conta os organismos/hiperprogramas anteriores) que vai levando os organismos/hiperprogramas a serem capazes de resolver qualquer problema na hierarquia aritmética. Aliado a isso, discutimos sobre a computabilidade ou incomputabilidade dos seres vivos e sobre a possibilidade de estudar matematicamente a evolução metabiológica de sistemas que podem ultrapassar, o quanto se queira, a “barreira” do computável.

Palavras-chave: Metabiologia. Evolução *Open-Ended* de Sistemas. Biologia Evolutiva. Computabilidade. Complexidade. Hipercomputação. Subcomputação.

ABSTRACT

ABRAHÃO, F. S. **Metabiologia, Subcomputação e Hipercomputação: em direção a uma teoria geral de evolução de sistemas**. Rio de Janeiro, 2015. Tese (Doutorado em História das Ciências e das Técnicas e Epistemologia) - Programa em História das Ciências e das Técnicas e Epistemologia, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2015.

Metabiology is a mathematical theory mainly based on algorithmic information theory and allows us to study an open-ended evolution of programs, that is, to study how fast the organisms/programs become more complex or more creative without never stagnate at a maximum level of complexity or creativity. We take, as starting point, the first chaitinian metabiological models in which the organisms are computable systems, Nature (or environment) is a hyper-computable system, mutations are randomly generated computable systems (that is, random algorithmic mutations) and the fitnesses are measured by how big is the output of the organisms. In this context, this paper focuses on two questions: Assuming that organisms are uncomputable systems, should metabiology become unsuitable or unsound? Or, on the other hand, assuming that Nature itself is a computable system, should metabiology become unsuitable or unsound? Thus, the main purpose is to demonstrate that metabiology walks towards a general theory of open-ended evolutionary systems, whether these systems are sub-computable, computable or hyper-computable. As a secondary purpose we show that scientific and philosophical consequences of our mathematical results spread across fundamental questions about biology and artificial life, as well as, about physics and artificial intelligence. To satisfy these two objectives, we build two new metabiological models for two new different configurations of organisms and nature. First, we demonstrate it is possible to build a metabiological evolution in which nature is a computable system and, hence, it makes the organisms and mutations to be sub-computable systems. To show this we introduce the recursive relative uncomputability (a computable/recursive version of the relative uncomputability that occurs among different Turing degrees). We do with Turing, Radó and Chaitin the same Skolem did with Cantor by making a relativization of the Busy-Beaver´s uncomputability in relation to the Omega number, so that this new uncomputability

can exist between a computer and any of his sub-computers. As a consequence, we introduce the “paradox” of uncomputability which says that exists at least one function that is not computable if we “look from the inside”, but that is computable if we “look from the outside”. Further, we correlate this pseudo-paradox with problems of human creativity and artificial intelligence and with problems of consciousness, artificial life and the computability of the Universe. Second, we build a metabiological model for the evolution of hyper-programs in which the organisms are hyper-computable systems of any finite order and nature is a hyper-program of ordinal order ω. In this last model we demonstrate it is possible an evolution of organisms/hyper-programs, with just “blind” random algorithmic mutations (it means, that do not take into consideration previous organisms/hyper-programs), which progressively leads the organisms/hyper-programs to be able of solving any problem in the arithmetical hierarchy. Allied to this, we discuss about computability or uncomputability of “living beings” and upon the possibility of mathematically studying metabiological evolutionary systems that can surpass, as much as we want, the “frontier” of the computable.

Keywords: Metabiology. Open-Ended Evolutionary Systems. Evolutionary Biology. Computability. Complexity. Hypercomputation. Subcomputation.

SUMÁRIO

INTRODUÇÃO

A biologia é reconhecidamente um campo de estudo científico no qual as relações entre as coisas são tão imbricadas e as tão desejadas “leis” são tão cheias de “exceções à regra” que parece quase impossível termos o mesmo sucesso para a biologia que a matemática teve em estruturar uma física teórica. E com certeza isso foi sempre tentado, sobretudo, ao longo do século XX.

Quem sabe, a matemática usada até os últimos tempos para modelar os sistemas físicos não seja a mais apropriada? Essas matematizações, por vezes extremamente difíceis, podem não ser suficientes para abarcar toda a complexidade das relações biológicas. Todas elas buscam a construção de equações ou esquemas de equações que quase nunca dão espaço para exceções. E quando dão, não descrevem o comportamento minucioso e quase imprevisível dos elementos constituintes do sistema (por exemplo, as médias da física estatística ou da genética de populações).

Os sistemas biológicos são dinâmicos e se modificam a toda hora. Seus elementos constituintes (moléculas, elementos químicos, sinais elétricos...) exercem funções que dependem muito das condições do entorno e do estado de outros elementos do sistema. Conseguir montar uma equação que traduza esse **processo** de transformação interna pelo qual passa todo ser vivo, a todo o momento, parece ser uma tarefa hercúlea. Contudo, se nos aferroarmos justamente à palavra ‘processo’, a própria matemática pode nos dar uma dica sobre que arcabouço teórico usar.

Pensando a teoria da computação como o estudo de objetos matemáticos e abstratos chamados de **programas**, estes nada mais são que matematizações de procedimentos, algoritmos ou processos pelos quais um **sistema** passa ao longo do tempo. São *bit strings* (uma cadeia finita de 0´s e 1´s numa fita de máquina de Turing universal). Nada mais que a forma mais geral de definirmos matematicamente um programa – ou, se desejar, um algoritmo. O que é computável o é por um programa, ou seja, existe um programa que pode realizar a operação desejada (por exemplo, decidir se um número natural pertence ou não a um conjunto).

E o que consideramos, aqui, um **sistema**? Essa definição será crucial para todo o trabalho. De fato, partiremos do princípio de que os seres vivos são sistemas. Podemos chamar essa hipótese de **hipótese sistêmica da vida.** Para nossos propósitos, um sistema pode

ser entendido de forma bem geral e simples: ele é uma sucessão de estados compostos por objetos ou entes. A mudança de um estado para outro sempre define uma função com o estado anterior como *entrada* (***input)*** e o próximo estado como *saída (**output**)*, por mais complicada e abstrata que ela seja.[1] Essa maneira mais "matemática" de definir um sistema é abrangente o suficiente para contemplar qualquer tipo de conjunto de coisas que se modifica com o tempo.

Por exemplo, podemos pegar a, também bastante geral, definição de sistema dada por Bertalanffy, em seu canônico livro sobre uma teoria geral de sistemas, que considera um sistema como um conjunto de elementos em interação.[2] Essas interações produzem uma mudança de estados desse conjunto, e não só isso, são elas que definem abstratamente a função que diz qual estado sucederá o outro. Logo, essa definição também cai sob a nossa definição de sistema. Além disso, definir sistema por meio de interações deixa um pouco nebuloso o caso de um conjunto de elementos cujos estados mudam com o tempo, mas que não se pode inferir que haja nenhuma interação ocorrendo entre esses elementos, ou ainda, que não haja relação causal entre seus estados. Por quê? A mudança dos estados pode não estar sendo feita pela interação entre os elementos, porém, pode estar sendo feita por algum agente externo – não importa se existente ou em potencial, material ou metafísico. E isso não nos impede de pensar tal sucessão de estados como um sistema, afinal seus elementos estão interagindo, apenas não diretamente, mas por intermédio de "algo" externo. Não é porque não se vê uma relação causal entre os estados que não há relação[3] entre eles. De qualquer forma, mesmos esses casos menos evidentes ou complicados se encaixam em nossa definição de sistema.

O leitor logo pode se perguntar: mas essa função de mudança de estados não precisa ser computável para o sistema poder ser completamente emulável por um programa? Sim. Dessa forma, nos fixaremos no estudo da evolução, primeiro, de sistemas computáveis e, depois, de sistemas não computáveis ou hipercomputáveis.

É preciso lembrar que essa ideia de fazer uma analogia entre seres vivos e programas não é nova. Ela já vem desde Von Neumann com a ideia de autômato celular, como contado pelo nosso principal autor de referência, Gregory Chaitin. E se formos mais longe, veremos

---

[1] Caso não haja um referencial temporal para que se defina uma sucessão, podemos usar o conceito lógico-matemático mais abstrato de relação ao invés de função.

[2] BERTALANFFY, L. V. **General System Theory:** Foundations, Development, Applications. New York: George Braziller, 1968. ISBN ISBN 0-8076-0453-4. p. 38

[3] Em seu sentido mais amplo.

que ela está ligada aos trabalhos de Turing sobre computação. Como curiosidade, o famoso biólogo Sydney Brenner menciona os trabalhos de Von Neumann como inspiração no estudo do DNA.

Onde, então, podemos usar programas para estudar os fenômenos biológicos? Para responder essa pergunta vamos direto à teoria na qual esse trabalho se baseia: a **metabiologia.** Podemos entendê-la, de forma geral, como sendo uma teoria matemática para a evolução *open-ended* de programas, software. Ela se inspirou nas teorias da **evolução biológica**, assim como nos conceitos de complexidade (*program-size complexity)* e de **probabilidade algorítmica**, ambos da **teoria da informação algorítmica** (TIA) – organismos evoluindo seriam programas que aumentam de complexidade. O que, grosso modo, quer dizer que se pode ir computando problemas que necessitariam cada vez mais de programas maiores para computar.

Como já se pode intuir a partir daí, a metabiologia não faz uma matematização dos organismos que visa simplificá-los. Pelo contrário, ela permite que qualquer organismo/programa exista e qualquer coisa que esse programa faça é relevante para os resultados finais: as aptidões. Em particular, o espaço de possibilidades é definido sobre todos os programas possíveis. Tudo que for computável – ou hipercomputável, como veremos –, é permitido. As coisas podem ser tão complexas quanto se queira e, justamente, o quão complexas elas são, ou se tornam, é o crucial para a teoria.

Gregory Chaitin nos mostrou que é possível construir um modelo matemático bastante abstrato e geral para a evolução dos **seres metavivos** (nossos organismos matemáticos da metabiologia, "feitos" puramente de informação algorítmica), o qual mostra um ganho real de complexidade, atrelado a um ganho de aptidão, à medida que as mutações algorítmicas aleatórias vão acontecendo, uma depois da outra com a presença da "seleção natural". Isso, juntando[4] conceitos de organismo com programa, desenvolvimento e metabolismo com cômputo, aptidão com números naturais cada vez maiores, complexidade biológica com complexidade algorítmica[5] e natureza com hipercomputação.

Nesse modelo já apresentado na metabiologia, os seres vivos e as mutações (estas, aleatoriamente geradas) são programas. Explicando melhor essa demonstração, monta-se um sistema com apenas um organismo, isto é, um programa qualquer, o qual pode ser

[4] Migrando e, talvez, fundindo.
[5] *Program-size complexity*.

transformado em outro organismo/programa por outro programa/mutação gerado de forma aleatória, isto é, uma mutação algorítmica aleatória. Este último organismo também pode ser modificado por outra mutação, e assim por diante. A natureza é um sistema não computável, emulável apenas por um hipercomputador e, por conseguinte, capaz de decidir sempre se um programa/organismo se detém ou não. Ela só permite que vingue os organismos que possuírem uma aptidão maior que a do anterior, o que podemos chamar de **"seleção natural"**. Essa **aptidão** nada mais é que o output do organismo/programa: um número natural, por exemplo. Por isso, é possível a evolução de programas, levando estes a se aproximarem cada vez mais da complexidade de Ω, a qual é infinita.[6] A função usada para medir a aptidão dos organismos é uma versão da conhecida função ***Busy-Beaver*** originalmente formalizada por Tibor Radó (1962). Essa versão da Busy-Beaver é denotada por $BB'(N)$. Com ela, fazemos a ligação com o número Ω. Um programa/organismo gerar um output/aptidão maior ou igual a $BB'(N)$ quer dizer, obrigatoriamente, que ele possui, no mínimo, N bits de **criatividade ou de complexidade incompressível**. Portanto, essa função é usada para "medir" o quão complexos se tornam os organismos ao longo da evolução.

O próprio uso, por nós, do termo **evolução** assume um caráter específico e definido. Para os nossos fins, evolução significa um processo pelo qual os organismos (vivos ou metavivos) ganham informação (que antes não tinham acesso). Ou ainda, um processo pelo qual os organismos aumentam de complexidade ou se tornam mais criativos. Por conseguinte, evoluir pode ter a ver ou não com se **adaptar.** Nossa aptidão é um referencial absoluto, que vale para qualquer organismo, o que não é o caso no mundo biológico. O meio ambiente sempre varia de local para local e de tempo em tempo. Se adaptar a um "entorno" pode não significar se adaptar a qualquer "entorno" e, portanto, não podemos dizer que adaptação corresponde a um aumento de aptidão absoluta. Mas, sim, apenas um aumento relativo ao ambiente em que o organismo em questão está inserido. Um organismo A pode estar mais adaptado em relação a um "entorno" A' do que um organismo B em relação ao "entorno" B' e, mesmo assim, B ser mais complexo que A. Por exemplo, sob certo ponto de vista, os seres vivos procariontes estão mais bem adaptados à Terra do que nós, humanos, mesmo considerando que um corpo humano é mais complexo que qualquer bactéria. Diferentemente, para a metabiologia e esta tese, se a aptidão dos organismos aumentar uma quantidade grande

[6] O número ômega é um número real entre 0 e 1, cujos dígitos depois da vírgula são incomputáveis.

suficiente, a complexidade dos organismos precisa ter aumentado. Há, de fato, muitas controvérsias na literatura biológica sobre a diferença entre adaptabilidade e evolução. Evitaremos maiores discussões sobre isso e nos ateremos sobre nossa noção de aptidão e evolução.

São apresentados três modelos, os quais chamaremos de **chaitinianos**. Em primeiro lugar, a **busca exaustiva** usa apenas mutações algorítmicas aleatórias que ignoram seu input, i.e., ignoram o organismo anterior. Podemos chamar essas mutações de "cegas". Estima-se a quantidade média de mutações (o **tempo de mutação)** que deve ser tentada até aparecer uma mutação que retorne como seu output um organismo/programa que tenha, ou produza, output/aptidão $\geq BB'(N)$. O tempo para isso acontecer é de cerca de 2^N. O **projeto inteligente** modela a evolução usando mutações escolhidas pela natureza/hiperprograma. Cada mutação algorítmica adiciona 1 bit de criatividade para o organismo novo em relação ao anterior. Assim, como N bits de Ω são suficientes[7] para calcularmos $BB'(N)$, somente N mutações são necessárias para atingirmos N bits de criatividade, ou seja, o tempo de mutação no projeto inteligente é $\leq N$ – é linear ou até mais rápido. Já a **evolução cumulativa** mistura as duas coisas. Esse modelo usa as mutações inteligentes do projeto inteligente, porém, elas não são escolhidas pela natureza/hiperprograma. São aleatórias, possuem uma probabilidade de ocorrer. Portanto, estimamos a quantidade média de mutações algorítmicas aleatórias para a sequência mostrada no projeto inteligente acontecer, obtendo um tempo de mutação $\leq N^2(\log_2 N)^{1+\epsilon}$, onde ϵ é uma constante.

O resultado da evolução cumulativa nos mostra que a evolução por mutações aleatórias pode, de fato, aumentar a criatividade ou complexidade dos organismos/programas em um tempo bastante razoável: no máximo e aproximadamente em um tempo quadrático. Apesar de as mutações serem algorítmicas (programas) e não mudanças pontuais no genoma/programa do organismo anterior, esse modelo nos põe mais próximos de uma prova matemática para a evolução *à la* **Darwin**, ou ainda, *à la* **neodarwinismo**. Isso, com certeza, pode gerar grandes discussões. Contudo, não será nosso objetivo aqui.

Mas, por que a natureza como um hipercomputador? Hipercomputadores são aparatos hipotéticos capazes de "computar", da mesma forma que o construto teórico-matemático chamado máquina oracular de Turing, bem conhecido na literatura. Hiperprogramas são

[7] A menos de uma constante.

programas que só podem ser rodados por hipercomputadores. A natureza metabiológica dos primeiros modelos feitos por Chaitin precisa conseguir resolver problemas incomputáveis em relação a qualquer computador, ideal ou usual. Esses problemas incomputáveis envolvem, justamente, decidir se programas arbitrários param ou não – o conhecido *problema da parada* de Turing. Portanto, essa natureza é um hiperprograma de ordem 1. Por conseguinte, ela é capaz de saber todos os dígitos de Ω, isto é, ela seria equivalente a uma máquina oracular de Turing de primeira ordem. Ela é assim justamente porque é capaz de decidir sempre se um programa arbitrário se detém ou não, como *já dito.*[8]

Dessa forma, nos ambientamos no panorama geral do que iremos tratar nesta tese. Não nos aprofundaremos no que seja a metabiologia, nos seus fundamentos e nem sobre seus resultados. Vamos partir daqui para elaborar, principalmente, duas outras problemáticas: a **evolução de subprogramas** e a **evolução de hiperprogramas**.

Uma vez assumido – ou tomado como hipótese – os resultados da metabiologia como relevantes e profícuos para o estudo da evolução da vida, cabem duas perguntas e, ao mesmo tempo, duas críticas: a Natureza pode não ser um hipercomputador, ou seja, ela pode ser um sistema computável? Os organismos podem não ser sistemas computáveis, ou seja, os organismos podem ser hiperprogramas? E, logo, somos obrigados a nos questionar: a metabiologia pode nos servir nesses dois casos, diametralmente opostos?

O objetivo principal deste trabalho é mostrar que, mesmo nessas duas opções, podemos construir modelos para a evolução análogos aos já feitos por Gregory Chaitin. Dito de outra forma, obtemos teoremas usando a matemática da metabiologia, os quais mostram que os organismos evoluem mesmo que a natureza seja computável ou, diametralmente opondo, mesmo que os organismos não sejam computáveis. Assim, pretendemos mostrar que a metabiologia é um campo de estudo muito mais abrangente, podendo servir ao estudo da evolução de qualquer sistema, seja ele computável ou não. Isto é, a metabiologia resiste a essas duas críticas. Como objetivo secundário – ou como provocações –, em decorrência dos conceitos novos (ou atualizados) elaborados para resolver esses dois casos, mostraremos que as consequências científicas e filosóficas dos nossos resultados matemáticos se alastram pelas questões fundamentais da biologia e vida artificial, como também da física e da inteligência artificial.

---

[8] Note que computar o número ômega e resolver o problema da parada são problemas equivalentes.

Nem todos os conceitos e definições poderão ser aqui explicados. De fato, um conhecimento prévio de teoria da informação algorítmica e de metabiologia, assim como certa familiaridade com alguns conceitos matemáticos, lógico-matemáticos e de teoria da computação se fazem necessários porque, infelizmente, não conseguimos nos abster de todo desse jargão e nem explicá-lo desde seu início. Tais conhecimentos se fazem, sobretudo, necessários para entender as demonstrações. Porém, possivelmente, não as ideias – assim gostaríamos.

O presente texto não se sustentará apenas sobre demonstrações ou argumentos matemáticos ou formais, pois queremos nos debruçar também sobre alguns problemas que envolvem esses resultados.[9] Assim, precisamos também dizer que tudo aqui deriva desse conceito forte e ubíquo de **computação** – fruto, principalmente, dos trabalhos de Turing. Portanto, de início, precisamos frisar que o leitor precisa estar um pouco familiarizado com o jargão computacional, como, por exemplo, máquinas de Turing, cômputo, emulação, input/output, etc. E, ao lado disso, do jargão biológico: ser vivo, mutação, evolução, aptidão, complexidade, etc. Este trabalho, assim como a própria metabiologia, trata de demonstrações matemáticas em teoria da computação, porém nosso pé está fincado na biologia – em particular, na teoria da evolução. Decorrente disso, herdamos as dificuldades inerentes de todo campo de investigação interdisciplinar.

Como dito, o leitor pode necessitar ter familiaridade com mais de uma área do conhecimento. E não nos limitaremos às peculiaridades da metabiologia, também faremos algumas discussões envolvendo física e ciência cognitiva. Visto que resultados e questionamentos novos serão feitos partindo de nosso campo de estudo interdisciplinar, talvez devêssemos informar ao leitor que é mais acurado e prudente classificarmos o tema desse trabalho como *transdisciplinar*. Além disso, precisamos dizer que a tese é muito mais sobre mostrar o desabrochar de várias ideias implicadas no tema abordado do que procurar solucionar ou resolver tudo.

Deve-se ter em mente que a pesquisa foi feita usando duas vias metodológicas. A primeira, a investigação matemática. Grande parte do que foi obtido resultou de

[9] Inclusive, pretende-se que as ideias apresentadas serão mais importantes que as próprias provas. Caso estas se mostrem falhas no futuro, pretendemos que as ideias de evolução de subprogramas, incomputabilidade relativa e evolução de hiperprogramas se mostrem pregnantes e profícuas. Além disso, mesmo nesse caso, acreditamos ou apostamos que alguém possa prová-las.

demonstrações matemáticas, obtenção de lemas e teoremas. Mas não nos prendemos às formalidades; para melhor compreensão, julgamos necessário sempre colocar as explicações sobre os conceitos e definições, assim como a apresentação das ideias principais das demonstrações. Estas discussões virão antes dos textos formais.

A segunda via foi a discussão filosófica, tanto sobre os fundamentos da teoria quanto sobre seus desdobramentos. Portanto, muitos conceitos já correntes da literatura não serão esmiuçados e nem vamos nos ater a defini-los, pois nosso objetivo é relacioná-los com os problemas do nosso tema, e não investigá-los. Além disso, todos eles podem ser buscados na literatura de referência listada ao final de nosso estudo.

Especificarmos, na Introdução, todas as hipóteses e conceitos principais em que vamos nos calcar ficaria confuso. Ao longo do texto eles ficarão evidentes e serão mencionados. É claro que podemos focar apenas nos resultados matemáticos, pois basta que sejam válidas as teorias matemáticas sobre as quais nos debruçamos. E, por consequência, não prescindiríamos de hipóteses científicas para defender nosso objetivo principal - que é sobre qualquer coisa que se encaixe nas nossas respectivas definições de sistema, aptidão e natureza. Porém, como já afirmado, vamos além. Quando começarmos a fazer as discussões científico-filosóficas, serão apresentados as hipóteses e conceitos necessários para cada problemática.

Devotaremos os capítulos 1 e 2 ao estudo do modelo metabiológico em que a natureza é um sistema computável. Para esse fim, nos vemos obrigados a introduzir um novo conceito: o de **subcomputação.** Vamos no valer também de – talvez – um novo fenômeno, ao qual daremos o nome de **incomputabilidade relativa recursiva**, mostrada no teorema 2.15.8. Um **subcomputador** ou uma **submáquina de Turing** é um computador ou máquina de Turing[10] para o qual qualquer programa sempre gera um output. Ele é sempre definido tomando-se como base um "computador mais poderoso" que o "contém". Assim, conseguimos colocar esse "computador mais poderoso" no papel da natureza metabiológica e os seres vivos ou organismos como seus subsistemas, ou seja, como seus subcomputadores.

Dessa dicotomia entre sistema e subsistema brota a incomputabilidade relativa recursiva que entra no lugar da **incomputabilidade clássica** – em particular, a incomputabilidade da função *Busy-Beaver* $BB'(N)$ e do número Ω -, fundamental para fazer a complexidade dos organismos/programas crescer ao longo da evolução. Pretendemos provar

[10] Na prática, se considera um computador como sendo uma máquina de Turing com recursos limitados. Contudo, para o propósito matemático da nossa tese, vamos usar os dois termos como sinônimos.

que os subprogramas evoluem usando um teorema análogo, ou isomorfo, ao feito por Chaitin nos modelos iniciais da metabiologia: o **projeto inteligente** e a **evolução cumulativa**. Os **tempos de mutação** máximos, isto é, os limitantes superiores das quantidades médias de mutações necessárias, para fazer a aptidão dos organismos chegar à complexidade de $BB'(N)$ serão, respectivamente, os mesmos.

No primeiro capítulo vamos introduzir e explicar os conceitos e definições necessários para o entendimento da demonstração. E, logo depois, fornecer as ideias centrais da prova. Com isso em mente, vamos levar ao leitor as problemáticas que subjazem, permeiam e se desdobram dessa prova de evolução de subprogramas.

Já o segundo capítulo será todo dedicado à prova do primeiro. Começaremos explicando-a em palavras e depois apresentaremos a parte formal cabível que julgamos necessária: os lemas e teoremas.

Quando chegarmos ao terceiro capítulo, mudaremos de "espectro". Vamos passar a estudar de forma matemática e a discutir sobre a metabiologia de organismos não computáveis, de "**hiperorganismos**". Estes passarão a ser hiperprogramas, i.e., programas de hipercomputadores, capazes de "computar" problemas incomputáveis. A natureza metabiológica será um hipercomputador ainda muito mais incomputável. Se tomarmos como referência os **graus de Turing**, os organismos e mutações serão hiperprogramas de grau sempre finito, enquanto que a natureza será um hiperprograma de grau ω, o primeiro ordinal limite. Em relação aos modelos iniciais de Chaitin, essa prova fará uma versão da **busca exaustiva**. Contudo, o tempo de mutação, nesse caso, se apresentará como sendo muito mais rápido em comparação ao modelo chaitiniano.

Resta-nos pedir desculpas ao leitor por apresentarmos um texto carregado de termos teóricos e científicos e, principalmente, pela repetição dos mesmos. Para facilitar o entendimento, nos vimos obrigados a sacrificar o conforto da leitura fluente. Ademais, vamos introduzir notações novas, porém, sempre baseadas naquelas já costumeiramente utilizadas na literatura específica.

1 “EMULANDO” ORÁCULOS NA METABIOLOGIA

1.1 INTRODUÇÃO

Adiantando o objetivo desse capítulo, queremos construir subprogramas[11] que possam desempenhar o papel dos organismos que evoluem na metabiologia. Antes, estes eram programas quaisquer e a natureza era um hiperprograma de ordem 1. Porém, se quisermos que a natureza seja computável[12], precisamos que um programa possa desempenhar o papel dela, isto é, ser capaz de fazer a “seleção natural”. Para isso, os organismos não podem ser programas quaisquer, eles precisam ser programas que sempre param. Caso contrário, nenhum programa/natureza poderia saber sempre se um programa qualquer produz output ou nunca se detém. Assim, a “seleção natural” iria, em algum momento, ser incapaz de decidir se o ser vivo deve morrer ou viver, ficaria em *loop*, rodando *ad infinitum*. Somente um hiperprograma daria conta dessa tarefa sem ficar rodando de modo interminável.

Mas por que queremos uma natureza computável? Ou antes: o que quer dizer a natureza ser computável? Claro, estamos nos prendendo ao arcabouço teórico da metabiologia. Por isso, pelo menos em nossa primeira abordagem, queremos que ela seja um sistema qualquer[13] que desempenhe a “seleção natural”, como já explicado. Então, se antes ela precisava ser um hiperprograma, agora, queremos que ela seja apenas um programa. Isto é, que um computador seja capaz de emulá-la: representar todas as relações – do sistema – entre todas as representações de todos os elementos nele envolvidos (tanto os de comportamento quanto os de funcionamento).[14]

Vale assinalar que não podemos nos esquecer de que esse é um dos temas importantes ainda em discussão na literatura científica. Ser a natureza um sistema computável, ou não, passa também por saber se o mundo físico-químico é contínuo ou discreto, digital ou

---

[11] Definiremos o que é um subprograma adiante.
[12] Explicaremos o que isso quer dizer mais à frente.
[13] Real ou teórico, físico ou não.
[14] O que pode ser uma boa, e geral, definição de **emulação** sem que esteja presa ao conceito de computação. Geralmente, entende-se emulação como sendo **diferente** de **simulação**, pois nesta última nem todas as relações do sistema original são representáveis, limitando-se apenas a imitar o comportamento e não o funcionamento interno (o que a emulação pretende fazer também).

analógico. Mesmo assim, a vida poderia ser um caso à parte. Isto é, mesmo as relações físico-químicas sendo computáveis, o sistema físico-químico compondo o que chamamos[15] de "ser vivo" poderia apresentar propriedades emergentes que o tornaria incomputável (por exemplo, tal sistema poderia receber informações de um oráculo *à la* Turing, de acordo com outras leis físicas que ainda desconhecemos). De qualquer forma, entender o universo, desde a sua formação, como um processo que computa o estado presente, tomado seu estado inicial como input, parece atrativo e retumbante em nossas perspectivas teóricas sobre o atual estado do conhecimento científico. Discutiremos melhor na seção 1.4.

É nesse espírito que continuaremos o que já foi obtido por Chaitin, no intuito de tentar modelar uma evolução metabiológica dentro de uma natureza que seja um sistema computável, partindo do princípio de que a podemos representar por um programa de uma máquina universal de Turing. Porém, daí surgem alguns problemas iniciais. O primeiro deles, obviamente, é que os organismos precisam ser programas que sempre param – ou que exista um programa que sempre decida se os organismos param ou não – não poderão ser quaisquer programas, como já frisamos. Além disso, queremos manter uma forma de "medir"[16] a complexidade ou a **criatividade** dos organismos – usando o que chamamos de **função de aptidão**, que liga a aptidão à complexidade mínima que o organismo pode ter – no decorrer da evolução, como foi feito originalmente por Chaitin, usando a função *Busy-Beaver*[17]. E isso não é trivial quando lidamos com programas que sempre param – ou ainda, que tenham o tempo de cômputo[18] limitado por uma função computável. Aqui entrará o conceito de **subcomputação**.

Basicamente, chamaremos de subcomputador ou **submáquina de Turing** qualquer máquina de Turing que sempre dá um output para qualquer input, que sempre para. Um subcomputador deve se comportar em relação ao seu respectivo computador, o qual contém (ou está definido sobre) o primeiro, da mesma forma que um computador se comporta em relação a seu hipercomputador (uma máquina universal de Turing oracular de primeira ordem, ou seja, um computador com acesso a funções incomputáveis através de um "oráculo"). Na verdade, não vamos emular **todas**[19] as propriedades de um hipercomputador em relação a um

[15] Apesar de não sabermos definir bem.
[16] Isso será explicado mais abaixo.
[17] Na verdade, uma forma ligeiramente modificada desta.
[18] Ver item 2.15.4.
[19] O que pode ser impossível, inclusive.

computador, vamos somente nos focar em criar uma função análoga a *Busy-Beaver* (nos trabalhos de Chaitin, a função $BB'(N)$) de forma que ela se comporte em relação a um subcomputador da mesma forma que a *Busy-Beaver* original se comporta em relação a um computador. Em outras palavras, ela precisa ser incomputável, **relativamente**, por qualquer **subprograma** (um programa quando rodado por um subcomputador) da mesma forma que a *Busy-Beaver* original é incomputável por qualquer programa rodado por um computador. A esse fenômeno chamaremos de **incomputabilidade relativa recursiva ou subincomputabilidade**. Lembre que ela não pode ser "muito" incomputável, pois uma máquina oracular de primeira ordem já é capaz de computar a $BB'(N)$. E nós queremos que a nova função *Busy-Beaver* seja tão, e não mais, incomputável para um subcomputador quanto a *Busy-Beaver* o é para um computador. Isso será de suma importância para construir nosso modelo de evolução análogo ao de Chaitin e, talvez, será a parte teórico-matemática mais inovadora e laboriosa.[20] Vamos simbolizar esta nova função de aptidão por $BB^{+}{}_{P^{**}{}_{T}\circ P_T}(N)$.

Assim, pretendemos mostrar que existe uma evolução de subprogramas[21], numa natureza computável, análoga à dos três modelos (*busca exaustiva, projeto inteligente* e *evolução cumulativa*) iniciais construídos pelo nosso principal autor de referência. Ou seja, mantendo o mesmo **tempo de mutação** (a quantidade média de mutações necessárias para se atingir uma aptidão $BB'(N)$ ou, no nosso caso, $BB^{+}{}_{P^{**}{}_{T}\circ P_T}(N)$), os mesmos tipos de organismos e os mesmos tipos de mutações. Para o primeiro modelo, o tempo de mutação é da ordem de 2^N, ou seja, exponencial. Para o segundo, o tempo é da ordem de N, ou seja, linear. Para o terceiro, e o mais interessante do ponto de vista do neodarwinismo, é da ordem de N^2, ou seja, praticamente quadrático. Queremos manter esses resultados também no presente trabalho, evidenciando ainda mais o fenômeno da incomputabilidade relativa.

É preciso dizer que apenas construiremos um correspondente ao *projeto inteligente* e um correspondente à *evolução cumulativa*.[22] Para a *busca exaustiva,* deixaremos como

[20] Ou seja, mostrar que a função $BB^{+}{}_{P_T}(N)$, para qualquer P_T, é só relativamente incomputável em relação a U_{P_T} é bastante simples e intuitivo. Ver seção 2.3.
[21] As mutações também serão subprogramas.
[22] Usaremos, para os dois casos, o mesmo subcomputador. O que muda é a forma como as mutações/subprogramas são geradas: aleatoriamente ou não.

exercício para o leitor, visto que ela pode ser facilmente construída tomando como base um subcomputador muito parecido[23] ao construído para os dois outros modelos.

Outro ponto importante é a questão experimental. O que nos afeta mais na prática é o fato de não podermos simular ou emular os modelos metabiológicos de evolução numa natureza oracular com nossos computadores, pois precisaríamos de um hipercomputador para fazê-lo. Artefato que não temos até o momento – e nem sabemos se ele de fato é possível. Já com uma natureza computável, isso seria bastante viável. Basta, no caso da *evolução cumulativa*, termos uma fonte quase perfeitamente aleatória para gerar *bit strings*, as mutações. Tudo feito até aqui tem esse caráter de poder ser programado.[24] Se alguém, no futuro, quiser explorar as consequências, na **computação experimental**, dos resultados apresentados neste trabalho, isso será possível.

A maior parte desta tese se concentra em definir um subcomputador para que ele tenha as propriedades (i.e., que possamos demonstrar os teoremas análogos de que tanto falamos) desejadas. O resultado/teorema principal é mostrar que esse subcomputador é, de fato, um subcomputador e que a definição não é vácua ou, ainda, que ele sempre se detém. Depois, demonstraremos que a evolução dos subprogramas funciona da maneira que queremos.

Começaremos discutindo as ideias, tanto as que inspiraram o trabalho quanto as da prova em si. Aqui, cabe um comentário: a **incomputabilidade relativa** é um conceito que já brotava de nossos estudos oriundos de outros temas relacionados à teoria da computação. O mote principal, já mencionado em parágrafos anteriores, seria construir, ou provar que existe, um computador que possa se "comportar" em relação a um subsistema seu (um subcomputador) da mesma forma que um hipercomputador (uma máquina oracular) se comporta em relação a um subsistema seu (em particular, um computador como conhecemos). Portanto, o presente trabalho vem, fortuitamente ou não, a dar um primeiro resultado positivo para isso: construímos uma natureza/programa que "finge" ser uma natureza/hiperprograma, de forma que a evolução dos seres vivos/subprogramas pode ocorrer como ocorre no caso dos seres vivos/programas. Para tal, faremos duas provas[25] isomorfas às feitas originalmente por Chaitin. E, caso exista algum problema técnico – ou nas demonstrações –, acreditamos (ou

[23] Que possa rodar as mutações de tamanho da ordem de N, as quais levam o organismo diretamente para aptidão $BB^{+}{}_{P^{**}{}_{T}{}^{\circ}P_{T}}(N)$.
[24] Inclusive a linguagem de programação que usaremos, mesmo não sendo usual, será definida para que possa ser programada.
[25] A do *modelo cumulativo* e a do *projeto inteligente.*

apostamos) que essas ideias são fortes, profícuas e que algum resultado equivalente pode ser mostrado por alguém no futuro.

Isto posto, vamos explicar as definições e as provas da forma mais discursiva possível. No item 1.3 falaremos das ideias gerais da demonstração – o "espírito" da prova – e no capítulo 2, as explicaremos. Na seção 2.15, deixaremos os lemas e teoremas formais caso o leitor queira se aprofundar na matemática envolvida.[26]

1.2 CONCEITOS-CHAVE

Entender o que chamamos por *organismo* é, talvez, a parte central do tema deste texto. A própria metabiologia, como o nome indica, tem como base a evolução dos seres vivos. No entanto, quando passamos para o plano teórico e abstrato da matemática, essa nomeação assume um conceito um pouco diferente. Na metabiologia, o organismo ou ser vivo é composto por informação, informação algorítmica[27]. Em geral, utilizamos bits para quantificar essa informação, fazendo um organismo metabiológico nada mais ser que uma *string*[28] de 0´s e 1´s. Além disso, essa *bit string* não é estática, justamente por ela ser algorítmica, assim como um ser vivo não é um sistema estático, pois este está sempre mudando, está em "movimento", é sempre um processo. Essa *bit string* é também algo que desencadeia um processo, no caso, um processo computacional (um cômputo). Porém, temos que dizer que o organismo metabiológico é "descorporificado"[29] ou, em outras palavras, que é um sistema matemático de relações, independente do que ele está representando – se é que necessita representar algo.[30] Portanto, usaremos a nomenclatura organismo/programa quando quisermos frisar bem que não estamos falando do organismo biológico.

Um subprograma – explicaremos melhor a seguir – é um tipo de programa. Vale dizer que quando falarmos de organismo/subprograma ainda estaremos no campo da metabiologia e não da biologia; apenas o estamos diferenciando de um programa qualquer.

---

[26] Colocamos apenas as demonstrações formais dos teoremas que julgamos mais importantes ou que a explicação discursiva não tenha dado conta. Claro, o leitor é convidado a verificar se todos os passos estão corretos.
[27] Que é diferente da informação estatística, shannoniana.
[28] Uma sequência em que um símbolo é posto ao lado do outro, um em cada casa (o bit).
[29] CHAITIN, V. O corpo metabiológico. **Scientiarum Historia IV**, Rio de Janeiro, 2011.
[30] Ver seção 1.2.3.

De forma análoga, definimos o que é a mutação/programa ou mutação/subprograma. Como em Chaitin, a mutação, aqui, também é um processo algorítmico que transforma o organismo anterior, gerando um novo organismo como seu output. Mas, agora, é um subprograma e não um programa qualquer.

Também se pode notar que tomamos emprestado o conceito de "natureza biológica", aquilo que abriga, engloba e contém a vida – no caso, até onde conhecemos, a vida na Terra. A natureza metabiológica também é um sistema matemático, no caso, um sistema de informação algorítmica, uma *bit string* a ser processada por um computador ou hipercomputador, o qual poderia ser, por exemplo, o Universo ou as leis da Física.[31] Qualquer organismo deve ser um subsistema dela, de forma que sua vida, do começo ao fim, possa ser processada (computada) dentro da mesma (ou pela mesma) - por exemplo, um aplicativo rodando dentro de um sistema operacional. Isso, inclusive, nos dá uma definição bem geral do conceito de **subcomputação**. Ou seja, um subcomputador nada mais é que um subsistema que pode ser **inteiramente (do começo ao fim)**[32] computado dentro de outro computador.

Agora, já podemos falar da diferença entre uma natureza computável e uma natureza oracular, incomputável. Mas o que é um oráculo? É um "mecanismo" abstrato, matemático, idealizado para ajudar um computador, uma máquina de Turing, a computar os problemas que este não poderia computar de forma alguma.[33] Um exemplo rápido é imaginar uma caixa-preta que um computador pode consultar para saber se um programa se detém ou não. Com isso, ele pode resolver toda uma gama de problemas que seriam insolúveis sem essa "caixa-preta". [34]

Ao contrário dos primeiros modelos feitos por Chaitin, almejamos uma natureza que não tenha acesso a oráculos. Pelo menos, que não tenha acesso a oráculos "verdadeiros", como explicaremos no próximo tópico. Ele precisa ser um sistema que possa ser emulado, por completo, por um computador. Consequentemente, dizemos que queremos uma natureza computável. No fundo, nossos esforços não serão para eliminar a necessidade de oráculos. Eles ainda estarão, de forma indireta, presentes, porém, serão "oráculos fingidos", que se

[31] Caso este seja passível de ser comparado a um computador ou a um hipercomputador.
[32] Esse advérbio é importante para diferenciar da simples **emulação**. Na subcomputação, qualquer subprograma produz um output, mesmo que este só seja sabido pelo computador que contém o tal subcomputador. Por outro lado, um programa pode emular outro que nunca se detém e o primeiro nunca sabe que ele nunca para.
[33] Para mais detalhes sobre hipercomputação, ver capítulo 3.
[34] Ver seção 3.3 sobre a nossa definição de hipercomputador, os quais usam fitas extras de uma máquina de Turing como oráculos.

passam por oráculos verdadeiros em relação a qualquer subsistema da natureza – agora, computável. No caso, nossos teoremas mostrarão uma evolução que se passa **como se** a natureza fosse oracular.

O papel-chave que a natureza tem que desempenhar nas demonstrações da metabiologia até o presente momento, como já mencionado na introdução, é o de fazer a "seleção natural", que nada mais é do que decidir se a aptidão do novo organismo é maior que a do anterior. Caso não seja, ela descarta o novo organismo e permanece com o anterior. Isso, diga-se de passagem, é apenas uma imitação rudimentar da estudada seleção natural do mundo biológico. De fato, poderíamos definir o que seja a natureza metabiológica de uma maneira muito mais geral que apenas pela seleção natural (afinal, a natureza biológica não faz somente isso). Mas esse não é o foco do presente trabalho.

1.2.1 **A linguagem *L***

A primeira definição importante que precisamos estabelecer é a própria linguagem de programação ou de máquina universal com a qual iremos trabalhar. O resultado apresentado a seguir não pretende mostrar que o fenômeno da subcomputação, ou mesmo a evolução metabiológica de subprogramas,[35] pode acontecer em qualquer linguagem em qualquer computador, mas sim mostrar que existe uma linguagem (ou uma classe delas) na qual os resultados vindouros são verdadeiros. É importante para nós que o modelo possa ser programado e não que ele seja totalmente ubíquo.[36] Além disso, essa linguagem pode ser empregada em qualquer computador usual, isto é, suas propriedades e regras de bem formação são programáveis. No fim, isso nos levará à conclusão de que o fenômeno da subcomputação pode ocorrer "dentro" de computadores que já conhecemos.

Precisamos de uma linguagem, isto é, um subconjunto de *bit strings* com o qual um programa pode decidir sempre se uma *bit string* é fruto de uma concatenação especial ou não. Concatenar é colocar duas *bit strings* uma ao lado da outra, em ordem, formando uma nova *bit string*. Enquanto essa operação é trivial numa linguagem qualquer, quando estamos trabalhando com *bit strings* autodelimitada, simplesmente juntar dois elementos pode gerar outro elemento malformado, no caso, não autodelimitado. No fundo, o problema que se

[35] Nesta, inclusive, precisamos trabalhar numa linguagem de programas autodelimitados.
[36] Apesar de o fenômeno da incomputabilidade relativa ser ubíquo.

coloca é lidar com as regras de bem formação enquanto fazemos essa concatenação, de forma que a mesma seja um procedimento sujeito a decisão (ou computável). Assim, existirá um programa que conseguirá lidar com qualquer elemento da linguagem desejada, sabendo sempre se ele está bem formado e se é fruto de concatenações.

Mas por que concatenações? Elas nos dão uma forma direta de simbolizar um programa tomando uma *bit string* qualquer como seu input – por exemplo, um programa p que é, na verdade, o programa p' quando este toma o programa p'' como seu input. Note-se que esse tipo de programa já é usado para demonstrar o *problema da parada* ou que a função *Busy-Beaver* não é computável. Porém, a forma que ele pode ter é completamente arbitrária, pois uma máquina universal de Turing o roda, de qualquer maneira. Por isso, não é à toa que precisamos dessa condição – essa concatenação especial – em nossa linguagem. Como estamos querendo construir um computador que consiga **emular** a incomputabilidade, se tornará necessário que possamos "ensinar" uma máquina a fazer e a reconhecer essas "concatenações" dentro da linguagem que ela própria está trabalhando.

Para diferenciar da concatenação comum, que é só juntar *strings*, vamos denotar essa nova operação por "$\circ$". Vamos simbolizar a concatenação comum por "$*$".[37] Quando colocarmos a palavra concatenação entre aspas, estaremos querendo dizer "$\circ$", do contrário, queremos dizer "$*$".

Antes de definir tudo, vamos dar um exemplo trivial que satisfaz nosso objetivo. O que, por sinal, já serve para nos mostrar que a definição não é vácua. Além de nos dar uma forte intuição sobre o que queremos com as propriedades que vamos enunciar para definir uma linguagem L.

1.2.1.1 Exemplo:

Tome uma máquina universal de Turing U' numa linguagem trinária (com os símbolos 0, 1 e h) qualquer.[38] Vamos, agora, autodelimitá-la. Dada uma *bit string* u, basta colocar na frente um programa que nomeia seu tamanho e, na frente deste, o tamanho do segundo em

[37] Atenção para não confundir nos casos em que "$*$" é usado para denotar multiplicação.
[38] O símbolo h denota a parada, para U saber quando termina a *string*.

quantidade de zeros e, então, depois o algarismo 1 para marcar que terminou a primeira parte. Assim, teremos $0^{|(|u|)_2|} * 1 * (|u|)_2 * u$ como sendo nossa nova *string*.[39] Essa é uma das formas triviais de tornar tudo autodelimitado.

Agora, vamos adicionar um prefixo na frente indicando quantas "concatenações" (o que irá denotar quantos inputs o programa está tomando, obviamente). No fundo, esse prefixo dirá quantos programas se tem que ler. Pegue o número de *bit strings* concatenadas na forma binária e o coloque na forma autodelimitada acima. Então, coloque essa *bit string* resultante como prefixo do programa concatenado com seus inputs todos. Todos estes já na forma autodelimitada acima.

Por exemplo, se quiser colocar o programa 01 com 101 e 1 como seus inputs, na nossa linguagem ele ficará da forma $(001 * 10 * 11) * (001 * 10 * 01) * (001 * 11 * 101) * (01 * 1 * 1)$.[40] Logo, quando essa *bit string* aparecer para a U – que é a nova máquina universal de Turing trabalhando sobre a nova linguagem e sobre U' –, ela irá rodar em U' o resultado de se pegar o programa 01 e dar a ele 101 e 1 como inputs, isto é, irá rodar $01 * 101 * 1 * h$.

Lembre que 01, 101 e 1 também são, individualmente programas de U'. Por conseguinte, $(01 * 1) * (001 * 10 * 01)$, $(01 * 1) * (001 * 11 * 101)$ e $(01 * 1) * (01 * 1 * 1)$ também serão programas de U. Se olharmos tudo isso, em nossa notação teremos que: $((01 * 1) * (001 * 10 * 01)) \circ ((01 * 1) * (001 * 11 * 101)) \circ ((01 * 1) * (01 * 1 * 1))$ é igual a $(001 * 10 * 11) * (001 * 10 * 01) * (001 * 11 * 101) * (01 * 1 * 1)$.[41]

Note, em primeiro lugar, que todo esse procedimento é computável – o que é uma das propriedades mais importantes da nossa linguagem L. A isso, chamaremos de **recursivamente funcionalizável**. Além disso, decidir se uma *bit string* está nesse formato bem específico é sempre possível. A isso dizemos ser uma linguagem **recursiva**, isto é, quando uma máquina sempre pode decidir se qualquer *bit string* pertence ou não à linguagem.

É fácil provar também que para todo programa P, w_1, ... e w_k de L, $|w_i| < |P \circ w_1 \circ ... \circ w_k|$. "Concatenar" sempre aumenta o tamanho.[42] Outra ponto importante é notar

---

[39] $(x)_2$ denota o número x na forma binária.
[40] Os parênteses são apenas para o leitor se situar.
[41] Como U está definido em relação a U', teremos que $U\big((01 * 1) * 0001 * 110 * (01 * 101 * 1)\big) = U(((01 * 1) * (001 * 10 * 01)) \circ ((01 * 1) * (001 * 11 * 101)) \circ ((01 * 1) * (01 * 1 * 1)))$.
[42] O que será vital no teorema 2.12, um dos teoremas centrais.

que, para todo programa P, w_1, ... e w_k de L, $|P \circ w_1 \circ ... \circ w_k| \leq \log_2(\log_2(k+1)) + 1 + \log_2(\log_2(k+1)) + \log_2(k+1) + |P| + |w_1| + |w_2| + \cdots + |w_k|$.[43] Logo, existe constante C tal que $|P \circ w_1 \circ ... \circ w_k| \leq C \times k + |P| + |w_1| + |w_2| + \cdots + |w_k|$.[44] Por último, qual seria o menor programa na linguagem L que nomeia um número natural N?[45] Na forma binária ele teria tamanho praticamente $\log_2(N)$. Então, autodelimitando-o e informando a quantidade de "concatenações", ele fica com um tamanho igual a $1 + 1 + 1 + \log_2(\log_2(N)) + 1 + \log_2(\log_2(N)) + \log_2(N)$. Logo, podemos dizer também que existem constantes C' e ϵ tal que $H(N) \leq C' + \log_2 N + (1+\epsilon)\log_2(\log_2 N)$.[46]

No entanto, a soma de todas as probabilidades de todos os programas precisa ser ≤ 1. Pois, do contrário, não definiríamos uma probabilidade. Além disso, como tudo que nos interessa são os elementos dessa linguagem, queremos que a soma seja exatamente 1. Esse nosso exemplo dá conta disso? Basta lembrar que o somatório de todas as probabilidades de programas autodelimitados é sempre menor ou igual a 1. No nosso caso, o somatório das probabilidades ficará dado pela expressão:

$$\frac{2}{2^4} + \frac{2^2}{2^7} + \frac{2^3}{2^8} + \frac{2^4}{2^{11}} + \frac{2^5}{2^{12}} + \cdots + \frac{2^k}{2^{k+2\,log_2(k)+1}} + \cdots = \frac{1}{2^2}\left(\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots\right) = \frac{1}{2^2}(1) = \frac{1}{2^2}$$

Como esse somatório não dá exatamente a unidade, sabemos que existe uma **redundância** na autodelimitação. Quando essa soma dá exatamente a unidade, dizemos que essa é uma codificação – ou uma linguagem – **completa.** Para o que queremos, precisamos que ela seja **completa**. Queremos que nosso espaço de possibilidades ou espaço amostral seja constituído apenas por programas válidos. Como fazemos? Temos duas formas simples: basta, então, multiplicar por 2^2 a probabilidade de cada programa, ou adicionar um programa

[43] Na verdade, pegamos o primeiro inteiro maior que o valor desse logaritmo. Dessa forma, calculamos o tamanho do número binário que corresponde ao número natural sendo logaritmizado. Portanto, $\log_2(0) = 0 + 1$, $\log_2(2) = 1 + 1$, $\log_2(7) = 2.807 ... = 3$ e assim por diante.
[44] O que é de suma importância para demonstrar a incomputabilidade relativa e a evolução cumulativa. Ver teorema 2.15.8 e 2.14.
[45] O que também é de suma importância para demonstrar a incomputabilidade relativa e a evolução cumulativa.
[46] Em particular, no nosso exemplo, $\epsilon = 1$.

de tamanho 1 e um de tamanho de 2 que não computam nada, para fazer o somatório de tudo dar exatamente 1.

E mais: sabemos que o somatório de todas as probabilidades de programas autodelimitados concatenados também dá menor ou igual a unidade. Logo, como a parte que indica o número de "concatenações" também é autodelimitada e completa, o somatório final ainda permanecerá resultando em 1. O que faz de L uma linguagem completa.

1.2.1.2 Definição:

Dizemos que uma linguagem de programação, definida sobre uma máquina de Turing universal U, é **recursivamente funcionalizável** se existir um programa tal que, dada quaisquer *bit strings* P e w como inputs, ele retorna uma *bit string* que vamos denotar por $P \circ w$, com a qual $U(P \circ w)$ é igual ao resultado da computação do programa P quando w é dado como seu input. Além disso, tem que existir um programa que decida se uma *bit string* é da forma $P \circ w$, quaisquer que sejam P e w. De modo análogo, o mesmo tem que valer para a concatenação genérica $P \circ w_1 \circ \ldots \circ w_k$, com o programa P recebendo $w_1, \ldots$ e $w_k$como inputs.

Assim, já podemos enunciar as definições gerais da linguagem L. E **definir a linguagem L e, por tabela, a máquina universal U**: seja U uma máquina de Turing universal sobre uma linguagem L **binária**, **autodelimitada, recursiva**, **recursivamente funcionalizável** e tal que existem constantes $\in$, C e C', para todo P e $w_1, \ldots$ e w_k, onde.

$$|w_i| < |P \circ w_1 \circ \ldots \circ w_k|, para\ i = 1,2, \ldots ou\ k$$

e

$$|P \circ w_1 \circ \ldots \circ w_k| \leq C \times k + |P| + |w_1| + |w_2| + \cdots + |w_k|$$

e

$$H(N) \leq C' + \log_2 N + (1 + \epsilon) \log_2 (\log_2 N)$$

Além disso, ela deve ser uma linguagem **completa**, isto é, a probabilidade de ocorrer um programa p qualquer deve ser igual a 1.[47] Em outras palavras,

$$\sum_{p \in L} 2^{-|p|} = 1$$

1.2.2 **Submáquinas de Turing ou Subcomputadores**

Definiremos o conceito chave para entender todo este trabalho. Se queremos criar um subsistema dentro de outro sistema – este último, já computável –, que tipo de subsistema será esse? Obviamente, tudo que o subsistema puder fazer, o sistema tem que poder fazer também. Aquele que contém o outro deve sempre ser mais poderoso, no sentido de poder de cômputo pelo menos.

A ideia é simples: o subsistema pode fazer tudo que o sistema pode, porém, com recursos limitados pelo próprio sistema. Por sistema pode-se entender um computador e por subsistema pode-se entender um subcomputador. Por exemplo, um subcomputador pode ser um programa ou uma sub-rotina que o computador roda, sempre gerando um output, enquanto executa várias outras tarefas.

Portanto, usaremos outra noção de vital importância: o **tempo de cômputo**. Dada uma máquina universal U, existe um programa que emula completamente ela mesma.[48] Com isso, podemos construir um programa que verifica cada ação (movimento da cabeça de leitura, troca de símbolos na fita, etc.)[49] realizada quando ela está rodando um programa qualquer. Vamos chamar de T esse programa que calcula quantos passos ou operações básicas U realiza quando está rodando o programa p. Logo, se $U(p)$ não parar, então $U(T * p)$ também não para. E vice-versa.

---

[47] Na verdade, essa condição não é estritamente necessária para chegarmos onde queremos. De todo modo, ela serve para simplificar a demonstração do teorema 2.15.13.
[48] Uma das propriedades centrais das máquinas universais de Turing.
[49] Aqui, faz-se necessário entender a definição de máquina de Turing, o que é central na teoria da computação.

Seja P_f um programa de U, definida sobre uma linguagem L, que computa uma função total f tal que $f: L \longrightarrow X \subseteq W$. A linguagem W não precisa necessariamente ser autodelimitada. Podendo ser todas as *bit strings* de tamanho finito de modo que possamos recursivamente enumerá-las em ordem por l_1, l_2, l_3, ... Por questões práticas, vamos escolher uma enumeração onde $l_1 = 0$.

Definimos o subcomputador (ou uma submáquina de Turing) $U/_f$ como sendo a máquina de Turing onde, para toda *bit string* w na linguagem de U, $U/_f\,(w) = U(P_f \circ w)$.

Essa definição é bem geral e transforma qualquer função total computável em uma submáquina. Portanto, a classe de todas as submáquinas é infinita, enumerável, mas não é recursiva.[50] Quando falarmos de subprogramas estaremos nos referindo a programas rodados por uma submáquina de Turing.

Aqui, vamos nos ater em uma subclasse de submáquina em particular: as submáquinas definidas por tempo de cômputo limitado. Na verdade, esta e as submáquinas mais genéricas, definidas antes, são quase equivalentes. Para mostrar isso, basta notar que se um programa computa uma função total, então existe um programa que computa o tempo de cômputo desse primeiro programa. Portanto, para qualquer função computável e total, existe uma submáquina com tempo de cômputo limitado capaz de computar essa função – e, possivelmente, outras funções também.

Seja P_T um programa arbitrário que calcula o tempo máximo de cômputo para um programa w qualquer. Ou seja, P_T pode ser uma função total qualquer. Então, existe uma submáquina de Turing definida pela função de tempo de cômputo P_T. Ou seja, construímos a submáquina $U/_{P_{SM} \circ P_T}$ (que será um programa de U que computa uma função total) onde um P_{SM} é um programa que recebe P_T e w como inputs, roda $U(P_T \circ w)$ e retorna:

(i) l_1, se $U(w)$ não parar em tempo de cômputo $\leq U(P_T \circ w)$;

(ii) l_{k+1}, se $U(w)$ parar em tempo de cômputo $\leq U(P_T \circ w)$ e $U(w) = l_k$;

---

[50] DORIA, F. A.; CARNIELLI, W. A. Are the Foundations of Computer Science Logic–Dependent? **Dialogues, Logics and Other Strange Things**, 20 Outubro 2008.

Esse programa define um subcomputador (uma submáquina de Turing) que nos retorna um símbolo conhecido (no caso, zero) quando um programa *w* não parar em tempo $\leq U(P_T \circ w)$ ou retorna o mesmo output (a menos de uma bijeção trivial) de $U(w)$ quando este parar em tempo $\leq U(P_T \circ w)$.

Para ser uma submáquina de Turing, ${}^{U}/_{P_{SM} \circ P_T}$ precisa estar definida para todos os inputs. Isso ocorre pelo fato de P_T ser total, por suposição.

Visto isso, vamos denotar somente por U_{P_T} a submáquina de Turing ${}^{U}/_{P_{SM} \circ P_T}$, tal que:

$$\forall w(\, U_{P_T}(w) = {}^{U}/_{P_{SM} \circ P_T}\,(w) = U(P_{SM} \circ P_T \circ w))$$

Além disso, dizemos que uma função ou um número é P_T-computável se ele for computável pelo computador U_{P_T}. Um 0-programa é um programa quando rodado por U. Um P_T-programa é um programa quando rodado por U_{P_T}. Se P_T for total, então todo P_T-programa é um subprograma de U_{P_T}, o qual é uma submáquina de Turing de U.

1.2.3 **Organismos, Complexidade e Aptidão**

Antes de apresentarmos as ideias da demonstração, vamos voltar a alguns conceitos já utilizados por Chaitin em seus artigos sobre metabiologia. De qualquer forma, tudo isso será discutido melhor mais adiante neste trabalho.

Assim como os programas são os organismos ou seres vivos na metabiologia, nossos organismos continuarão a ser programas, porém, programas de uma submáquina ou um subcomputador, ou seja, subprogramas. Pode-se pensar esse subcomputador como uma sub-rotina da natureza, um dos muitos programas que ela é capaz de rodar. É bom lembrar que a natureza, aqui, é um programa mais poderoso que o subcomputador. Nos modelos originais de Chaitin, a natureza era uma máquina de Turing oracular – um hiperprograma - e, portanto, também muito mais poderosa que a máquina de Turing que rodava os programas/organismos.

Não à toa fazemos tais analogias. Quase que por definição do nosso conceito de Natureza – real, do mundo físico que conhecemos e que carrega a vida dentro de si –, somos

inclinados a entender os seres vivos em geral como subsistemas de um sistema muito maior, a própria natureza. Por mais complexo que seja o sistema físico-químico que suporta a vida, ele "roda" dentro das regras da natureza, a qual detém todas essas "regras".[51] Isso nos permite dizer que a natureza "roda" os seres vivos.

Caso a natureza físico-químico-biológica que admiramos e procuramos entender seja um sistema computável (passível de compreensão humana ou não), essa analogia se torna direta e inequívoca.

Não iremos discutir o que se pode entender por aptidão nessa parte do trabalho, nem como isso, de fato, ocorre na biologia. A ideia básica, aqui, é a de poder comparar organismos quanto aos seus desempenhos ou ao seu "poder de sobrevivência"[52] frente ao meio ambiente que os contém e que os cerca. Daí que quanto maior o número que um organismo/programa retorna (isto é, gera como output), maior será sua aptidão. Novamente, como dito na Introdução, essa interpretação pode ser muito simplória para descrever o que ocorre na biologia. Adaptação pode não estar ligada a maior aptidão absoluta, que independe do "entorno". Um organismo é apto, menos apto ou mais apto dependendo do meio e das circunstâncias que ele está inserido. Além disso, esses dois fatores podem mudar constantemente com o tempo. Porém, tantos os modelos chaitinianos como os desta tese somente consideram um organismo por vez sendo mutado. Não há ecossistema nem comunidades de seres vivos.[53] Pode-se considerar, então, que também modelamos um só meio ou "entorno" constante durante a evolução dos seres metavivos solitários.

Por exemplo, esse tipo de discussão pode ser parcialmente evitada se considerarmos os seres vivos como sistemas e a aptidão metabiológica como uma medida de quão complexo ou complicado é o ser vivo em funcionamento. Ou seja, se seu "entorno" exigir do organismo uma adaptabilidade que seja mais "complicada" ou complexa, ele precisará ser, para sobreviver ou estar adaptado, um sistema tão ou mais complexo que o "entorno" exige - sob esse ponto de vista, não faz muito sentido em falar do mais apto, apenas do apto, do adaptado ao meio. Basta perceber que as relações de um organismo com seu meio também tem que ser representadas pelo conjunto de todas as relações que compõe o organismo em funcionamento,

---

[51] Independentemente dessas "regras" serem definíveis ou não. Para a analogia funcionar não importa se a Natureza é, em última instância, indecifrável.
[52] Estamos evitando falar de adaptabilidade, nos atemos apenas ao aspecto geral e essencial.
[53] Pelo menos, não de maneira direta.

o organismo vivo. Afinal, seu funcionamento, ou sua mudança de estados, inclui todas as formas, ou relações, que o meio interage com ele.

Toda essa discussão também é análoga se os seres vivos forem sistemas hipercomputáveis, como no capítulo 3, ou subcomputáveis [54], como neste capítulo. Infelizmente, um tratamento mais profundo desse ponto de vista com a biologia, entre adaptação e evolução, é desejável, mas fugiria do nosso objetivo nesta tese por ser amplo e controverso.

Com isso em mãos, ficou fácil para a metabiologia tomar de empréstimo o conceito de complexidade algorítmica ou *program-size complexity* da teoria da informação algorítmica (TIA) para usarmos como "medida" de complexidade ou de criatividade dos organismos/programas. Um organismo/programa tem complexidade maior ou igual a N se nenhum outro programa de tamanho menor que N consegue calcular uma aptidão maior ou igual à aptidão que esse primeiro organismo consegue calcular. É aqui que entra a função *Busy-Beaver*. No nosso caso, a função *Busy-Beaver Plus*, simbolizada por $BB^{+}{}_{P'_T}(N)$. A forma canônica de definir a complexidade algorítmica de um programa P é pelo tamanho do menor programa que retorna como output o programa P. A metabiologia usa essa "conversa" e equivalência, proveniente da teoria da informação algorítmica, entre a função *Busy-Beaver* e a *program-size complexity*.

Enfim, qual a ideia por trás da função $BB^{+}{}_{P'_T}(N)$? Ela pega um valor N e leva num outro número, grande o suficiente para que nenhum subprograma (i.e., nenhum programa de $U_{P'_T}$) seja capaz de calculá-lo. Isso vale seja qual for o programa P'_T. Ela é definida para ter, em relação aos subprogramas, o mesmo papel que a *Busy-Beaver* tem em relação aos programas.

Portanto, assim como a função BB' utilizada por Chaitin, essa nossa nova função também "mede" a complexidade dos organismos/subprogramas, assim como a primeira "mede" a complexidade dos organismos/programas. Vale assinalar que um organismo com aptidão $BB^{+}{}_{P'_T}(N)$ é incompressível a qualquer outro com tamanho menor que N. Isso quer dizer que nenhum subprograma com tamanho menor que N consegue fazer o que ele consegue. E está, inclusive, consonante com a ideia da biologia de que um organismo mais

[54] O caso subcomputável é interessante porque a própria enumeração entre números inteiros e menores subprogramas emuladores também será recursiva ou computável.

complexo[55] não se resume ao funcionamento de seus subsistemas, órgãos ou células. Quem sabe também não tenha a ver com a ideia de complexidade irreduzível de Michael Behe[56]?

De qualquer forma, podemos tirar diretamente daí a ideia de **incomputabilidade**, só que, agora, **relativa** à submáquina $U_{P'_T}$ ao invés de U: ou seja, não terá como um subprograma de $U_{P'_T}$ calcular todo valor de $BB^+{}_{P'_T}(N)$ dado N como seu input.

A inter-relação entre complexidade, criatividade e a função de aptidão é característica fundamental já contida nos modelos de Chaitin e que queremos manter em nossa evolução com uma natureza computável, não oracular. Por isso, adotaremos a nova função $BB^+{}_{P'_T}(N)$ como nossa **função de aptidão.**

1.3 AS IDEIAS CENTRAIS DA PROVA

O cerne do argumento é o que podemos chamar de **"laço autorreferencial"**. Antes de explicá-lo, voltemos ao tão conhecido **método diagonal**. Como já mencionado antes, podemos dizer que se trata de um argumento sobre o qual construímos um procedimento ou uma função que, caso ela se refira a si mesma de alguma forma, obrigatoriamente produzirá um resultado diferente do que se tinha antes, gerando uma contradição. Pelo argumento diagonal, provamos que um objeto matemático só estaria bem definido se ele já estivesse bem definido antes, e caso esse último já estivesse definido antes, e assim por diante. Uma função, por exemplo, para estar bem definida num valor qualquer precisaria "chamar a si mesma" e, logo, já precisaria estar bem definida para outro valor. Daí, a contradição.

No entanto, esse processo de 'diagonalização' é essencial para a função *Busy-Beaver Plus* ($BB^+{}_{P'_T}(N)$). Ela sempre "olha" para os valores dados em função da submáquina $U_{P'_T}$ para qualquer subprograma de tamanho $\leq N$ e retorna um valor que não poderia ter sido gerado por nenhum desses subprogramas. Contudo, queremos achar uma submáquina que eventualmente, para todo N, ela consiga calcular/computar $BB^+{}_{P'_T}(N)$, mesmo que seja com

[55] Mesmo que o adjetivo "mais complexo" não esteja bem definido entre os biólogos e filósofos da biologia, dentro da nossa definição ele está.
[56] ABDALLA, M. **La Crisis Latente del Darwinismo**. Sevilla: Publidisa, S.A., 2010. ISBN ISBN 978-84-937871-1-0.

um subprograma bem grande. Isso nos leva diretamente à 'autodiagonalização', pois, esse subprograma precisa "chamar" o próprio programa que computa a submáquina em que ele está rodando e, mesmo assim, retornar um valor, sempre.

Como poderíamos, então, "ensinar" ou programar uma submáquina para se autodiagonalizar sem que seus cômputos entrem num "loop" sem fim? Sem que esse cômputo sempre altere o valor de uma função, visto que o valor desta antes era outro? O "truque" que encontramos é bem simples: **qualquer subprograma para estar bem definido só precisa que os subprogramas de tamanho menor estejam bem definidos.** Um subprograma nunca vai depender do output de si mesmo – e nem de subprogramas de tamanho maior – para poder gerar um output. É importante alertar que, sem ter isso em mente, o leitor pode facilmente se perder ou "entrar em parafuso" quando for analisar a demonstração a fundo, percorrendo as autorreferências que os programas que iremos definir fazem.

Assim, conseguimos que essa submáquina $U_{P'_T}$ ou subcomputador autodiagonalizante consiga computar, com programas suficientemente grandes, os valores de $BB^+{}_{P'_T}(N)$, função que já necessita de uma "diagonalização", em primeiro lugar, para estar definida.

Ok, mas e o "laço autorreferencial" que mencionamos no início deste subtópico? A submáquina de Turing que iremos construir para satisfazer nossos anseios será denotada por $U_{P^{**}{}_T \circ P_T}$. Numa "ponta do cadarço" temos a submáquina $U_{P^*{}_T \circ P'{}_T \circ P_T}$ que depende dos programas P'_T e P_T para ser total e, além disso, $U_{P^{**}{}_T \circ P_T}$ é igual a $U_{P^*{}_T \circ P'{}_T \circ P_T}$ quando $P^{**}{}_T \circ P_T$ é colocado no lugar de P'_T, i.e., $U_{P^{**}{}_T \circ P_T}$ é igual a $U_{P^*{}_T \circ P^{**}{}_T \circ P_T \circ P_T}$. Na outra "ponta do cadarço" temos uma propriedade que é válida para $U_{P^*{}_T \circ P'{}_T \circ P_T}$ quando P'_T e P_T forem totais, mas que só é necessariamente válida para $U_{P'_T}$ se este for exatamente o próprio $U_{P^*{}_T \circ P'{}_T \circ P_T}$.[57] Essa propriedade é essencial para provarmos a evolução no *projeto inteligente* de forma análoga à feita por Chaitin.

Enfim, quando provarmos que $P^{**}{}_T \circ P_T$ é total, podemos **"amarrar as duas pontas"**, ou seja, tal propriedade vai ser válida para $U_{P^*{}_T \circ P^{**}{}_T \circ P_T \circ P_T}$ e, consequentemente – o que antes não era necessariamente válido -, também para $U_{P^{**}{}_T \circ P_T}$, como desejado.[58] O "laço", concretizado pelo teorema 2.15.19, só pode ser feito por essas duas "pontas" e elas estão

[57] Ver lema 2.15.16
[58] Ver teorema 2.15.19

necessariamente "amarradas" pelo fato de que a autorreferência em questão precisa estar bem definida.

Até aqui falamos das ideias gerais que "desenham" a demonstração, por isso, agora, vamos discutir como de fato ela é, e o que estamos fazendo.

A prova consiste em construir uma submáquina de Turing $U_{P^{**}{}_T \circ P_T}$ capaz de rodar subprogramas/organismos de forma que possamos demonstrar a evolução metabiológica desses organismos de maneira análoga à já demonstrada por Chaitin. Porém, agora com uma natureza computável.

Em primeiro lugar, vamos construir um análogo à função *Busy-Beaver* e um análogo ao número Ω de Chaitin, a *halting probability*. Respectivamente, estes serão a função $BB^{+}{}_{P^{**}{}_T \circ P_T}(N)$ e o número real $\Omega_{P^{**}{}_T \circ P_T}$.

Feito isso, precisamos fazer com que essa submáquina seja capaz de rodar os programas requeridos na demonstração do *modelo cumulativo* e do *projeto inteligente*. Vamos criar um mundo análogo de forma que os argumentos dos teoremas da metabiologia sobre esses modelos possam ser igualmente aplicáveis. Resumindo: (a) que $U_{P^{**}{}_T \circ P_T}$ seja capaz de rodar um programa $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$ que calcule $BB^{+}{}_{P^{**}{}_T \circ P_T}$ a partir de uma aproximação ρ inferior finita de $\Omega_{P^{**}{}_T \circ P_T}$ como seu input;[59] (b) que $U_{P^{**}{}_T \circ P_T}$ seja capaz de rodar as mutações/programas M'_k, isto é, mutações que calculam qual é o programa $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$ que retorna um valor menor ou igual ao output do organismo anterior e que somam $1/2^k$ ao respectivo ρ, retornando como output um novo programa $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ onde $\rho' = \rho + 1/2^k$ tal que: $\rho' \leq \Omega_{P^{**}{}_T \circ P_T}$ se, e somente se, $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ é um subprograma que se detém dentro do tempo de cômputo de $U_{P^{**}{}_T \circ P_T}$ e possui uma aptidão maior que a do organismo/programa anterior (ou seja, se, e somente se, $U_{P^{**}{}_T \circ P_T}(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho') = U(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho')$).[60]

A ideia é que $U_{P^{**}{}_T \circ P_T}$ seja uma extensão da submáquina U_{P_T}. É **importante frisar que P_T é sempre tomado como sendo um programa arbitrário que computa uma função total qualquer**. Com isso, vamos estender o poder de U_{P_T} para que essa extensão $U_{P^{**}{}_T \circ P_T}$ seja sempre capaz de rodar os programas requeridos do parágrafo anterior.

---

[59] Ver teorema 2.15.20
[60] Ver teorema 2.15.19

Os programas π'_{Ω}, P'_{M} e P''_{M}, $P^{**}{}_{T}$, $P^{*}{}_{T}$ e P_T não precisam ser necessariamente os menores ou melhores – que tenham menor tamanho, em bits, ou que calculam as coisas com mais rapidez. Basta que $P^{**}{}_{T}$ seja capaz de reconhecê-los e rodá-los quando for necessário. Isso ficará mais claro quando explicarmos a definição de cada um deles.

Caberá a nossa natureza computável somente a tarefa de verificar se o output do novo organismo é maior que o do anterior. Não esqueça que, como qualquer mutação (com o organismo anterior como input) é um subprograma, sempre será gerado um organismo novo, não há a opção dessa computação nunca parar. Da mesma forma, qualquer organismo sempre produzirá um output, sempre retornará[61] um número natural. A única questão que permanece é verificar se o output do último é maior que o do primeiro, o que é um problema decidível. Quanto ao resto, ela realiza as mesmas funções da natureza oracular de antes, só permitindo um novo organismo com aptidão maior que o antigo, isto é, que nomeia um número maior que o anterior. Caso a aptidão seja menor ou igual, ela "mata" o novo organismo e mantém o anterior. Esse seria o processo que poderíamos chamar de **"seleção natural" na metabiologia**.

Uma vez conseguido isso tudo, as demonstrações da evolução no modelo de *evolução cumulativa* e no modelo do *projeto inteligente* de Chaitin se tornam análogas. Basta substituir: U por $U_{P^{**}{}_{T}\circ P_T}$; a natureza oracular com acesso a Ω por uma natureza computável que possa decidir sempre se um output de um programa qualquer de $U_{P^{**}{}_{T}\circ P_T}$ é maior ou igual a um output de outro programa qualquer de $U_{P^{**}{}_{T}\circ P_T}$; substituir $BB'(N)$ por $BB^{+}{}_{P^{**}{}_{T}\circ P_T}(N)$; Ω por $\Omega_{P^{**}{}_{T}\circ P_T}$; $\pi_{\Omega}\circ 0^{|\rho|}1\circ\rho$ por[62] $\pi'_{\Omega}\circ P^{**}{}_{T}\circ P_T\circ 0^{|\rho|}1\circ\rho$; M_k por $P''_{M}\circ P^{**}{}_{T}\circ P_T\circ k\circ P'_{M}\circ P^{**}{}_{T}\circ P_T$;[63]

Vamos começar explicando os programas π'_{Ω} e as novas mutações M'_{k}. Depois mostraremos como definir uma extensão de U_{P_T} dada pelo programa $P^{*}{}_{T}\circ P'{}_{T}\circ P_T$, onde $P'{}_{T}$ é, **por suposição, mais uma função total computável arbitrária** e na qual já podemos encontrar os programas de (a) e (b), só que ainda em função de $P'{}_{T}$ (e não de $P^{**}{}_{T}\circ P_T$, isto é, ainda não em função de si mesmos).

---

[61] Outputs binários podem ser facilmente colocados em correspondência com os números naturais.
[62] P_ρ deixarão de ser programas de U e passarão a ser subprogramas, ou seja, programas de $U_{P^{**}{}_{T}\circ P_T}$.
[63] Iremos chamar essa nova mutação/programa de M'_{k}.

Será no lugar de P'_T que faremos a autorreferência de um programa chamando a si mesmo. Nosso programa ${P^{**}}_T \circ P_T$ nada mais será que um programa que lê a si mesmo e roda ${P^{*}}_T \circ {P^{**}}_T \circ P_T \circ P_T$.

O cerne da questão é provar que, mesmo com essa autorreferência, o programa ${P^{**}}_T \circ P_T$ computa uma função total – claro, se a função P_T for total. É importante lembrar que estamos provando uma implicação: se P_T for total, então ${P^{**}}_T \circ P_T$ é total. O que é bem diferente de provar apenas que ${P^{**}}_T \circ P_T$ é total. Por exemplo, mesmo que P_T seja uma função computável total, mas que um sistema axiomático formal arbitrário não possa demonstrar que ela é total, ele ainda pode demonstrar a implicação: se P_T for total, então ${P^{**}}_T \circ P_T$ é total. Porém, obviamente, não poderá provar que ${P^{**}}_T \circ P_T$ é total, pois, isso implicaria diretamente que P_T é total.[64] Resumindo, provaremos que ${P^{**}}_T \circ P_T$ é uma **extensão total** de P_T. Aqui, faremos uma prova por indução.

Como dito, vamos fazer uma prova por indução matemática sobre todos os tamanhos possíveis dos inputs w. Como eles são *bit strings* finitas, só podem ter tamanho finito. Consequentemente, o tamanho dos w´s serão números naturais. Primeiro, como de costume nesse tipo de demonstração, vamos provar que existe k_0 tal que, para todo w com $|w| \leq k_0$, $U({P^{**}}_T \circ P_T \circ w)$ está bem definido. Em seguida, usando a hipótese indutiva, vamos provar que se $U({P^{**}}_T \circ P_T \circ w)$ está bem definido para qualquer w de tamanho $\leq k$, então $U({P^{**}}_T \circ P_T \circ w)$ estará bem definido para qualquer w de tamanho $k + 1$. Basicamente, isso é possível porque o tempo de cômputo de qualquer programa de $U_{{P^{**}}_T \circ P_T}$ que tenha tamanho n somente depende do output de programas de $U_{{P^{**}}_T \circ P_T}$ que tenham tamanho $< n$. O programa ${P^{**}}_T$ será construído para que isso seja verdadeiro. Apesar de ${P^{**}}_T$ ser um programa que chama a si mesmo, qualquer programa de $U_{{P^{**}}_T \circ P_T}$ nunca depende do resultado de si mesmo, nem de subprogramas maiores, para poder gerar seu output.

Agora só nos resta mostrar que a evolução funciona. Vamos começar com o *projeto inteligente*. Para isso precisamos provar que:

---

[64] DORIA, op. cit..

(i) as mutações M'_k (isto é, as mutações $P''_M \circ P^{**}_T \circ P_T \circ k \circ P'_M \circ P^{**}_T \circ P_T$) levam qualquer subprograma/organismo w em um outro da forma $\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ tal que $\rho' = \rho + {}^1/_{2^k}$; [65]

(ii) $\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$ possui uma aptidão menor ou igual à do organismo anterior;[66]

(iii) $\rho' \leq \Omega_{P^{**}_T \circ P_T}$ se, e somente se, $\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ é um programa, com maior aptidão que a do organismo/subprograma anterior, que se detém, ou seja, que $U_{P^{**}_T \circ P_T}\left(\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) = U\left(\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) > U_{P^{**}_T \circ P_T}(w)$;[67]

Note que ρ é sempre uma aproximação inferior a $\Omega_{P^{**}_T \circ P_T}$. Com isso, caso a soma de ${}^1/_{2^k}$ à aproximação inferior ρ de $\Omega_{P^{**}_T \circ P_T}$ a leve mais próximo de $\Omega_{P^{**}_T \circ P_T}$, então $U_{P^{**}_T \circ P_T}$ deixará $\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ rodar quanto tempo for necessário até parar[68] e, caso a soma de ${}^1/_{2^k}$ à aproximação inferior ρ de $\Omega_{P^{**}_T \circ P_T}$ ultrapasse $\Omega_{P^{**}_T \circ P_T}$, então $U_{P^{**}_T \circ P_T}$ tratará $\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ como um programa que não para – o que nesse caso será verdade. Isso acontece porque as mutações foram construídas para esse fim.[69]

Dessa forma, por um argumento totalmente isomorfo ao usado no *projeto inteligente* chaitiniano, para qualquer k, a sequência de mutações M'_1, M'_2, M'_3,..., M'_k leva sempre qualquer organismo/subprograma inicial a um organismo/subprograma final que produz um output (isto é, sua aptidão) maior ou igual a $BB^+_{P^{**}_T \circ P_T}(k)$. Resumindo, a quantidade média de mutações necessárias (o **tempo de mutação**) para se atingir a aptidão $BB^+_{P^{**}_T \circ P_T}(N)$ é $\leq N$.

O cerne dessa prova passa, outra vez, por se fazer uma indução sobre os k's: para $k = 1$ será trivial o resultado; se for verdadeiro para k, então se mostrará verdadeiro para $k + 1$. Os detalhes passam pelas próprias definições dos programas π'_Ω, P''_M e P'_M, que foram construídos para exibir as propriedades desejadas.

---

[65] Isso sairá diretamente da definição desse programa/mutação.
[66] O que se mostrará quase trivial. E note que ρ' não é a mesma coisa que ρ.
[67] Já esse resultado dependerá de lemas e teoremas um pouco mais elaborados.
[68] Devido ao teorema 2.15.19.
[69] E porque um pequeno "truque" está contido no programa P_{MS}, como veremos no capítulo seguinte.

Com esse modelo em mãos, partimos para demonstrar a *evolução cumulativa*. Assim como Chaitin usa as mutações do *projeto inteligente* para estimar uma quantidade média de mutações aleatórias – o que é totalmente diferente do anterior, no qual a natureza/programa escolhe as mutações – necessárias para se chegar a uma aptidão $BB^{+}{}_{P^{**}{}_{T}\circ P_{T}}(N)$, também usamos essas mutações de maneira completamente análoga. A questão é que, como a natureza não permite nunca a aptidão decrescer, não importará a posição em que irá ocorrer a subsequência de mutações M'_1, M'_2, M'_3,..., M'_k no meio de outras mutações/subprogramas quaisquer. De qualquer forma, no final, obteremos um organismo/subprograma com aptidão $\geq BB^{+}{}_{P^{**}{}_{T}\circ P_{T}}(k)$.

Agora, estamos permitindo que ocorra qualquer mutação, sendo elas completamente aleatórias. Claro, cada mutação/subprograma tem uma probabilidade de ocorrer. Com isso, podemos estimar uma quantidade média de mutações para que apareça M'_2 em algum lugar depois de M'_1, M'_3 em algum lugar depois de M'_2, e assim por diante até chegarmos a M'_k. Isso é feito, basicamente, somando a quantidade média de tentativas para ocorrer cada uma delas separadamente. No fim, conseguimos um resultado bastante aproximado para esse somatório, uma cota superior, como feito originalmente por Chaitin. Esse valor, a menos de constantes, é da ordem de $k^2(\log_2 k)^{1+\epsilon}$, onde ϵ também é uma constante.

Note que, como podem existir outras maneiras – além de usar a subsequência M'_1, M'_2, M'_3,..., M'_k – de se chegar à aptidão $BB^{+}{}_{P^{**}{}_{T}\circ P_{T}}(k)$, e como $k^2(\log_2 k)^{1+\epsilon}$ é uma cota superior a um somatório, teremos que $k^2(\log_2 k)^{1+\epsilon}$ só poderá ser, também, uma cota superior para o **tempo de mutação.** Ou seja, a quantidade média de mutações/subprogramas aleatórias para se atingir uma aptidão $\geq BB^{+}{}_{P^{**}{}_{T}\circ P_{T}}(k)$ é $\leq k^2(\log_2 k)^{1+\epsilon}$. O que é um resultado completamente análogo ao obtido por Chaitin.

1.4 PROVOCAÇÕES EM “PARADOXO”

1.4.1 **Quais ideias estão por trás disso tudo?**

Ao tentarmos criar uma natureza metabiológica que seja computável, mas que “se comporte da mesma maneira” que uma oracular, isto é, da mesma maneira que uma natureza incomputável – ou em outras palavras ainda, como um hipercomputador –, vamos nos deparar

com uma série de dificuldades. A primeira delas, como já conhecido na literatura, é que não teremos um programa que resolva o *problema da parada*. O que é necessário para saber se uma mutação algorítmica dá algum novo organismo/programa (o output da mutação aplicada sobre o organismo anterior) e se esse organismo/programa tem aptidão (seu o output) maior que o anterior ou não. Uma natureza computável obrigatoriamente precisaria ser capaz de rodar alguma função que fosse capaz de realizar essa tarefa, a qual, sabemos, é impossível para qualquer programa arbitrário. Como a evolução ocorreria, então?

Caso queiramos construir uma natureza computável que "se comporte como" uma incomputável para a metabiologia[70], precisamos saber, pelo menos, como se darão os novos modelos de *busca exaustiva*, *projeto inteligente* e *evolução cumulativa*. Usaremos os três modelos já estabelecidos como referência para construir três novos modelos análogos.[71] Eles precisam apresentar as mesmas propriedades desejadas por nós: evoluir com organismos e mutações análogos e num **tempo de mutação**[72] igual ou aproximadamente igual. Mas como pudemos fazer isso?

Infelizmente – ou não –, apenas limitar o tempo de cômputo não nos garante que a evolução irá acontecer como desejamos. Nós precisamos, por exemplo, que exista uma função $F(N)$ crescente tal que, pra todo N, o output de nenhum programa (que a natureza computável permita rodar) de tamanho menor ou igual a N (a menos de uma constante) se iguale ou supere o valor de $F(N)$. Essa é a nossa função de aptidão, que serve para "medir" a complexidade [73] ou criatividade dos organismos/programas através do fenômeno da incompressibilidade. Sem essa função, não faria sentido falarmos de uma evolução de programas que vão se tornando mais e mais criativos.

Além do mais, para todo N, tem que existir um organismo/programa, que a natureza computável permita rodar, de tamanho grande o suficiente tal que seu output se iguale ou supere $F(N)$. Caso contrário, teríamos um cenário em que a complexidade dos organismos não cresce a partir de certo ponto, ou seja, a complexidade dos organismos seria limitada. Algo não desejável caso queiramos que a vida – no nosso caso, a "metavida", composta por seres metabiológicos –, como um todo, possa sempre se tornar mais e mais criativa. Não é

[70] E, por conseguinte, pertencente à teoria da informação algorítmica e à teoria da computabilidade.
[71] O de *busca exaustiva* é deixado a cargo do leitor.
[72] É a quantidade média de mutações necessária para levar um organismo inicial qualquer até um organismo final que possui uma aptidão desejada.
[73] *Program-size complexity*.

para toda limitação de tempo de cômputo de um programa que existe esse organismo/ programa. Por exemplo, limitando o tempo de cômputo de uma máquina de Turing por um valor constante, a complexidade desses programas será limitada. Pois, a partir de uma quantidade suficientemente grande de bits em um programa, os próximos bits não farão diferença no cômputo do mesmo. Logo, precisamos que o tempo de cômputo seja suficientemente grande para se aproximar do tempo de cômputo necessário para computar a função F, sem nunca conseguir alcançá-lo por completo. E é aqui que está, como falamos, a dificuldade de se definir uma função relativamente incomputável **de primeira ordem** que não seja muito incomputável em relação aos subprogramas. Talvez esta seja a principal dificuldade, e então se mostrou imperativo o uso da autorreferência, *à la* método diagonal.

Outro requisito não trivial é que, para todo N, exista uma mutação/programa, ou uma sequência de mutações/programas, que levem qualquer organismo inicial a um organismo final com aptidão $\geq F(N)$. É isso que nos diz que a natureza computável é capaz de fazer a evolução sem fim, na qual a criatividade ou complexidade dos organismos sempre pode crescer – e aí reside outra dificuldade.

Para resolver esses problemas, tomaremos emprestado o uso do que chamaremos de **subcomputação**. Coisa que tem relação íntima com o termo subrecursão.[74] Com isso, nascerá o fenômeno da **incomputabilidade relativa ou subincomputabilidade**. Na verdade, um fenômeno totalmente análogo já é bem conhecido na literatura, também chamado de **computabilidade relativa**. No entanto, ele trata da computabilidade entre duas máquinas oraculares de Turing de diferentes ordens. Por exemplo, um conjunto pode ser computável por uma máquina de Turing oracular de ordem N e não pode por uma de ordem $N-1$. Como será dito, esse fenômeno é uma de nossas inspirações. Porém, aqui, tratamos de relativizar a computabilidade entre duas máquinas de Turing sem acesso a oráculos. Para diferenciar os dois fenômenos, vamos chamar o primeiro de **incomputabilidade relativa oracular** e o segundo de **incomputabilidade relativa recursiva.**[75]

Então, vamos nos dedicar, agora, a falar das ideias e problemas que subjazem ao plano filosófico e matemático desse conceito.

---

[74] Ver Basu (1970) e Rose (1987).
[75] De fato, quase não mencionaremos mais a incomputabilidade relativa oracular. Por isso, em geral, quando falarmos de incomputabilidade relativa estaremos falando do caso recursivo.

Primeiro, além dos teoremas de incompletude, do problema da parada e da incomputabilidade de diversas funções (por exemplo, a *Busy-Beaver*), existe outro teorema da lógica matemática também muito famoso, que possui tanta, ou talvez mais, profundeza nas questões filosóficas da linguagem ou na própria fundamentação da matemática. O chamado "paradoxo" de Skolem, que não enunciaremos formalmente aqui, na verdade consiste em se demonstrar que um problema que parece um paradoxo na verdade não o é. O proposto paradoxo se formaria como consequência do famoso teorema de Cantor em contraposição ao de Löwenheim-Skolem. Vamos explicar melhor o que acontece.

O teorema de Cantor – possivelmente o mais importante e conhecido teorema matemático, na teoria dos conjuntos, sobre o infinito – nos diz que a quantidade de pontos num segmento qualquer de uma reta de números reais (ou a quantidade de subconjuntos de números naturais) é estritamente maior que a quantidade de números naturais. Apesar de esta ser também uma quantidade infinita. Isso tudo, tomando como base um conceito matemático básico que chamamos de bijeção, i.e., que podemos sempre, ou indefinidamente, fazer pares de elementos, um de cada conjunto, sem repetições e de forma unívoca. Então, aplicamos o famoso artifício do *método diagonal*, o qual, podemos dizer, corresponde a um argumento em que construímos um procedimento ou uma função que, caso ela chame a si mesma – ou se refira a si mesma de alguma forma – obrigatoriamente produz um resultado diferente do que se tinha antes, gerando uma contradição. Com esse método, conclui-se que não pode haver bijeção[76] entre os dois conjuntos em questão. Furtaremo-nos de apresentar essa demonstração aqui. No entanto, é importante frisar bem essa ideia, pois ela será preciosa para o entendimento do espírito do trabalho proposto.

Esse tipo de método, ou estilo, de demonstração está presente, inclusive, nas provas de incompletude, no problema da parada e da incomputabilidade da função *Busy-Beaver*. A autorreferência também se tornará nosso "motor". Todavia, diferentemente do usual, em que a autorreferência ou o "elemento diagonalizador" são usados para mostrar uma limitação – ou que algo não existe, que algo sempre escapa, culminando em geral numa redução ao absurdo no final –, nós vamos incorporar estas ideias com a própria autorreferência e construir um sistema que existe justamente porque **se autodiagonaliza**. Antes usávamos a autorreferência para mostrar que a existência do sistema suposto leva a uma contradição; agora, a usamos

[76] Na verdade, não pode haver a sobrejeção do naturais sobre o conjunto de todos os subconjuntos dos naturais.

para mostrar o contrário, que o sistema suposto está bem definido. Além de abraçarmos e admirarmos o elemento diagonalizador, como já se fazia antes nos belos teoremas metamatemáticos, nós, agora, o internalizamos dentro de um sistema artificial ou teórico: um programa. Nós o estamos usando para criar e não para provar uma não existência. Internalizar a autorreferência é como andar na corda bamba em cima de um abismo. Um passo em falso e o sistema se torna contraditório, colapsa, "se suicida". O modo mais fácil para nos assegurarmos de que isso não acontecesse foi fazer com que os programas maiores se tornassem definidos porque os menores estão, e assim por diante. Um "método diagonal recursivo", que sempre olha para os menores, e não para todos de uma vez.

Voltando ao teorema de Löwenheim-Skolem advindo da teoria dos modelos na lógica matemática, ele nos prova, por exemplo, que existe um modelo (na verdade, o universo desse modelo) enumerável[77] para qualquer teoria, numa lógica de primeira ordem[78], dentro de uma linguagem enumerável, caso esta teoria seja satisfazível por algum modelo. [79] Sua demonstração é mais técnica que a de Cantor[80], porém, a ideia principal é a de que o tamanho da linguagem em que uma teoria é definida determina o tamanho do modelo que satisfaz essa teoria.

Mas o que é um modelo? Resumindo e pegando a ideia principal, é um conjunto de estruturas semânticas matematizadas por funções e conjuntos. Ele tenta formalizar matematicamente o que é o "significado" de uma sentença. O faz tomando sempre como referência uma metalíngua que contém obrigatoriamente (ou por definição) todas as proposições[81] verdadeiras em relação a uma teoria[82] qualquer. Sendo mais formal agora, dada uma linguagem numa lógica de primeira ordem, uma **estrutura** é composta pelo conjunto de todos os objetos ou elementos (chamamos esse conjunto de conjunto universo ou domínio da

---

[77] Aos conjuntos infinitos que podem ser colocados em bijeção com o conjunto dos números naturais chamamos de enumeráveis ou denumeráveis, ou seja, que podem ser contados, elemento a elemento.
[78] Com apenas um quantificador ∀ e um ∃ sobre variáveis de elementos constantes, e não sobre predicados (como no caso da lógica de segunda ordem).
[79] Podemos também trocar a condição de ser satisfazível pela condição de ser consistente.
[80] E não pertence ao escopo deste trabalho.
[81] Uma proposição, no âmbito da filosofia da linguagem ou da mente, seria aquilo que é o que uma sentença está significando. Uma proposição é composta por conceitos assim como uma sentença é composta por palavras ou símbolos. Por exemplo, "a neve é branca" e "the snow is white" são duas sentenças diferentes, porém, denotam a mesma proposição.
[82] Uma teoria é um conjunto de sentenças fechado sobre todas as deduções. É o conjunto de todas as sentenças que derivam dedutivamente de um conjunto finito ou não de sentenças.

estrutura) que as variáveis[83] e constantes podem denotar e é composta também pelas diversas composições destes através de todas as funções da linguagem (por exemplo, $+,\times,\div,-$) e de todos os predicados da linguagem (por exemplo, $=,<,\in$). Se qualquer função de satisfatibilidade, as quais levam as sentenças da teoria em questão em "proposições" nessa estrutura (ou metalíngua), levar em "proposições" compostas a partir dos elementos dessa estrutura – como descrito acima -, então dizemos que essa estrutura é um **modelo** para a teoria ou que **satisfaz** tal teoria. Isto é, uma estrutura é um modelo para uma teoria se, e somente se, o "espelhamento" (um homomorfismo entre todas as relações sintáticas) de todas as sentenças da teoria estiver contido nessa estrutura.

Contudo, existe uma teoria dos conjuntos capaz de demonstrar, inclusive, o teorema de Cantor. Por exemplo, tome a axiomática de Zermelo-Fraenkel com o axioma da escolha[84]. Além disso, é amplamente aceito que ela é *satisfazível* ou consistente. Aí se apresenta o problema: existirá um modelo enumerável, um conjunto com enumeráveis elementos, para toda uma teoria (ZFC) que diz existir um conjunto não enumerável. **Quando olhamos o conjunto de elementos, pertencentes ao modelo da teoria, que correspondem aos elementos do conjunto que a teoria demonstra ter uma quantidade não enumerável de elementos, ele pode ter uma quantidade enumerável de elementos.** Essa antinomia é um paradoxo? Ou ainda, resulta numa contradição?

A resposta para isso, por mais estranha que possa parecer, dada pelo próprio Skolem em 1922, é que a propriedade da enumerabilidade depende da existência de um conjunto, que corresponde à função que faz a bijeção, dentro do próprio modelo. Essa função, ou conjunto, não pode existir dentro de nenhum modelo de ZFC, pois, se existisse, tornaria enumerável o conjunto de todos os subconjuntos dos naturais. Porém, pode existir "quando olhamos de fora", quando esse conjunto/função não pertence ao modelo. Em outras palavras, a função que faz a bijeção entre os naturais e o conjunto de todos os seus subconjuntos nunca pertence a nenhum modelo da teoria dos conjuntos[85], mas pode existir. Assim, entende-se que não se forma um paradoxo verdadeiro. Forma-se o que podemos chamar de **pseudoparadoxo metalinguístico**: quando olhado "de fora", um objeto tem certa propriedade, mas olhando "de

[83] Aquilo que toma o lugar dos x´s no "$\forall x$" e no "$\exists x$".
[84] A qual é uma famosa candidata a sistema axiomático formal capaz de carregar todas as demonstrações matemáticas dentro de si.
[85] Claro, caso esta seja suficientemente forte e satisfazível.

dentro" tem a propriedade contrária. Na "língua" pode ser dito uma coisa, mas na "metalíngua", o contrário. Ou ainda, a teoria, que está "dizendo", pode "dizer" algo contraditório com outra teoria que está "dizendo o que a primeira teoria diz". Por essa razão, faz sentido chamar o "paradoxo" de Skolem de **pseudoparadoxo metalinguístico da denumerabilidade.**

Essa cisão que ocorre quando passamos da língua para a metalíngua configura o âmago dos nossos "paradoxos" levantados neste capítulo. É preciso frisar bem a ideia desse pseudoparadoxo metalinguístico, pois é daí que vem nossa inspiração para a incomputabilidade relativa, especialmente.

Esse tipo de "inconformidade" formal, de fato, gerou – e, talvez, ainda gera – incômodo na comunidade acadêmica, sobretudo, lógico-matemática. Era do desejo na época logo anterior a esse resultado que se achasse uma axiomatização para a teoria dos conjuntos, numa lógica de primeira ordem, que fosse *categórica*. Isto é, que pudesse existir apenas um modelo, a menos de um isomorfismo, satisfazendo tais axiomas. O teorema de Löwenheim-Skolem pôs um fim nisso, ao mostrar que o tamanho dos modelos de teoria consistentes, numa lógica de primeira ordem, está mais preso ao tamanho da linguagem do que ao que a teoria "diz".

Por coincidência ou não, o mesmo aconteceu com os teoremas de Gödel, em 1931, colocando um fim aos anseios de se construir um sistema axiomático formal que fosse forte o suficiente para demonstrar grande parte do que sabemos sobre a aritmética (ou sobre a teoria dos conjuntos) e, ao mesmo tempo, consistente e completa, que "abraçasse tudo". Em outros termos, não pode existir um programa que decida sempre se uma sentença matemática é verdadeira ou não.

Mas o que isso tem a ver com o "paradoxo" de Skolem?

Debruçar-nos-emos sobre um fenômeno irmão da incompletude: a incomputabilidade. O mais conhecido e principal é, obviamente, o problema da parada, mostrado por Turing em 1936. Sabemos de uma miríade de problemas e funções que são incomputáveis. Para construirmos nosso novo pseudoparadoxo, vamos no focar na incomputabilidade: o fato de não poder haver programa que compute um problema ou uma função. Por isso, cabe a pergunta: assim como existe um **pseudoparadoxo metalinguístico da denumerabilidade** pode existir um **pseudoparadoxo metalinguístico da computabilidade**?

Vamos caminhar nessa direção dentro do âmbito da metabiologia. Não à toa falamos de início em fazermos uma natureza computável se "comportar" como uma oracular – já que um sistema com acesso a um oráculo *à la* Turing é um sistema incomputável.

Tentar emular as propriedades do incomputável dentro de um computador seria uma tarefa infindável e impossível. Já sabemos disso. Por maior e mais complexo que seja um programa que tente computar o que um oráculo de primeira ordem pode saber, ele sempre estará fadado a falhar em um número infinito de problemas. A quantidade de informação necessária para computar um oráculo de primeira ordem é infinita e qualquer programa que se proponha a isso sempre contém, no máximo, uma quantidade finita de informação. É como partir do número zero e andar a passos infinitesimais em direção ao número real 1. Você nunca irá chegar, a não ser que dê infinitos passos.

Passando para o jargão lógico-matemático, cada sistema axiomático formal (SAF) pode ser entendido – ou traduzido – por um programa. Lembre que um sistema axiomático formal é uma coleção finita de sentenças numa linguagem determinada e numa lógica dedutiva formal (em geral, a lógica de primeira ordem). Portanto, não pode existir nenhum sistema formal[86] que possa demonstrar corretamente quais são os bits de Ω.[87] Fazendo a equivalência entre os graus de Turing e as sentenças na hierarquia aritmética[88], isso tudo nos permite dizer, então, que não pode existir nenhum sistema formal que possa sempre decidir se qualquer sentença aritmética de ordem Σ_1 [89] é verdadeira ou é falsa[90]. Isso já limita muito o poder de qualquer SAF. Eles não são capazes nem de lidar com o primeiro nível na hierarquia aritmética. Podem-se buscar diversos tipos de extensões da aritmética, porém, por exemplo, mesmo adicionando os **princípios de reflexão** ao esquema de provas possíveis, que nada mais é que tentar forçar a *corretude* para dentro do sistema [91], nunca poderemos demonstrar sempre se um programa qualquer se detém ou não.

Por outro lado, não se trata de um teorema tão devastador assim para os anseios da computação. De fato, isso não exclui a possibilidade de sempre podermos ir incrementando nossos programas indefinidamente. Todavia, nunca chegaremos ao final, a um programa completo. Claro, pelo visto, sempre podemos ir incrementando o poder dos sistemas axiomáticos; uma forma é adicionar mais e mais axiomas. O que nos faz ficar cada vez mais

---

[86] Forte o suficiente para carregar qualquer demonstração aritmética, no mínimo.
[87] O número Ω de Chaitin é nosso exemplo canônico de número oracular de primeira ordem.
[88] Ver também capítulo 3.
[89] Σ_1 é o conjunto de sentenças aritméticas na forma normal **prenex** (isto é, com quantificadores abrangendo sempre a fórmula inteira) tal que só existe um único quantificador de existência ∃ na fórmula inteira.
[90] Dentro de um modelo standard da aritmética, no qual o próprio número Ω está "inserido".
[91] BARWISE, J. (Ed.). **Handbook of Mathematical Logic**. Amsterdam: Elsivier Science Publisher B.V., 1977. ISBN 0444863885.

próximos de saber os bits de Ω. Porém, de qualquer forma, estaremos sempre a infinitos *bits* de distância dele. O leitor poderia, então, se perguntar agora: mas e de onde vêm esses axiomas? Será que esse processo gerador de tais axiomas para nós pode ser uma fonte de informação infinita? Poderiam vir de algum oráculo ao qual temos acesso? Eis o que está no âmago do nosso trabalho. Discutiremos esse assunto mais à frente.

A questão é que, ao "correr atrás" de um hipercomputador usando um computador, estaremos olhando para a computabilidade como uma **propriedade absoluta** e não como uma **propriedade relativa.** Um oráculo de primeira ordem é o mesmo – a menos de um isomorfismo ou de uma emulação – independentemente da máquina universal de Turing que use como referência. Não é possível fazer um programa qualquer computar um problema incomputável que um hipercomputador poderia "computar". Essa impossibilidade vem, justamente, ao tomarmos o fato de um problema ser incomputável como uma propriedade absoluta. No entanto, assim como a denumerabilidade, como mostrado por Skolem, a computabilidade também pode depender "de onde se está olhando": se de fora ou de dentro do sistema. Depende do ponto de vista de "quem está perguntando". Quando vemos a incomputabilidade como uma propriedade relativa, abre-se um novo "leque de possibilidades".

Portanto, definindo o que seja **um subcomputador**, que nada mais é do que uma máquina de Turing que sempre gera um output (e que sempre se detém), a definição de **incomputabilidade relativa** aparece naturalmente sobre uma nova versão relativa da função *Busy-Beaver*[92]. Além disso, construímos um subcomputador tal que essa nova função *Busy-Beaver* é "tão" incomputável para ele quanto a *Busy-Beaver* original é para um computador – o que, enfim, nos dá um resultado a favor do **pseudoparadoxo metalinguístico da computabilidade**.

Isso fica cristalizado com o resultado final, quando mostramos que existe uma evolução de organismos/subprogramas metabiológicos numa natureza computável "da mesma forma", como já mencionamos no início, que a evolução de organismos/programas metabiológicos numa natureza incomputável (oracular). A nova natureza, um sistema computável, "se comporta como se fosse" oracular, como se fosse um sistema incomputável,

[92] Reconhecidamente como uma função incomputável e usada nas demonstrações da metabiologia.

porém, somente em relação aos subprogramas. Ela simula, de forma indireta, um oráculo de primeira ordem de Turing. Só que para seus subprogramas.

Isso não quer dizer que emulamos todas as propriedades de um hipercomputador de primeira ordem. Em outras palavras, o resultado aqui suscitado é um primeiro resultado positivo quanto a mostrar que a computabilidade é um fenômeno relativo: de um sistema para um subsistema. Algumas propriedades importantes do incomputável[93], mas não **todas**, foram relativizadas. Pelo menos em alguns aspectos, pudemos "imitar" os "deuses", mas ainda não em todos. Neste trabalho, os que conseguimos "imitar" foram apenas os suficientes para fazer uma metabiologia *à la* Chaitin, i.e., como se a natureza fosse um hipercomputador.

Olhando a hierarquia de Turing, onde se encontra uma "escada" de hipercomputadores, o de cima mais poderoso que o de baixo, podemos ver que cada um no nível imediatamente acima está definido justamente pelo que o de baixo (de referência) não consegue fazer, i.e., não consegue computar. Essa hierarquia é sempre construída "de baixo pra cima", partindo no nível zero, que é o da máquina de Turing universal, ou seja, um computador com recursos ilimitados. Porém, nada nos impediria de "olharmos de cima para baixo". Como definir uma máquina universal de Turing dado, primeiramente, um hipercomputador de primeira ordem? É respondendo a essa pergunta que vamos chegar diretamente na subcomputação.

Seguindo a própria definição de subcomputador, veremos que uma máquina universal de Turing nada mais é que um "subcomputador" de um hipercomputador de primeira ordem. Este último sempre pode decidir se uma máquina de Turing para ou não, mas não é um computador e sim um hipercomputador. Portanto, é possível entender a própria hipercomputação como algo relativo desde o início. Desse ponto de vista, o presente trabalho não traz tanta mudança de perspectiva assim sobre o assunto. Até, poderemos dizer que, caso consigamos relativizar todas as propriedades de um hipercomputador de primeira ordem, traduzindo-as para a subcomputação, criaremos um **grau de Turing negativo**, abaixo do nível zero, dentro da mencionada hierarquia. E, isso dado, por que não especularmos sobre toda uma hierarquia negativa de Turing? Um campo que poderíamos chamar de **hipercomputação relativa recursiva**, onde poderíamos emular oráculos dentro de um

[93] Em nosso caso, estamos nos restringindo aos problemas incomputáveis de primeira ordem, como a função *Busy-Beaver*, por exemplo.

computador, justamente criando subsistemas de subsistemas, e assim por diante. No entanto, isso ficará para investigações futuras.

Talvez, a forma mais adequada de apresentar os resultados desta tese seria ter primeiro formalizado o que é um grau negativo de Turing, definido a hierarquia negativa de graus e, só então, usado esse arcabouço teórico para construir a evolução metabiológica *à la* Chaitin numa natureza computável. Ao invés disso, pulamos essa definição mais geral e tentamos mostrar que pouco se precisa para conseguir formalizar uma evolução metabiológica computável "como se fosse incomputável" – nosso objetivo principal nesta parte do texto. Isto e as próprias provocações filosóficas desta seção ajudam a levantar o fenômeno da incomputabilidade relativa recursiva como algo frutífero e penetrante.

1.4.2 **O problema da intuição matemática**

Portanto, como já indicado acima, essas ideias não têm inspiração proveniente apenas de problemas matemáticos. O exemplo que iremos estudar nessa seção concerne ao matemático – e não à matemática propriamente –, apesar de ser fortemente relacionado com os teoremas de incompletude. Ou seja, diz respeito a nós humanos. A questão é: **o matemático**[94] **é completo**[95]? Ele é capaz de responder a qualquer pergunta sobre a aritmética ou sobre a teoria dos conjuntos? Muitos dos termos e conceitos que serão levantados nos próximos parágrafos careceriam de uma definição e uma explicação filosófica mais completa, o que optamos por não esmiuçar aqui, visto que, por enquanto, toda essa discussão tem um caráter muito mais elucidativo e inspirador do que explicativo.

Nós de fato temos essa intuição – bem fundada ou não – de que, para qualquer sistema formal desse tipo, conseguimos construir uma sentença godeliana, indecidível pelo próprio

[94] Apesar da ambiguidade interessante daí advinda, estamos nos referindo a nós, humanos, que fazemos e estudamos matemática.

[95] Um sistema axiomático formal completo seria aquele que sempre pode demonstrar se uma sentença é falsa ou verdadeira. Ou, mais formalmente, uma teoria é completa se, e somente se, obrigatoriamente qualquer sentença ou sua própria negação pertencer a essa teoria. Uma teoria é, resumidamente, o conjunto de todas as sentenças que derivam dedutivamente do sistema axiomático.

sistema, mas que sabemos – que intuímos[96] – se é verdadeira. Parece que temos um "mecanismo" de criação de axiomas em nossa mente. Por exemplo, qualquer extensão[97] da axiomática de Peano (PA)[98] feita para demonstrar cada vez mais sentenças indecidíveis – mas verdadeiras no modelo standard – pela PA, não consegue demonstrar sua própria consistência. O que, para nós, parece intuitivo e verdadeiro, por construção.[99] Não é à toa que criamos provas da consistência da aritmética em sistemas mais poderosos que o dedutivo, aos quais não podemos aplicar os teoremas de incompletude *à la* Gödel [100], sobre as quais voltaremos a falar logo abaixo.

O problema todo passa a ser, então, como decidimos ou intuímos que uma sentença é verdadeira ou não. O que, em nossa mente, realiza tal tarefa? E quão poderoso é esse processo mental? Se esse "mecanismo" mental de criação de axiomas for equivalente a um processo hipercomputável, estaria explicado por que nenhum sistema axiomático formal que criamos consegue dar conta de tudo que podemos fazer. Pois, nenhum deles seria capaz de computar os problemas que esse "mecanismo" mental consegue. **A nossa intuição estaria sempre um passo**[101] **à frente da consciência,** o que é uma ideia muito importante para nós, como veremos adiante. No entanto, a subcomputação e a incomputabilidade relativa mostra que essa pode não ser a única opção.

Enunciamos e pensamos sobre sistemas formais enquanto dizemos estar conscientes. Enquanto dizemos, num senso comum – absolutamente não tão infundado –, estar conscientes de tudo aquilo que nos vêm à mente, inclusive, conscientes da própria matemática. Mas, como verificar se somos incomputáveis, no que concerne à consciência e à intuição, pelo menos?

Um modo fácil seria comprovar que conseguimos trabalhar com infinitas coisas ao mesmo tempo. Imagine saber todas as casas decimais de π ao mesmo tempo ou contar de um até infinito. Se fôssemos capazes desses feitos, poderíamos rodar uma regra infinitária – por

---

[96] De fato, podemos criar um sistema mais forte no qual essa sentença se torne demonstrável. Mas, assim, podemos criar outra sentença godeliana para este último, e assim por diante. Esse processo não teria fim e, por isso, nenhum sistema desse tipo justificaria a sensação ou intuição que temos de que a sentença é verdadeira.
[97] Por exemplo, usando os princípios de reflexão de Feferman, já mencionados.
[98] O principal sistema axiomático para a aritmética.
[99] Caso achemos intuitiva e verdadeira a consistência de PA.
[100] Por exemplo, a demonstração de Gentzen. Ver ABRAHÃO, F. S. **Demonstrando a Consistência da Aritmética**. Dissertação de mestrado. ed. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2011.
[101] Nesse caso, seria mais apropriado até dizer que estaria infinitos passos à frente.

exemplo, *a la* Gentzen ou *a la* Schütte – formando um sistema de prova não dedutivo[102] muito mais poderoso que PA, no qual não vale Gödel, e que pode responder a qualquer pergunta sobre a aritmética, ou seja, que é completo. Uma regra infinitária é uma regra de derivação lógica que é capaz de partir de um número infinito de premissas e retornar apenas uma conclusão. No caso, é uma regra que "dedutifica" a indução: se as sentenças $P(1), P(2), P(3), \dots, P(n), \dots$ são válidas para qualquer n, então $\forall x P(x)$.

No entanto, isso parece fora de nossa capacidade humana, pelo menos, até onde sabemos. Como verificaríamos que a sentença $P(n)$ é válida para todo n sem percorrer individualmente cada instanciação da mesma? Mas nem tudo está perdido. Pode ser que esse processo infinitário ocorra de forma inconsciente, em algum nível infraconsciente – "abaixo" do consciente[103] – dos processos mentais, os quais interagem "conosco" somente pela intuição, nos dando a sensação ou convicção[104] de que uma sentença é verdadeira ou de que não é.

Parece que estamos presos a raciocinar conscientemente com um número finito de conceitos e imagens por vez. Parece que somos finitos a cada instante. Quando nos deparamos com uma sentença indecidível, para termos certeza de que nossa intuição sobre ela é verdadeira ou não, criamos outro sistema mais forte que a demonstre. Este, por sua vez, também possui seus indecidíveis, para os quais criamos outro sistema mais forte ainda para justificar nossa intuição. E, assim por diante. Tal processo de "criação de axiomas" não pararia ou não encontraria fim. Por conseguinte, nossa pergunta se transforma em: esse processo que continua para sempre é computável?

Para responder que não, precisamos excluir a possibilidade de que os processos geradores dessas verdades intuitivas sejam processos computáveis. Caso sejam, isso deve levar a uma contradição. Um exemplo de contradição seria a computabilidade desses processos contra o "fato" de nunca conseguirmos criar um sistema formal que compute esses processos? Esse "fato" não é tão certo assim. Apenas parece que é válido, considerando nosso conhecimento até o presente momento.

---

[102] Pois, este sistema é capaz de rodar não só todas as regras de derivação dedutivas mas também a regra infinitária.

[103] Numa analogia com a computação, diríamos que se a consciência está no nível da linguagem de programação, o infraconsciente está no nível das sub-rotinas da linguagem de máquina. Está num nível de processos muito mais de base, os quais, dentre outras coisas, "geram ou produzem" a consciência do matemático.

[104] O termo mais adequado seria o da filosofia da mente: **atitude proposicional**.

Enfim, se conseguíssemos criar tal sistema, não seria mais possível construir sentenças indecidíveis tais que saibamos intuitivamente que elas são verdadeiras ou falsas. O nosso processo de "criação de axiomas" cessaria[105]. Um sistema formal com essa característica pode ser chamado de **monolito**: aquilo sobre o qual nossa matemática humana se vê totalmente ignorante em relação a sua consistência. Um monolito responderia bastante bem que o processo de "criação de axiomas" humano é computável, mas, enquanto tal monolito não aparece – se é que isto é possível –, resta-nos tentar responder à última pergunta. Colocando-a em outras palavras: pode nossa intuição matemática ser computável e consistente, mesmo que nunca consigamos achar[106] um monolito?

Voltando ao nosso tema, o interessante é que a subcomputação e a incomputabilidade relativa dá uma resposta quase totalmente positiva a essa pergunta. Isto é, responde que é possível esse processo de criação ser computável enquanto que nunca conseguiremos criar um sistema formal que compute esse mesmo processo, i.e., enquanto esse próprio processo de criação de axiomas nunca termine. Por quê? Caso a consciência seja um subsistema, um subcomputador, da mente, tal processo poderia ser computável e, por outro lado, incomputável relativamente à consciência, ou seja, incomputável relativamente a qualquer subprograma. O presente trabalho mostra essa possibilidade. **Apesar de existir um sistema axiomático que faça tudo que o matemático pode fazer, ele poderia nos ser inalcançável.**

Por exemplo, podemos imaginar a consciência como composta por estados mentais[107], os quais são outputs de processos mentais[108], que, por sua vez, são programas. No entanto, a conformação desses programas/processos mentais pode ser tal que qualquer sequência de outputs/estados mentais da consciência que forme um cômputo seja um subprograma rodando. Dessa forma, a única opção de sabermos a resposta de um problema relativamente incomputável pela consciência é através da intuição, a qual não provém de um raciocínio ou juízo[109] do consciente. O "pensar de forma consciente" seria subcomputável, porém, o "pensar de forma não consciente ou intuitiva" pode ser mais poderoso que isso, pode ser

[105] Do ponto de vista de um modelo standard.
[106] É mais que demonstrar, porque um monolito só pode ser evidenciado como tal pelo teste empírico e/ou cognitivo – subjetivo em grande parte, com certeza – de que não consigamos saber nem intuir se ele é consistente ou não, por exemplo.
[107] MCLAUGHLIN, B.; BECKERMANN, A.; WALTER, S. (Eds.). **The Oxford Handbook of Philosophy of Mind**. New York: Oxford University Press Inc., 2009. ISBN ISBN 978-0-19-926261.
[108] MCLAUGHLIN, B.; BECKERMANN, A. et al, Id. Ibid..
[109] Nada impedindo que estes provenham do infraconsciente, portanto.

computável[110]. E no hiato entre esses dois níveis de pensamento aparece justamente a incomputabilidade relativa.

O que nos leva de volta à ideia de que a intuição estaria sempre um passo à frente da consciência. No entanto, um pouco diferente do que se a intuição fosse realmente hipercomputável. Neste caso, podemos tomar emprestada a famosa parábola de Zenão sobre Aquiles com a Tartaruga. Imagine que a tartaruga (nossa intuição) esteja a uma distância suficientemente grande e infinitamente divisível, mas finita ou limitada, à frente de Aquiles (nossa consciência matemática). Por mais que ele corra[111], nunca chegará na tartaruga, a qual estará sempre a uma distância à frente dele. Pois, mesmo que Aquiles chegue no lugar de onde partiu a Tartaruga, esta já terá andado um pouco e estará mais à frente. E esse processo segue indefinidamente, justamente porque a distância a se percorrer pode ser divisível o quanto se queira. Portanto, para ele alcançá-la, teria que percorrer uma quantidade infinita de intervalos de distância e, por isso, numa quantidade infinita de intervalos de tempo. Esse problema é solucionável matematicamente por meio do cálculo integral, já que podemos mostrar que essa soma infinita de intervalos de tempo converge para uma quantidade final finita, isto é, o tempo para ele alcançar a Tartaruga é, afinal, finito.[112] Em nossa analogia, então, a consciência só poderia alcançar a intuição se lhe fosse permitida uma sequência infinita de pensamentos.

Já no caso computável, do parágrafo anterior, é melhor usarmos outra parábola: a imagem do Burro e a Cenoura. Imagine um homem sentado num burro faminto com uma vara e uma cenoura presa na ponta desta. O homem exibe a cenoura na frente do rosto do burro e assim faz o faminto burro andar para alcançar a cenoura, mas este nunca a alcança. Porém, não porque ela está a uma distância infinita dele, mas sim porque toda vez que ele anda a cenoura anda junto. A cenoura está sempre "um passo" à frente do burro. Em nossa analogia, a consciência corre para alcançar a intuição, mas ela nunca tem sucesso porque, por mais que ela corra, a intuição sempre estará um passo à frente em função do que a consciência correu.

Sob essa perspectiva, mesmo que um dia cheguemos a um candidato a monolito, ele nunca poderá computar tudo o que nossa intuição pode. O máximo que ele pode fazer é computar tudo o que nossa intuição já computou até o momento que o criamos. Portanto, para

---

[110] Por um computador que tenha o respectivo subcomputador como seu subsistema.
[111] Considerando que Aquiles só pode correr uma distância finita por vez.
[112] Outra forma de solucionar a parábola do Aquiles e a Tartaruga é através dos ordinais de Cantor.

nós, conscientes, pareceria que nós mesmos (nossa mente) somos incomputáveis, enquanto não realmente sendo – caso algo ou alguém possa nos olhar "de fora". Aí estaria outro pseudoparadoxo metalinguístico[113]: o **"paradoxo" da intuição matemática**.

É preciso dizer que existem diversos tipos de intuições matemáticas. Aqui, nos remetemos tanto à matemática lógico-formal quanto à computacional, porque elas são as mais próximas do nosso tema, além de serem as áreas canônicas da onde brotam os argumentos que dizem que nossa capacidade humana para matemática não é computável, ou emulável dentro de um computador.[114]

De qualquer forma, podemos dizer que esse "paradoxo" é uma ironia interessante contra o nosso "antropocentrismo de cada dia"[115]. Estaríamos destinados (programados) a nos vermos como "superiores" às máquinas, mesmo que, na verdade, sejamos computáveis. Seríamos algum tipo de "máquina arrogante", que se faz acreditar **completa** sem sê-lo. É, de fato, mais irônico ainda pensar que um possível "pilar central" da consciência[116], por meio da qual tomamos ciência de nós mesmos, tenha como base uma espécie de autoengano: a incomputabilidade relativa[117].

1.4.3 **A consciência como um subcomputador da mente**

A ideia da consciência[118] ser um subcomputador da mente resolve outros problemas interessantes. A definição básica de um subcomputador é que ele é um computador que sempre produz output para qualquer input, ou ainda: existe um programa que sempre sabe o output de qualquer subprograma de seu respectivo subcomputador. Nós sabemos que

---

[113] Só que, agora, muito mais filosófico que matemático.
[114] HORST, S. The Computational Theory of Mind. **The Stanford Encyclopedia of Philosophy**, 2011. Disponivel em: <http://plato.stanford.edu/archives/spr2011/entries/computational-mind/>. Acesso em: 23 Março 2015.
[115] Obviamente, uma ironia apenas válida caso nossa mente seja de fato computável.
[116] Ou da Inteligência Artificial Fraca, se quisermos ser mais acurados e respeitar nossas hipóteses filosóficas.
[117] Aqui nos baseamos na hipótese da mente computável. Porém, um "paradoxo" análogo também vale para a se a mente e a intuição forem hipercomputáveis e a consciência um hipercomputador de ordem menor (também, um "subcomputador" da mente).
[118] Não valerá a pena, por enquanto, distinguir entre autoconsciência, tomar ciência, senciência, experiência qualitativa, estado de vigília e outras propriedades ditas da consciência. Para simplificar, consideremos que o consciente é o estado mental em que todas estas propriedades estão presentes. O que, provavelmente, se encaixa no senso comum do que entendemos por consciência. Ver MCLAUGHLIN, B.; BECKERMANN, A. et al, op. cit..

identificar qualquer processo mental com um cômputo gera vários problemas filosóficos. Um exemplo poderia ser a natureza dos fenômenos da consciência, ou o que são nossos estados qualitativos da experiência cognitiva. Esse é um campo muito rico e controverso, mas fugiria completamente do objetivo do nosso texto. Para fins práticos, vamos nos ater à natureza processual ou funcional da mente, onde o que importa são as relações entre as coisas ou objetos mentais. Se o leitor quiser se aprofundar mais nessas questões filosóficas, é apropriado dizer que estaríamos nos atendo à visão funcionalista[119] da mente. Dessa forma, seria pertinente discutir se a mente computa ou hipercomputa, nos permitindo continuar com nossa elucidação sem maiores problemas.

Por que parece que sempre podemos produzir um conceito sobre tudo aquilo que tomamos consciência?[120] Por que estamos sujeitos a ilusões de ótica, ou ainda, por que a imagem que vemos, a qual já seria pré-processada, não é a mesma imagem que nossos olhos captam?[121] Por que sempre podemos fazer alguma escolha – mesmo que não saibamos qual a melhor – se tivermos um limite obrigatório de tempo para isso?[122] Por que os pensamentos nunca param de vir a nós, enquanto estamos no estado consciente?

Todas essas perguntas dançam em volta de uma função básica da mente. Estamos falando dos processos mentais que geram um estado mental dado que outro ocorreu, isto é: aqueles que geram os pensamentos à consciência; aqueles que escolhem algo quando somos obrigados; aqueles que produzem a imagem e sentido, os quais tomamos consciência, a partir dos que os olhos captam; aqueles que sempre produzem um entendimento sobre tudo aquilo que é objeto da mente. Esses são exemplos – ingênuos, porém, não menos elucidativos e gerais – com um simples intuito de mostrar que os processos mentais sempre produzem um resultado final, sempre geram um output, sempre nos dão outro estado mental, dado que outro ocorreu. Eles nunca entram em loop, eles nunca param – pelo menos não enquanto se diz que estamos conscientes ou sãos. Essa seria uma função básica da mente: manter os pensamentos fluindo. Nós só teríamos acesso àquilo que a mente já "computou".

Uma experiência mental simples – e profunda do ponto de vista filosófico – que torna isso evidente é tentar escolher a escolha do agora. Como? Tente escolher aquilo que você vai

---

[119] O funcionalismo, resumidamente, é um ponto de vista filosófico em que o que importa são as relações entre as coisas ou elementos de um sistema e não do que essas coisas são feitas e nem o que são.
[120] Guardadas as devidas situações em que não se considera um indivíduo plenamente consciente de si mesmo.
[121] Isso poderia ser estendido a todos os sentidos.
[122] Salvo quando se diz que o indivíduo está acometido de algum distúrbio mental mais sério.

escolher pensar. Tente escolher seu próximo pensamento enquanto você se predispõe a escolher esse próprio pensamento. Não importa se você decidiu pensar em bananas ou em barcos. Você só toma conhecimento da escolha feita quando o próprio pensamento contendo sua escolha aparece. Você só soube que escolheu pensar em bananas ao invés de barcos quando essa escolha já se apresentou. Depois, claro, você pode mudar sua escolha indefinidamente, conforme tenha vontade. Contudo, isso não muda o aparente fato de que não podemos escolher o que escolhemos. A escolha, em última instância, só pode ser escolhida ou não quando já nos foi apresentada. Nós só temos acesso às escolhas/outputs que a mente já "computou".[123] A mente só nos dá livre-arbítrio até onde ela consegue "computar". No fundo, a escolha é feita pela mente[124] antes mesmo de sabermos conscientemente qual ela é. Qualquer coincidência com as teorias compatibilistas[125] do **livre-arbítrio** não é mera coincidência, como explicaremos um pouco melhor a seguir.

Partindo da hipótese de que a mente é computável[126], não é à toa que isso tudo casa perfeitamente com a ideia de subcomputação. Para manter os pensamentos sempre fluindo, qualquer processo ou operação consciente precisa sempre resultar em algo, isto é, a mente sempre tem que poder dar algum pensamento resultante desse processo.

Se assumirmos, agora, a hipótese de que a mente é computável, então, poderíamos dizer que a consciência é o produto, ou uma sequência de outputs de processos mentais (subprogramas) que os geram. Um dos efeitos colaterais mais interessantes – e, talvez, irônico – que pode aparecer, como mostrado aqui, é a intuição ter um poder de cômputo sempre maior que qualquer operação consciente: o "paradoxo" da intuição matemática, por exemplo. O que, fortuitamente ou não, pôde – e poderia para outros exemplos não discutidos nesta tese – também nos dar *insights* sobre vários problemas da ciência cognitiva e filosofia da mente. Como já mencionado, um desses "efeitos colaterais" seria o aparecimento do livre-arbítrio[127]. Nós nunca conseguimos prever o processo mental infraconsciente que faz a escolha porque este poderia ser relativamente incomputável pela consciência – esta entendida como um

---

[123] Completando a analogia: o input desse cômputo pode incluir os estados mentais anteriores à escolha, fatores fisiológicos, bioquímicos, etc.
[124] Isto é, pelos processos mentais infraconscientes.
[125] MCKENNA, M.; COATES, J. Compatibilism. **The Stanford Encyclopedia of Philosophy**, 2015. Disponivel em: <http://plato.stanford.edu/archives/sum2015/entries/compatibilism/>. Acesso em: 14 Abril 2015.
[126] Ou hipercomputável, se o leitor considerar que um computador é um subcomputador de um hipercomputador.
[127] Na verdade, um livre-arbítrio fraco, em que nossas escolhas parecem livres para nós enquanto que, na verdade, elas são determinadas por relações causais deterministas.

espaço de subprogramas rodando. O compatibilismo entre a computabilidade da mente e o livre-arbítrio seria, justamente, feito pelo fenômeno da **incomputabilidade relativa recursiva**. De qualquer forma, toda essa discussão merece ser investigada com mais profundidade no futuro.

Sob alguns aspectos, majoritariamente dentro do escopo do funcionalismo, todas essas ideias, sim, corroboram com a visão da mente computável, na qual a mente pode ser entendida como um sistema físico-químico cujas relações podem ser emuladas por um computador suficientemente poderoso [128]. Contudo, sendo mais cuidadoso com nossas conclusões, só podemos dizer que elas servem para dar um contra-argumento às teorias que invocam a "aparente" incomputabilidade da mente como justificativa para sua própria incomputabilidade. No entanto, elas não servem para dar "um ponto final" ao assunto, a favor da hipótese computacional da mente. A mente ainda pode ser realmente incomputável, por motivos outros que não a sua "aparente" incomputabilidade. Por acaso, mas não menos intencionalmente, essa opção também é explorada, de forma indireta, através dos "hiperorganismos", nessa tese, no capítulo 3.

1.4.4 **O "paradoxo" da computabilidade biológica**

Vamos, então, nos voltar para problemas mais objetivos, ditos das ciências naturais. A biologia é, com certeza, um dos focos de toda a problemática do tema desta tese. Veja a própria metabiologia. Simular ou emular num computador o funcionamento de um ser vivo ou um organismo inteiro, por mais simples que seja, se tornou um programa de pesquisa importante em alguns campos, e importante dentro da própria biologia. Isto tem tudo a ver com os seres vivos serem computáveis ou não, ou seja, se seria necessário um hipercomputador[129] para emular um ser vivo ou se um computador já bastaria.

Os organismos são conhecidos por serem sistemas altamente complexos, com intrincados mecanismos químicos, dos quais ainda se acredita não saber como muitos deles funcionam. Por exemplo, o projeto genoma humano se deparou com numerosas dificuldades

[128] Por suficientemente poderoso, dentro da teoria da computação, entende-se um computador com recursos energéticos e memória suficientemente grandes.
[129] Também suficientemente poderoso.

para identificar a função de cada gene. O próprio DNA não codificante ou "DNA lixo", como é chamado uma parte do nosso genoma que se acreditava não ter função fenotípica, pode possuir funções de ativação e inibição dos próprios genes codificadores de proteínas. Mas isso, a princípio, não configuraria uma impossibilidade em relação a esse sistema (o organismo) ser computável. Existem programas tão complexos que nem chegamos perto ainda de programá-los. A dificuldade não é parâmetro restritivo a esta hipótese. Precisamos de algo mais substancial, precisamos de uma impossibilidade.

Por isso, a questão fica em saber qual o poder de cômputo de um sistema vivo: se é hipercomputável ou se é computável. Isso vem desde se a física do universo é digital ou analógica, até se podem surgir propriedades emergentes não computáveis de sistemas digitais físico-químicos. Maiores discussões e exemplos sobre o assunto podem ser encontrados no capítulo sobre evolução de hiperprogramas. Para o âmbito da subcomputação, a questão importante se torna saber se, de maneira bastante análoga ao "paradoxo" da intuição matemática, os seres vivos podem parecer incomputáveis para nós, mesmo não o sendo realmente?

Se somos um tipo de organismo e a nossa biologia parece para nós mesmos como sendo incomputável, isso quer dizer que o ser humano – enquanto organismo – é incomputável? Assumindo, outra vez, um ponto de vista próximo ao funcionalismo[130] e ao fisicalismo[131], teríamos que o nosso "poder de cômputo mental" não poderia ser superior ao poder de cômputo do ser humano, enquanto organismo. Visto que, assim, a mente – e também a consciência – humana seria um subsistema do organismo humano.

Pois, então, a mente não poderia ser um subcomputador do organismo humano? E, por sua vez, a consciência um subcomputador da mente. Vamos dar um jeito simples de exemplificar isso. Antes, porém, vale ressaltar que não é nossa intenção dizer quais sistemas físicos fazem o papel de hardware para os programas que fariam o papel de organismo ou mente. Primeiro, porque dentro da nossa linha de discussão, partindo de nossas hipóteses, especificar qual é o hardware não importa. Segundo, porque se trata de um campo de investigação ainda muito controverso que escaparia muito do nosso escopo.

Dito isto, sigamos em frente.

---

[130] Só que agora para a biologia.
[131] O fisicalismo quer dizer que a mente pode ser completamente explicada em termos da física ou das leis do Universo. No nosso caso, a mente humana seria um subsistema do organismo humano.

Podemos imaginar as relações bioquímicas do corpo humano configurando um sistema computável por algum programa P de uma máquina universal de Turing. Esse programa é capaz de rodar dentro dele outro programa E_M que emula uma submáquina de Turing U_M, tal que U_M é capaz de rodar um subprograma M que computa todos os programas que computam os processos mentais. Por sua vez, o subprograma M do subcomputador U_M é capaz de rodar dentro dele outro programa E_C que emula uma submáquina de Turing U_C, tal que U_C é capaz de rodar um subprograma C que computa a consciência, i.e., que computa todos os cômputos formados por sequências de estados mentais conscientes. Sob esse exemplo, a consciência humana seria emulada por outra emulação feita pelo organismo humano: seria um subsistema de outro subsistema.

Mas o presente trabalho mostra que, se for assim, é possível que o organismo humano seja capaz de computar funções que a consciência humana não consegue e, no caso acima, nem mesmo a mente. Por isso, pareceria que a biologia é muito mais complexa que qualquer computador definido ou programado por nós mesmos. Seguindo a mesma linha de pensamento da intuição matemática, podemos chamar esse fenômeno hipotético de **"paradoxo" da computabilidade biológica**. "De dentro" a biologia parece incomputável, mas "de fora" ela pode não ser.

O problema é que, dessa forma, estaríamos fadados a nunca conseguir emular um organismo – pelo menos, quase tão complexo quanto o humano – num computador tão poderoso quanto nós. Na verdade, não conseguiríamos que um programa suficientemente comprimido "desse conta do recado". Para resolver no consciente todos os problemas que as funções computadas pelo organismo consegue, precisaríamos ir aumentando cada vez mais o tamanho do subprograma [132] (ou ainda, da teoria científica). Nenhuma teoria científica elaborada por nós seria suficientemente abrangente. Isso casa perfeitamente com os resultados de incompletude e/ou incomputabilidade clássicos e nos faria suspeitar, com certeza, de que a biologia é de fato incomputável.

Mas quer dizer que os biólogos ou os esforços de criar **vida artificial** estariam fadados ao fracasso? Não necessariamente. Por quê? É preciso notar que quando estamos lidando com a subcomputação – mais ainda, do jeito que ela foi aqui definida –, o fato de existir um programa P maximamente comprimido num nível superior ao subcomputador que pode

[132] O suficiente para que ele escape de ser abrangido pela BB^+, por exemplo.

computar esse problema não quer dizer que o programa *P* não é um subprograma. Ele é, porém, não produz output para todos os valores de entrada. Ou, se produz, é um que representa o fato de seu cômputo não ter terminado, onde o tempo de cômputo não foi suficiente.

Portanto, uma hipotética teoria científica que seja suficiente para explicar o funcionamento de um organismo tão complexo quanto o nosso não poderia ser "rodada" em nossa consciência, pois o tempo de cômputo nesta é limitado. No entanto, nada impede que ela possa ser "rodada" em outro "hardware" que não seja nossa mente e que não esteja limitado pelo nosso poder de cômputo – quem sabe, talvez, um computador de silício usual suficientemente poderoso. Para conseguirmos emular a biologia "em nossas cabeças", as teorias científicas seriam sempre não complexas o bastante e pequenas para podermos prever todas as consequências dela conscientemente, mas não quer dizer que não possamos tomar conhecimento de uma teoria científica que possa ser complexa o suficiente em relação a outros "hardware". *Entender* é diferente de *prever*. *Saber* qual programa é diferente de *poder* rodá-lo. Essa diferença surge mais forte na discussão do item a seguir. Enfim, mesmo com as hipóteses sob as quais a subcomputação nos imporia limites ao conhecimento, emular um ser vivo em algum aparato artificial (por exemplo, um computador) ainda é uma possibilidade. Vida artificial e o "paradoxo" da computabilidade biológica podem conviver.

1.4.5 **O "paradoxo" da computabilidade do Universo**

Ainda podemos levar mais além essa linha de pensamento. E se o Universo – ou todas as leis da física – for um gigantesco computador dentro do qual tudo é um subcomputador seu? Entender os processos físicos como cômputos é também um dos grandes temas de discussão presentes na literatura atual.[133] Se assim for, podemos derivar que o Universo é um computador. Seguindo essa hipótese, já mostramos que pode ocorrer o fenômeno da subcomputação e a incomputabilidade relativa recursiva dentro de computadores e, por esse motivo, pode ocorrer dentro do Universo. Assim, temos carta branca para pensar na

[133] SZUDZIK, M. The Computable Universe Hypothesis. **arXiv**, Dezembro 2010.

possibilidade de qualquer subsistema físico em nosso Universo ser um subcomputador dele. Vamos partir dessa hipótese.

Por argumentos análogos ao já feito para a intuição matemática e para a computabilidade biológica, pode ser que o funcionamento dos processos físicos ou as leis da física pareçam complexos demais para qualquer teoria científica que elaboremos. Pode ser que estes nos pareçam incomputáveis. E, no entanto, não sê-lo de fato. Justamente, porque podemos ser subcomputadores de outro subcomputador, e assim por diante, até chegar ao nível máximo computacional do Universo. Por exemplo, pegue a sequência encaixante de subcomputadores exemplificada na computabilidade biológica e ponha o programa P como um subprograma de uma submáquina U_B, tal que U_B é emulável por um programa E_B rodado dentro de um programa P_F que computa todas as relações físicas. Aqui também valem todas as ressalvas feita para o problema 1.4.4.

Até aqui tudo que vale para o problema da computabilidade biológica vale para também para a computabilidade do Universo. Apenas estendemos o argumento. O que nos obriga a nomear um novo e totalmente análogo "paradoxo": o **pseudoparadoxo da computabilidade do Universo**.

Contudo, surge uma nova problemática. Supondo que todo subsistema do Universo seja um subcomputador, fica vetada a possibilidade de um dia acharmos uma teoria científica que possa emular o funcionamento do Universo em algum aparato inventado por nós. Isto, ao contrário, é permitido no caso da computabilidade biológica. Por que é diferente agora? Porque esse aparato qualquer, por exemplo, um computador eletrônico ou até mesmo um computador quântico, será também um subsistema/subcomputador, e, assim, estará sujeito às limitações da incomputabilidade relativa. O que impede que ele também consiga "rodar" nossa teoria científica escolhida. Não haverá, também, tempo de cômputo suficiente.

Portanto, sob essa nova hipótese computável do Universo, ficamos com duas opções nas mãos: ou nunca poderemos conhecer todas as leis da física, mas ainda podendo usar as que sabemos para prever, com a ajuda ou não de outros aparatos físicos, alguns fenômenos; ou, caso consigamos colocar todas essas leis numa teoria científica, nunca poderemos usá-la para prever todas as suas consequências e nem emular o nosso próprio Universo sob o qual vivemos – nem com a ajuda de outros aparatos físicos. Ficamos entre conhecer e prever. Nunca os dois ao mesmo tempo. Ou não conhecemos tudo, mas podemos prever sobre aquilo que conhecemos, ou conhecemos tudo e não conseguimos prever sobre aquilo que conhecemos. Essa disjunção exclusiva funcionaria como uma espécie de "**lei de ignorância**" para qualquer subsistema/subcomputador do computador maior: o Universo e suas leis. A este

resultado com caráter muito mais epistemológico que científico, guardadas as devidas hipóteses, chamaremos de **princípio de incerteza do Universo computável.** Logo explicaremos por quê.

Esse resultado tem um "cheiro" muito provocador. Ele nos lembra, por uma analogia vaga ou não, alguns outros resultados do mundo científico. Primeiramente, os próprios **teoremas de incompletude**. Estes dizem que não se pode ter um sistema axiomático formal, forte o suficiente para carregar grande parte da aritmética, que seja ao mesmo tempo consistente e completo. Isto é, que nunca gere duas sentenças contraditórias, mas que sempre gere uma das duas opções, para quaisquer sentenças. De novo, temos apenas duas opções: ou temos um sistema consistente, porém, com sentenças totalmente independentes (sobre as quais o sistema não sabe dizer que "sim" ou que "não"); ou temos um sistema completo, porém, que a qualquer momento gerará duas sentenças contraditórias.

O "paradoxo" da computabilidade do Universo deriva de um fenômeno básico na matemática: a incomputabilidade relativa recursiva. Além disso, como já falamos, a incomputabilidade clássica têm, em seu "coração", profunda semelhança intuitivo-matemática com os resultados de incompletude da aritmética (ou até mesmo da teoria dos conjuntos). São teoremas "irmãos", como já bem estabelecido na literatura. Logo, se o leitor perceber que a definição de incomputabilidade relativa é praticamente a mesma da incomputabilidade clássica, só que tomando como referência uma submáquina de Turing e não uma máquina universal de Turing, a relação íntima entre a incompletude e o princípio de incerteza do Universo computável fica bastante evidente. Mesmo assim, achamos que o assunto ainda pode pedir mais reflexões em investigações futuras.

A semelhança com o **princípio de incerteza de Heisenberg**, o qual proíbe que se saiba, com precisão, a posição e o momento linear de um observável subatômico também é tentadora. Aqui, também temos uma disjunção exclusiva: ou sabemos a posição, com uma margem de erro suficientemente pequena, mas ficamos impossibilitados de medir o momento linear com a precisão desejada; ou sabemos o momento linear, mas ficamos impossibilitados de medir a posição com a precisão desejada.

Contudo, é preciso dizer que, no caso da incompletude, parece ficar mais fácil fazer uma analogia entre: consistência e saber todas as leis da física com a completude e poder prever; ou consistência e poder prever com completude e saber todas as leis da física. Já no caso da incerteza da mecânica quântica, talvez possamos fazer essa ligação, porém, o próprio princípio de Heisenberg nos dá um resultado fundamental que proíbe qualquer tipo de aparato que possa prever completamente, em nível subatômico, o comportamento de um sistema

físico, pois precisaríamos saber com precisão suficiente a posição e o momento linear das partículas. Portanto, a opção de poder prever já ficaria vetada.

Se admitirmos um Universo computável com subcomputações dentro dele, o melhor que poderíamos fazer, de qualquer forma, seria conhecer todas as leis da física. Prever tudo já seria impossível. Uma forma ingênua, talvez, de especularmos sobre o fato seria: como qualquer aparato físico é um sistema composto de partículas subatômicas, este sistema só pode ser um subsistema do sistema que rege o comportamento de partículas subatômicas. Logo, qualquer sistema físico numa escala suficientemente grande seria um subcomputador de um computador que é capaz de rodar a mecânica quântica. Ao medir uma partícula subatômica, se experenciaria o fenômeno da incomputabilidade relativa recursiva. Não teríamos variáveis escondidas, teríamos informação inacessível. Além disso, sabemos que o fenômeno da incomputabilidade está intimamente ligado com o da aleatoriedade[134], por isso, somos tentados a especular sobre se essa explicação através da subcomputação não teria algo a ver com a natureza indeterminista descrita pela mecânica quântica, inerentemente descrita por ondas de probabilidades. Poderíamos pensar em **aleatoriedade relativa recursiva**, em que um número é aleatório em relação a um subcomputador, mas não o é em relação a um computador. Outro problema para o futuro.

Com certeza, toda essa discussão é ainda incipiente demais para tirarmos maiores conclusões. Pode ser que essas migrações conceituais não cheguem a lugar nenhum – o que achamos improvável. No entanto, acreditamos que urge uma investigação vindoura sobre isso.

Enfim, não estamos tentando aqui exaurir as explicações para essas questões mais filosóficas. Estamos nos atendo ao campo das hipóteses, por enquanto. Ademais, nem tivemos a oportunidade de defini-las melhor. De qualquer forma, pretende-se que fique claro o quão profícuo e inspirador pode ser essa ideia do pseudoparadoxo metalinguístico (sistema e subsistema) quando trazida para o campo da computabilidade.

Mesmo que os resultados matemáticos a seguir apresentem algum erro ou problema, as ideias que estão por trás são muito mais importantes e pungentes. Quem sabe, mesmo que esse trabalho falhe em provar a existência do "paradoxo" da computabilidade, alguém num futuro próximo conseguirá fazê-lo. É a nossa aposta. Esse fenômeno, depois de concebido,

---

[134] CALUDE, C. (Ed.). **Randomness and Complexity**. Singapore: World Scientific Publishing Co. Ptc. Ltd., 2007.

possui grande pregnância em nossa razão matemática, científica ou filosófica – se é que, em última instância, podemos separá-las.

2 DEMONSTRANDO A EVOLUÇÃO DE SUBPROGRAMAS

2.1 INTRODUÇÃO

Este capítulo vai se dedicar totalmente à demonstração da evolução dos subprogramas, já iniciada no capítulo 1. Porém, lá, nos dedicamos à introdução dos conceitos, às definições, a dar as ideias centrais da prova, e a *linká-los* aos problemas filosófico-científicos considerados mais relevantes. Agora, o leitor poderá verificar matematicamente o que foi construído até aqui – o que será feito em dois estágios. Até a seção 2.14, explicaremos em detalhes a prova. Na seção 2.15, deixaremos os lemas e teoremas que julgamos necessários para certificar o nosso resultado, caso se queira verificá-los ou apenas se aprofundar na matemática por trás do problema.

A demonstração da nossa versão do **projeto inteligente** pode ser vista no item 2.13 e a da **evolução cumulativa** no item 2.14, os quais são as últimas provas antes da seção 2.15. Escolhemos começar logo definindo a versão relativa da função **Busy-Beaver** e a mostrar como dela deriva diretamente – quase de modo trivial – o fenômeno de **incomputabilidade relativa recursiva**. Mais tarde, definiremos a versão relativa do número Ω (a *halting probability)* que chamaremos de **time limited halting probability**, denotada por $\Omega_{P'_T}$. Olhando $\Omega_{P'_T}$ em relação à definição de submáquina de Turing $U_{P'_T}$, que será dada logo após, pode-se observar que estão atrelados, da mesma forma que Ω e a máquina universal de Turing U.

Dito isto, começaremos a definir os programas necessários para construir a desejada submáquina $U_{P^{**}{}_T \circ P_T}$. Não há mistério sobre esses programas. Na verdade, eles podem ser facilmente programados. Assim, apenas indicamos declarativamente o algoritmo deles. Os programas P_T ou P'_T denotam programas arbitrários que computam funções totais quaisquer, respectivamente. A demonstração se baseia em tomar um programa P_T qualquer como base e sobre ele construir $P^{**}{}_T \circ P_T$, que também é total se P_T computar uma função total.

Poderemos notar que, para os nossos resultados, não faz diferença se essas funções são provadamente totais ou não,[135] basta que sejam computáveis e totais.

No item 2.11 está a definição do subcomputador $U_{P^{**}{}_T \circ P_T}$, o qual prescinde do subcomputador $U_{P^*{}_T \circ P'{}_T \circ P_T}$ definido no item 2.10. O subcomputador $U_{P^{**}{}_T \circ P_T}$ nada mais é que o subcomputador $U_{P^*{}_T \circ P'{}_T \circ P_T}$ quando este coloca $P^{**}{}_T \circ P_T$ no lugar de $P'{}_T$. Em outras palavras, nada mais é que quando $U_{P^{**}{}_T \circ P_T}$ chama a si mesmo no lugar de $P'{}_T$ em $U_{P^*{}_T \circ P'{}_T \circ P_T}$. Portanto, é entre essas duas submáquinas que fazemos a autorreferência para construir uma máquina que "se autodiagonaliza". E, por causa disso, aparece o cerne da questão que é provar que $P^{**}{}_T \circ P_T$ computa, de fato, uma função total, se P_T for total. Ou ainda, que $U_{P^{**}{}_T \circ P_T}$ é de fato uma submáquina se U_{P_T}também o for. Para isso, faremos a prova no item 2.12 e a repetiremos com um pouco mais de detalhes no teorema 2.15.18.

Com essas demonstrações em mãos, acreditamos que já seja suficiente para o leitor "amarrar o laço", mencionado na seção 1.3, e finalizar a prova nos itens 2.13 e 2.14. Esperamos que não seja tortuoso ler nossas demonstrações apesar da notação carregada. O que, por sinal, já é algo comum em teoria algorítmica da informação ou teoria da computação. Tentamos introduzir o mínimo possível de notações novas. Muitas delas, juntamente com outras definições, já estão no capítulo 1. Dessa forma, a leitura do capítulo anterior é premissa essencial para entendermos este.

2.2 A FUNÇÃO *BUSY-BEAVER* PLUS

Seja $P'{}_T$ uma função total. Vamos definir a função $BB^+{}_{P'{}_T}(N)$, a função *Busy-Beaver Plus*, pelo seguinte processo:

(i) Gera uma lista de todos os outputs de $U_{P'{}_T}(w)$ tal que $|w| \leq N$;

(ii) Toma o maior número dessa lista;

(iii) Soma 1;

---

[135] Para mais sobre funções provadamente totais, ver: DORIA, F. A.; CARNIELLI, W. A. Are the Foundations of Computer Science Logic–Dependent? **Dialogues, Logics and Other Strange Things**, 20 Outubro 2008.

(iv) Retorna esse valor.

O próprio nome dessa função já foi pensado para lembrar o nome da função *Busy-Beaver* e, consequentemente, não é por coincidência que as duas têm quase a mesma definição. Se tirarmos o passo (iii), ela se torna exatamente a função Busy-Beaver para subcomputadores, aqui denotada por $BB_{P'_T}(N)$.

De maneira inversa, também podemos definir a função *Busy-Beaver Plus* para máquinas universais de Turing U, denotando-a por $BB^+(N)$, apenas trocando $U_{P'_T}(w)$ no passo (i) por $U(w)$. Portanto, já deve ficar claro que:

$$BB^+(N) = BB(N) + 1$$

e

$$BB^+{}_{P'_T}(N) = BB_{P'_T}(N) + 1$$

Mas por que usar a função BB^+ ao invés da BB? Essa deve ser, supomos, a pergunta imediata do leitor. Como estamos lidando com submáquinas de Turing e P'_T é arbitrário, pode existir um programa de P'_T, com N como input, que computa o maior valor dado por qualquer outro programa de P'_T que tenha tamanho $\leq N$. Quando estamos lidando com uma máquina universal de Turing U, isso não pode acontecer. Porém, com submáquinas pode. Então, a função $BB^+{}_{P'_T}$ é acionada para nos garantir que ela mesma não seja computável por nenhum programa de $U_{P'_T}$, apesar de sê-lo por algum programa de U, como veremos adiante. A *Busy-Beaver* contém a ideia do maior output de qualquer programa de tamanho $\leq N$; já a *Busy-Beaver Plus* contém a ideia de superar, ao menos por 1, qualquer programa de tamanho $\leq N$. Respectivamente, a primeira nos dá uma maximização e a segunda, uma superação mínima.

Seguindo essa linha de pensamento, para simbolizar essa função podemos evocar outra vez a imagem do homem sentado num burro guiando o animal com uma cenoura presa na ponta de sua vara de pesca. Não importa o quanto o burro ande em direção à cenoura, ela sempre estará à mesma distância dele, inatingível. Porém, isso não acontece porque a distância do burro e a cenoura é infinita, mas sim porque toda vez que o burro anda, a cenoura anda junto, sempre se mantendo a sua frente. Por mais crescente que seja a função P'_T, o

programa de U que computa $BB^{+}{}_{P'_T}(N)$ apenas se baseia no que $U_{P'_T}$ fez para superá-lo o mínimo possível.

É importante o leitor notar que, analogamente à *Busy-Beaver*, a $BB^{+}{}_{P'_T}(N)$ serve para medir a criatividade ou a *sub-program-size complexity* dos subprogramas em relação ao subcomputador P'_T. Por quê? Pela própria definição da mesma, se um subprograma gera um output $\geq BB^{+}{}_{P'_T}(N)$, ele precisa ter obrigatoriamente um tamanho $> N$. Ele precisa ter mais de N bits de informação incompressível, ou seja, mais de N bits de criatividade. Claro, podemos construir um programa que compute a função $BB^{+}{}_{P'_T}(N)$ – caso a função P'_T seja computável. O que permitiria que existisse um programa de tamanho bem menor que N – por exemplo, de tamanho $\leq C + (1+\epsilon)\log_2 N$. Mas isso não é uma contradição, pois, esse programa nunca poderá ser um subprograma de $U_{P'_T}$, isto é, nunca poderá ser um programa que rode em tempo de cômputo dado por P'_T. Se for, entrará em contradição direta com a definição de $BB^{+}{}_{P'_T}$, tornando o programa P'_T não definido para qualquer input, isto é, uma função não total.

Podemos dizer, sem dúvida, que essas duas funções têm um espírito diagonalizador, vindo de Cantor, mas, a BB^{+} tem, como explicado acima, um poder diagonalizador um pouco maior que a BB. Maior o suficiente para evitar que ela não seja computável por qualquer submáquina de Turing que a defina. Portanto, as duas crescem com extrema rapidez, mas a *Busy-Beaver Plus* cresce um pouco mais rápido que a *Busy-Beaver*.

Mais à frente o leitor poderá constatar que essa função é a mesma que a dada pela função computável $U\left(\pi'_\Omega \circ P'_T \circ 0^{|U(P_\Sigma \circ P'_T \circ N)|}1 \circ U\left(P_\Sigma \circ P'_T \circ N\right)\right)$.[136] Isso é proposital, pois faz com que π'_Ω e $BB^{+}{}_{P'_T}(N)$ se assemelhem bastante a π_Ω e $BB'(N)$, respectivamente.

2.3 INCOMPUTABILIDADE RELATIVA RECURSIVA

Antes de continuarmos com a demonstração da evolução, vamos analisar mais um pouco a função $BB^{+}{}_{P'_T}(N)$, provando um resultado crucial e simples que rege todo o espírito

[136] De novo, P'_T precisa ser uma função computável total.

deste trabalho. Podemos dizer que este traduz quase tudo que queremos dizer por *incomputabilidade relativa*.

Seja P'_T uma função total e $U_{P'_T}$ uma submáquina de Turing. Então, podemos provar que a função $BB^+{}_{P'_T}(N)$ é **relativamente incomputável** por qualquer programa de $U_{P'_T}$. Mas o que quer dizer ela ser **relativamente incomputável**? Podemos dizer que tal coisa se deve ao fato de não existir um programa que compute o resultado final da função em questão para qualquer valor que o apliquemos. Usando termos mais matemáticos, não existe um programa que, para qualquer elemento do domínio da função, retorne o respectivo elemento da imagem. Ou ainda: não existe programa que, para todo input N, o output correspondente seja igual a $BB^+{}_{P'_T}(N)$. É verdade que alguns programas podem calcular uma quantidade enorme de valores de N, mas nunca para todo N – em algum momento ele irá falhar, retornar uma resposta errada ou nunca retornar resposta alguma.

Uma forma mais intuitiva de entender o que se passa é buscar um programa e "colocar ao seu lado" o seu input, como por exemplo, $U_{P'_T}(P * N)$. Onde $*$ denota uma concatenação qualquer, e não necessariamente a "concatenação" "$\circ$" com a qual estamos trabalhando aqui. Na verdade, isso vale para qualquer maneira de comprimir a informação de P e N num programa qualquer (ou subprograma). Portanto, ele pode não ter a forma $P * N$. De todo modo, vamos provar o teorema para a "concatenação" – já mencionada neste parágrafo – que importa para nós. Fica a cargo do leitor demonstrar para qualquer forma de informar N como input de P, bastando para tal notar que $H\left(U_{P'_T}(P * N)\right) \leq |P \circ N|$.

Vamos nos valer da mesma ideia utilizada na demonstração do teorema de incompletude de Chaitin, sendo desta vez para mostrar uma incomputabilidade – uma incomputabilidade relativa à submáquina $U_{P'_T}$.

Quando N é dado como input a um programa P qualquer, ele vem na sua forma comprimida com tamanho $\cong H(N)$[137], donde $|P \circ N| \cong C + H(N)$. Mas, como já sabemos pela AIT, para qualquer constante C, existe um N_0 suficientemente grande tal que $C + H(N_0) < N_0$. Logo, pela definição de $BB^+{}_{P'_T}$, o output de $P \circ N_0$, quando rodado pela submáquina $U_{P'_T}$, vai ser levado em conta quando calcularmos a $BB^+{}_{P'_T}(N_0)$. Assim, obrigatoriamente,

[137] Na verdade, vamos usar a propriedade de que $|P \circ N| \leq C + |P| + C' + log_2 N + (1 + \epsilon)\, log_2(log_2 N)$.

$$BB^{+}{}_{P'_T}(N_0) \geq U_{P'_T}(P \circ N_0) + 1 > U_{P'_T}(P \circ N_0)$$

O que nos levará a uma contradição, caso P compute $BB^{+}{}_{P'_T}$ quando rodado pela submáquina $U_{P'_T}$.

Também, segundo o mesmo argumento, pode-se demonstrar trivialmente que $BB^{+}{}_{P'_T}(N)$ é uma função **relativamente incompressível** por qualquer subprograma de tamanho menor ou igual a N. Isto é, a partir de certo tamanho N_0, nenhum programa de tamanho $N \geq N_0$ quando rodado por P'_T dará valor maior ou igual a $BB^{+}{}_{P'_T}(N)$.

- Demonstração:

Queremos demonstrar que:

$$\forall P \exists N_0 \forall N (\, N \geq N_0 \;\rightarrow\; U_{P'_T}(P \circ N) < BB^{+}{}_{P'_T}(N))$$

Vamos começar tomando um programa P arbitrário.
Pela definição da linguagem L e de U, temos que para todo N:

$$|P \circ N| \leq C + |P| + C' + \log_2 N + (1+\epsilon)\log_2(\log_2 N)$$

Seja $C + |P| + C' = C''$.
Sabemos que $\exists N_0 \forall N (\, N \geq N_0 \;\rightarrow\; C'' + \log_2 N + (1+\epsilon)\log_2(\log_2 N) < N)$.
Logo, $\exists N_0 \forall N (N \geq N_0 \;\rightarrow\; |P \circ N| < N)$.
Consequentemente, pela definição de $BB^{+}{}_{P'_T}$, teremos que:

$$\exists N_0 \forall N (\, N \geq N_0 \;\rightarrow\; U_{P'_T}(P \circ N) < U_{P'_T}(P \circ N) + 1 \leq BB^{+}{}_{P'_T}(N))$$

■

2.4 O PROGRAMA $P_{\sum}$

Vamos entender outro programa crucial para nós.

Seja $P_{\sum}$ o programa que pega $P'_T \circ N$ como input e calcula a soma binária de todas as probabilidades algorítmicas dos programas de L com tamanho $\leq N$ que param em tempo de cômputo dado por P'_T (ou, se preferir, que $U_{P'_T}(w) \neq l_1$).[138] Se $N = 0$, então ele retorna 0.

Caso P'_T não seja uma função total (isto é, não pare para algum input w), $P_{\sum}$ também não estará definido para todos os inputs. Caso P'_T seja, então, obviamente, $P_{\sum}$ será.

Vale lembrar novamente que estamos numa linguagem autodelimitada. Logo, o output de $P_{\sum}$ sempre será um número real binário entre 0 e 1.

Mas o que acontece se aumentarmos o número N cada vez mais? Como a soma de todos os programas de L é 1, existirá o limite de $U(P_{\sum} \circ P'_T \circ N)$ quando N tende para o infinito (contanto que P'_T seja uma função total). O que esse limite nos diz?

2.5 O NÚMERO DE CHAITIN RELATIVO

Pela construção do programa P_{SM}, é possível notar facilmente que $P_{\sum}$ calcula a soma de todos os programas de L que param em tempo de cômputo dado por P'_T. Com isso, obtemos uma analogia direta com o número Ω de Chaitin: conseguimos um número que é a soma de todos os programas que param, porém, agora, em tempo de cômputo limitado por uma função computável. O que nos leva a chamar esse número de *time limited halting probability*.

Seja P'_T uma função total.

Denotamos por $\Omega_{P'_T}$ a ***time limited halting probability*** dada por:

---

[138] Ver definição 2.15.5.

$$\Omega_{P'_T} = \sum_{p \text{ é um programa de } L \text{ que se detém em tempo de cômputo} \leq U(P'_T \circ p).} 2^{-|p|}$$

Em outras palavras, p é contado nessa soma se, e somente se, existe o valor de $U(p)$ e, para todo y e x, se $U(T \circ p) = x$ e $U(P'_T \circ p) = y$, então $x \leq y$. Dessa forma, quanto mais crescente for a função dada pelo programa P'_T, mais próximo de Ω chegamos. Além disso, excluímos de vez os programas p que não param: depois de y cômputos, se o programa p não tiver parado ainda, ele só pode nunca parar ou parar em tempo x com $x > y$.[139]

Na verdade, não precisamos realmente da condição de que P'_T seja total. Se $U(P'_T \circ p)$ não estiver definido, de qualquer forma, $U(p)$ precisa estar definido, então p estará incluso em $\Omega_{P'_T}$. Por exemplo, se P'_T for um programa que não se detém qualquer input, então

$$\Omega_{P'_T} = \sum_{p \in L.} 2^{-|p|} = \Omega$$

Trata-se apenas de uma formalidade de um caso extremo. Para nós, importará somente o caso em que P'_T é, de fato, total. Além disso, se P'_T não for total, $\Omega_{P'_T}$ poderá ser um número incomputável.

2.6 SUBMÁQUINAS DE TURING

Devemos definir, então, uma submáquina de Turing usando essa noção de tempo de cômputo limitado. Iremos fazer da maneira mais intuitiva, aproveitando nossa definição bem geral de submáquinas.

Seja P_T um programa arbitrário que calcula o tempo máximo de cômputo para um programa w qualquer. Ou seja, P_T pode ser uma função total qualquer.

---

[139] E isso é decidível.

Então, existe uma submáquina de Turing definida pela função de tempo de cômputo P_T. Ou seja, existe ${}^{U}/_{P_{SM} \circ P_T}$. Mas como a programamos?

P_{SM} é um programa que recebe P_T e w como inputs, roda $U(P_T \circ w)$ e retorna:

(i) l_1, se $U(w)$ não parar em tempo de cômputo $\leq U(P_T \circ w)$;
(ii) l_{k+1}, se $U(w)$ parar em tempo de cômputo $\leq U(P_T \circ w)$ e $U(w) = l_k$;

Esse programa define um subcomputador (uma submáquina de Turing) que nos retorna um símbolo conhecido (no caso, 0[140]) quando um programa de w não parar em tempo $\leq U(P_T \circ w)$ ou retorna o mesmo output (a menos de uma bijeção trivial) de $U(w)$ quando este parar em tempo $\leq U(P_T \circ w)$.

Para ser uma submáquina de Turing, ela precisa estar definida para todas os inputs. Isso ocorre pelo fato de P_T ser total.

Vamos denotar somente por U_{P_T} a submáquina de Turing ${}^{U}/_{P_{SM} \circ P_T}$, de tal modo que:

$$\forall w(\, U_{P_T}(w) = {}^{U}/_{P_{SM} \circ P_T}\,(w) = U(P_{SM} \circ P_T \circ w))$$

Diremos que uma função ou um número é x-computável se ele for computável pelo computador U_x. Assim como um 0-programa é um programa quando rodado por U, um x-programa é um programa quando rodado por U_x.

2.7 OS ORGANISMOS/SUBPROGRAMAS $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$

Primeiramente, expliquemos o programa $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$. O que esse programa faz? Ele é quase o mesmo programa π_Ω usado por Chaitin em seus modelos iniciais. Porém,

[140] Lembre da definição 2.15.5.

ele toma como referência a submáquina de Turing $U_{P'_T}$ ao invés da máquina U. E mais: ele não retorna índices de aproximações a $\Omega_{P'_T}$, mas sim aproximações a $BB^+(N)$.

Enfim, π'_Ω lê a *bit string* $P'_T \circ 0^{|\rho|}1 \circ \rho$ e soma todas as probabilidades algorítmicas dos programas de tamanho $\leq n$ que param em tempo de cômputo dado por P'_T.[141] Começa com $n = 1$ e continua até k tal que a soma de todas as probabilidades algorítmicas de todos os programas de tamanho $\leq k$ se iguale ou ultrapasse o valor de ρ. Então, calcula o maior output desses subprogramas de tamanho $\leq k$, soma 1 a ele e retorna esse valor. Se $\rho = 0$, então ele retorna 0.

Note que se $\rho > \Omega_{P'_T}$, esse programa nunca parará. Caso $\rho = \Omega_{P'_T}$ – o que seria uma possibilidade ideal –, π'_Ω também poderá não parar. Contudo, por outro motivo. Nesse caso, π'_Ω nunca terminaria de ler seu input, pois $\Omega_{P'_T}$ pode ter tamanho infinito.

Daí podemos dizer que π'_Ω pega uma aproximação finita e inferior a $\Omega_{P'_T}$ e calcula um valor que supera o valor de qualquer output produzido por qualquer programa envolvido[142] na soma que resulta na aproximação inferior em questão.

Então, de maneira quase direta, é possível mostrar que

$$U\left(\pi'_\Omega \circ P'_T \circ 0^{|\rho'|}1 \circ \rho'\right)$$

sempre para (ou está bem definido) quando $\rho' \leq U\left(P_\sum \circ P'_T \circ N\right)$, para qualquer valor de N. E não importa se $|\rho'| > |U\left(P_\sum \circ P'_T \circ N\right)|$, pois estamos lidando como números reais entre 0 e 1. Por exemplo, os zeros à direita não afetam o fato de $\rho' \leq U\left(P_\sum \circ P'_T \circ N\right)$. Isso será importante para as demonstrações seguintes.

2.8 O SUBPROGRAMA P'_M

Voltando-nos para as mutações, queremos que o programa $P''_M \circ P'_T \circ k \circ P'_M \circ P'_T \circ w$ seja rodado em $U_{P^*_T \circ P'_T \circ P_T}$. Esse programa será nossa mutação M_k, onde w é o organismo

[141] Ou seja, de todo subprograma com tamanho $\leq n$ de $U_{P'_T}$ cujo output seja diferente de l_1.
[142] Por ordem crescente de tamanho.

anterior a ser mutado. Para chegarmos aí, vamos dividi-lo em dois: ${P'}_M \circ {P'}_T \circ w$ será input de ${P''}_M \circ {P'}_T \circ k$. Ambos, separados ou acoplados, serão permitidos em $U_{{P^*}_T \circ {P'}_T \circ P_T}$, como mostraremos a seguir.

O programa ${P'}_M$ recebe ${P'}_T \circ w$ como input, lê a si mesmo e, então, roda $U\left(P_\sum \circ {P'}_T \circ n\right)$ para $n = 1,2,\ldots$ testando em cada caso se $U\left({\pi'}_\Omega \circ {P'}_T \circ 0^{|U(P_\sum \circ {P'}_T \circ n)|} 1 \circ U\left(P_\sum \circ {P'}_T \circ n\right)\right) > U_{{P'}_T}(w)$. Quando achar o primeiro n que isso seja verdadeiro, toma o valor $N = n - 1$ e constrói $U\left(P_\sum \circ {P'}_T \circ N\right)$. Então, retorna ${\pi'}_\Omega \circ {P'}_T \circ 0^{|U(P_\sum \circ {P'}_T \circ N)|} 1 \circ U\left(P_\sum \circ {P'}_T \circ N\right)$.

Além disso, pela definição da função ${BB^+}_{{P'}_T}$, teremos que

$$U\left({\pi'}_\Omega \circ {P'}_T \circ 0^{|U(P_\sum \circ {P'}_T \circ N)|} 1 \circ U\left(P_\sum \circ {P'}_T \circ N\right)\right) = {BB^+}_{{P'}_T}(N) \leq U_{{P'}_T}(w)$$
$$< U\left({\pi'}_\Omega \circ {P'}_T \circ 0^{|U(P_\sum \circ {P'}_T \circ N+1)|} 1 \circ U\left(P_\sum \circ {P'}_T \circ N+1\right)\right)$$

Utilizaremos esse resultado mais adiante justamente para mostrar que a sequência de mutações $M_1, M_2, \ldots M_k$ sempre nos leva a uma aptidão $\geq {BB^+}_{{P'}_T}(k)$.

É claro que se ${P'}_T$ for total, ${P'}_M$ também será. Por quê? Basta perceber que quando ${P'}_M$ testar para $n = |w|$ obrigatoriamente o output de ${\pi'}_\Omega \circ {P'}_T \circ 0^{|U(P_\sum \circ {P'}_T \circ n)|} 1 \circ U\left(P_\sum \circ {P'}_T \circ n\right)$ será maior que $U_{{P'}_T}(w)$. Portanto, $N < |w|$. O que nos diz que ${P'}_M$ só precisa testar os programas de tamanho $\leq |w| \leq |{P'}_M \circ {P'}_T \circ w| - 1$ para retornar um valor.[143]

Por isso, note também que se ${P'}_T$ estiver definido para todo w' tal que $1 \leq |w'| \leq |{P'}_M \circ {P'}_T \circ w| - 1$, então $U({P'}_M \circ {P'}_T \circ w)$ também estará bem definido – fator de suma importância mais adiante.

2.9 A MUTAÇÃO/SUBPROGRAMA ${P''}_M$

[143] Lembre-se da definição da linguagem *L*.

Já o programa P''_M recebe P'_T, k e w como inputs, onde k é um número natural[144], e roda $U_{P'_T}(w)$, obtendo y tal que $y = U_{P'_T}(w)$. Em seguida, cai numa das duas opções:

(i) Caso y seja da forma $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$, onde ρ é um número real positivo binário entre zero e um, P''_M soma $1/2^k$ ao número ρ gerando outro número ρ''. Especialmente se $\rho = 0$, então apenas toma $\rho'' = U\left(P_\Sigma \circ P'_T \circ (1)\right)$. Depois, adiciona uma quantidade suficiente (ou nenhum) de zeros à direita de ρ'' até que $\left|\pi'_\Omega \circ P'_T \circ 0^{|\rho''\circ 0 \ldots 0|}1 \circ \rho'' \circ 0 \ldots 0\right| - 1 \geq U(P_{MS} \circ P'_T \circ |\rho|)$ e $\left|\pi'_\Omega \circ P'_T \circ 0^{|\rho''\circ 0 \ldots 0|}1 \circ \rho'' \circ 0 \ldots 0\right| - 1 \geq U(P_{MS} \circ P'_T \circ k)$. Por fim, retorna $\pi'_\Omega \circ P'_T \circ 0^{|\rho''\circ 0 \ldots 0|}1 \circ \rho'' \circ 0 \ldots 0$.

(ii) Caso y não seja dessa forma, retorna $U_{P'_T}(w)$.

Note que, pela definição da linguagem *L*, existe um programa que sempre decide se qualquer programa se encaixa no caso (i) ou no caso (ii). Logo, caso P'_T seja uma função total, sempre haverá um y tal que $y = U_{P'_T}(w)$. Note também que k é informado no input e $|\rho|$ é informado por y.

Assim, sempre se pode calcular o output do programa P_{MS}.[145] Por quê? Apesar de ele estar em função de P'_T, o cômputo de P'_T não importa em nada para o ouput de P_{MS}. O que importa é somente a própria *bit string* P'_T. Isso só é possível pelo fato de a linguagem *L* ser computável e completa. Dessa forma, o somatório de todos os programas que param em tempo dado por $P^*_T \circ P'_T \circ P_T$, caso P'_T seja total, vai ser menor que 1 e tenderá a um número real no limite quando levarmos em conta todos os programas. As propriedades do limite farão com que o programa P_{MS} se detenha sempre, independentemente de P'_T ser total ou não.

Por isso, $P''_M \circ P'_T \circ k \circ w$ sempre parará. Além disso, basta que P'_T esteja definido para w para que $U(P''_M \circ P'_T \circ k \circ w)$ esteja bem definido.

Não é coincidência que P''_M se acople com P'_M. Os dois programas foram concebidos para esse fim. Para visualizar isso, substitua $P'_M \circ P'_T \circ w$ no lugar de w em $P''_M \circ P'_T \circ k \circ$

---

[144] Lembre que não há problema em se admitir números naturais como inputs, vide a notação 2.15.1.d).
[145] Ver teorema 2.15.13.

w gerando o programa $P''_M \circ P'_T \circ k \circ P'_M \circ P'_T \circ w$. Esse programa composto pega um programa w de $U_{P'_T}$ e leva num outro da forma $\pi'_\Omega \circ P'_T \circ 0^{|\rho'|}1 \circ \rho'$, onde $\rho' = \rho'' \circ 0 \ldots 0 = \rho'' = {}^{1}/_{2^k} + \rho$ e $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$ sempre calcula um número maior que $U_{P'_T}(w)$, caso $\rho' \leq \Omega_{P'_T}$.

Novamente, o leitor deve estar se perguntando o porquê dos zeros à direita. Eles servem para aumentar o tamanho de $\pi'_\Omega \circ P'_T \circ 0^{|\rho'|}1 \circ \rho''$ até um ponto em que $P^*_T \circ P'_T \circ P_T$ – o qual iremos definir logo abaixo – deixa que ele rode o tempo que for necessário se, e somente se, $\rho' \leq \Omega_{P'_T}$. Isso pode ser melhor averiguado no lema 2.15.16.

2.10 A SUBMÁQUINA $U_{P^*_T \circ P'_T \circ P_T}$

Agora, vamos definir a submáquina de Turing $U_{P^*_T \circ P'_T \circ P_T}$. Vale lembrar novamente que P'_T e P_T são duas funções totais, por suposição. Isso se manterá até chegarmos na submáquina $U_{P^{**}_T \circ P_T}$, onde P_T ainda continua total por suposição, mas no lugar de P'_T entrará o próprio $P^{**}_T \circ P_T$.

Lembre também do programa T que recebe w como input e retorna o tempo de cômputo de $U(w)$. Obviamente, ele só está definido para aqueles inputs w tais que w são programas que param. Porém, isso não afetará o programa P^*_T abaixo porque só precisaremos rodar o programa T quando w for um programa que sempre para.

O programa P^*_T recebe P'_T, P_T e w como inputs e retorna:

(i) $U(T \circ w)$, se w for da forma $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$ e, além disso, $U\big(P_\Sigma \circ P'_T \circ (|\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big) \geq \rho$ e $U(T \circ w) > U(P_T \circ w)$;

(ii) $U(T \circ w)$, se w for da forma $P'_M \circ P'_T \circ w'$ e $U(T \circ w) > U(P_T \circ w)$;

(iii) $U(T \circ w)$, se w for da forma $P''_M \circ P'_T \circ k \circ w'$ e $U(T \circ w) > U(P_T \circ w)$;

(iv) $U(P_T \circ w)$, caso não se encaixe nos casos (i), (ii) e (iii);

Aqui, estamos montando um programa sobre duas funções totais: P'_T e P_T. O primeiro passo para verificar se P^*_T é total é notar que existe um programa que decide sempre se w entra no caso (i), (ii), (iii) ou (iv). Esse programa precisa primeiro verificar a forma[146] do programa w, uma vez que π'_Ω, P'_T, P'_M e P''_M já podem ser conhecidos por P^*_T. Além disso, pelas definições de π'_Ω, P'_M e P''_M e pela suposição de P'_T já ser uma função total, teremos que $U(T \circ w)$ estará sempre bem definido sobrando apenas decidir se $U(T \circ w) > U(P_T \circ w)$, isto é, outra coisa decidível. O caso (iv) sai direto do fato de P_T também ser total.

É importante prestar atenção também ao fato de que $P^*_T \circ P'_T \circ P_T$ é um tempo de cômputo suficiente para computar qualquer coisa que U_{P_T} computa. No caso (iv) é óbvio. Caso seja um programa da forma (i), (ii) ou (iii), a condição $U(T \circ w) > U(P_T \circ w)$, que está sempre presente, nos garante que $P^*_T \circ P'_T \circ P_T$ sempre seja uma extensão de P_T, nunca uma restrição.

Resumindo o que $P^*_T \circ P'_T \circ P_T$ faz: no caso (i), $U_{P^*_T \circ P'_T \circ P_T}$ permite um programa da forma $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$ que calcula um valor sempre superior a qualquer output de qualquer programa de $U_{P'_T}$ com tamanho $\leq |\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho| - 1$; no caso (ii), $U_{P^*_T \circ P'_T \circ P_T}$ recai no caso (i); o caso (iii) sempre modifica ρ caso o output do programa w em $U_{P'_T}$ tenha um output da forma $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$, caso contrário, o caso (iii) retorna o próprio w.

É claro que, como para todo w , $U(P_T \circ w) \leq U(P^*_T \circ P'_T \circ P_T \circ w)$, então $\Omega_{P^*_T \circ P'_T \circ P_T} \geq \Omega_{P_T}$. Contudo, e se quisermos que $U\left(P_\sum \circ P'_T \circ N\right)$ seja sempre uma aproximação ao próprio $\Omega_{P^*_T \circ P'_T \circ P_T}$ ao invés de ser uma aproximação a $\Omega_{P'_T}$? Isto é, que no caso (i), $U_{P^*_T \circ P'_T \circ P_T}$ permita um programa $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$ calcular um valor sempre superior a qualquer output de qualquer programa do próprio $U_{P^*_T \circ P'_T \circ P_T}$ com tamanho $\leq |\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho| - 1$? Qual programa colocar no lugar de P'_T? Ele existe? Aqui ficará evidente o emprego da autorreferência, um elemento "diagonalizador" dentro do próprio programa, para construir o próprio programa P^{**}_T.

---

[146] Lembre-se da definição da linguagem *L*.

2.11 A SUBMÁQUINA $U_{P^{**}{}_T \circ P_T}$

O programa $P^{**}{}_T$ recebe P_T e w como inputs, lê a si mesmo, a P_T e a w, monta[147] $P^{*}{}_T \circ P^{**}{}_T \circ P_T \circ P_T \circ w$ e roda $U(P^{*}{}_T \circ P^{**}{}_T \circ P_T \circ P_T \circ w)$, retornando:

(i) $U(T \circ w)$, se w for da forma $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$ e, além disso, $U\big(P_\Sigma \circ P^{**}{}_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big) \geq \rho$ e se $U(T \circ w) > U(P_T \circ w)$;

(ii) $U(T \circ w)$, se w for da forma $P'_M \circ P^{**}{}_T \circ P_T \circ w'$ e se $U(T \circ w) > U(P_T \circ w)$;

(iii) $U(T \circ w)$, se w for da forma $P''_M \circ P^{**}{}_T \circ P_T \circ k \circ w'$ e se $U(T \circ w) > U(P_T \circ w)$;

(iv) $U(P_T \circ w)$, caso não se encaixe nos casos (i), (ii) e (iii);

A grande questão está em saber se esse programa é uma função total. Não será tão simples como nos casos anteriores. O caso (iv) permanece trivial, visto que só depende de P_T. Nos outros, o problema é que, quando introduzimos uma autorreferência, podemos cair num programa que entra em loop. Para provar que esse "referir a si mesmo" dentro de $P^{**}{}_T$ nunca gera um loop eterno, para qualquer w, vamos fazer uma prova por indução.

Note que, como π'_Ω, P'_M e P''_M são predeterminados, eles têm um tamanho mínimo e, logo, existirá um tamanho máximo para w tal que $U(P^{**}{}_T \circ P_T \circ w)$ nunca cairá nos casos (i), (ii) ou (iii). Com isso, como P_T é dado como total, $P^{**}{}_T \circ P_T$ sempre estará bem definido para esses w's suficientemente pequenos.

Para finalizar a prova por indução bastará verificar que, se $P^{**}{}_T \circ P_T$ está bem definido para qualquer w de tamanho $\leq k$, então $P^{**}{}_T \circ P_T$, estará bem definido para qualquer w de tamanho $k + 1$. É isso que faremos no teorema abaixo.

Se conseguirmos demonstrar que $P^{**}{}_T \circ P_T$ é total, conseguiremos montar uma $BB^{+}{}_{P^{**}{}_T \circ P_T}(N)$ e um $\Omega_{P^{**}{}_T \circ P_T}$, de forma que $BB^{+}{}_{P^{**}{}_T \circ P_T}(N)$ é tão incomputável em relação a $U_{P^{**}{}_T \circ P_T}$ quanto $BB(N)$ é em relação a U. O que é análogo ao que acontece quando estamos lidando com U e $BB'(N)$, correspondentemente no lugar de $U_{P^{**}{}_T \circ P_T}$ e $BB^{+}{}_{P^{**}{}_T \circ P_T}(N)$. Note-se que $BB^{+}{}_{P^{**}{}_T \circ P_T}(N)$ também servirá para medir a complexidade dos programas de $U_{P^{**}{}_T \circ P_T}$:

[147] Na verdade, $P^{**}{}_T \circ P_T$ é posto como se fosse um programa só no lugar de $P'{}_T$ dentro de $P^{*}{}_T \circ P'{}_T \circ P_T$. A forma correta de simbolizar isso seria $P^{*}{}_T \circ (P^{**}{}_T \circ P_T) \circ P_T \circ w$. Porém, por razões práticas, vamos no abster dos parênteses.

o output (ou aptidão) igual ou superior a $BB^{+}{}_{P^{**}{}_{T}\circ P_T}(N)$ não pode ser atingido por nenhum subprograma de tamanho $\leq N$.

Outra propriedade essencial que conseguimos reproduzir é que existem programas de $U_{P^{**}{}_{T}\circ P_T}$ que usam aproximações de $\Omega_{P^{**}{}_{T}\circ P_T}$ para se aproximar, cada vez mais, de $BB^{+}{}_{P^{**}{}_{T}\circ P_T}(N)$. É como se a máquina que computa a função $BB^{+}{}_{P^{**}{}_{T}\circ P_T}(N)$ fosse uma máquina de Turing oracular de primeira ordem, **só que agora em relação a $U_{P^{**}{}_{T}\circ P_T}$.**

Colocamos P_T como input de $P^{**}{}_{T}$ somente para deixar claro que cada P_T define uma função $P^{**}{}_{T}\circ P_T$ diferente que calcula o tempo de cômputo a partir de w. Mas, se o leitor quiser, ou achar mais fácil de formalizar a prova, pode deixar P_T incluso (e não como um input) na programação de $P^{**}{}_{T}$ e $P^{*}{}_{T}$. Dessa forma, $P^{**}{}_{T}$ passaria a rodar $U(P^{*}{}_{T}\circ P^{**}{}_{T}\circ w)$ ao invés de $U(P^{*}{}_{T}\circ P^{**}{}_{T}\circ P_T\circ P_T\circ w)$.

2.12 A SUBMÁQUINA $U_{P^{**}{}_{T}\circ P_T}$ ESTÁ BEM DEFINIDA?

Seja P_T uma função total.
Então, para qualquer $w \in L$, $U(P^{**}{}_{T}\circ P_T\circ w)$ é um valor bem definido.
Ou seja, $U(P^{**}{}_{T}\circ P_T\circ w)$ é uma função total em w.

– Demonstração:

a) Caso válido para todo w tal que $|w| \leq k_0$:

Como π'_{Ω}, P'_{M} e P''_{M} são programas conhecidos, eles possuem um tamanho (dentro da linguagem L de nossa escolha). Além disso, conhecemos também o programa $P^{**}{}_{T}\circ P_T$, o qual também terá, por isso, um tamanho.

Logo, existe um k_0 tal que, para qualquer *bit string* w com $|w| \leq k_0$, w nunca será da forma $\pi'_{\Omega}\circ P^{**}{}_{T}\circ P_T\circ 0^{|\rho|}1\circ\rho$ ou $P'_{M}\circ P^{**}{}_{T}\circ P_T\circ w'$ ou $P''_{M}\circ P^{**}{}_{T}\circ P_T\circ k\circ w'$. Isso se

deve ao fato de os prefixos $\pi'_\Omega \circ P^{**}{}_T \circ P_T$, $P'_M \circ P^{**}{}_T \circ P_T$ e $P''_M \circ P^{**}{}_T \circ P_T$ terem um tamanho mínimo[148]. Mesmo quando colocamos tudo na linguagem autodelimitada.

Pela definição da linguagem L, saber se w é da forma $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$ ou $P'_M \circ P^{**}{}_T \circ P_T \circ w'$ ou $P''_M \circ P^{**}{}_T \circ P_T \circ k \circ w'$ é um problema decidível. Logo, se $|w| \leq k_0$, então $P^{**}{}_T$ sempre decidirá que w está no caso (iv). E, como P_T é total, então existe y tal que $U(P^{**}{}_T \circ P_T \circ w) = y$, o que fecha nosso caso a).

b) Caso válido para todo w tal que $|w| \leq k$:

Queremos demonstrar que será válido também para qualquer w com $|w| = k + 1$.

Em primeiro lugar, tome um programa w arbitrário de tamanho $k + 1$.

Temos que w é da forma $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$, $P'_M \circ P^{**}{}_T \circ P_T \circ w'$, $P''_M \circ P^{**}{}_T \circ P_T \circ k' \circ w'$ ou de qualquer outra forma.

Igualmente ao caso a), saber se $w = \pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$, $w = P'_M \circ P^{**}{}_T \circ P_T \circ w'$, $w = P''_M \circ P^{**}{}_T \circ P_T \circ k' \circ w'$, ou nenhum dessas, é um problema decidível. Logo, $P^{**}{}_T$ pode sempre decidir corretamente sobre qualquer uma dessas opções. Vamos analisar abaixo cada possibilidade em separado.

(i) Caso $w = \pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$:

Basta $P^{**}{}_T$ calcular $U\left(P_\Sigma \circ P^{**}{}_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\right)$ e verificar se $U\left(P_\Sigma \circ P^{**}{}_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\right) \geq \rho$.

O cerne da questão é que $U(P^{**}{}_T \circ P_T \circ w)$ precisa estar definido para todo w, onde $|w| \leq \left|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho\right| - 1$, para que $P_\Sigma \circ P^{**}{}_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)$ sempre pare.

---

[148] No caso em que, porventura, U seja uma máquina universal na qual $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$ ou $P'_M \circ P^{**}{}_T \circ P_T \circ w'$ ou $P''_M \circ P^{**}{}_T \circ P_T \circ k \circ w'$ possa ter tamanho 1, podemos usar outros programas equivalentes, sem sacrifícios para que isso não aconteça.

Porém, pela hipótese indutiva, $U({P^{**}}_T \circ P_T \circ w)$ está bem definido para todo w com $|w| \leq k$. Por isso, como $\left|{\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho\right| = k+1$, então $U({P^{**}}_T \circ P_T \circ w)$ está definido para todo w, onde $|w| \leq \left|{\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho\right| - 1 = k$.

Logo, quando ${P^{**}}_T$ rodar $P_\sum \circ {P^{**}}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)$, ele irá parar, nunca entrando em loop.

Ademais, como ρ é dado por w, $U\left(P_\sum \circ {P^{**}}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\right)$ é calculável. E resolver se um número natural é maior ou igual ao outro é um problema decidível, então ${P^{**}}_T$ poderá verificar corretamente se $U\left(P_\sum \circ {P^{**}}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\right) \geq \rho$.

Caso não seja, então iremos ao caso (iv).

Caso seja, pela definição de ${\pi'}_\Omega$, temos que, para todo ρ', se $U\left(P_\sum \circ {P^{**}}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho'| - 1)\right) \geq \rho'$, então ${\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho'$ sempre irá parar.

Logo, se $U\left(P_\sum \circ {P^{**}}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\right) \geq \rho$, $U(T \circ w) = U\left(T \circ {\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho\right)$ sempre estará bem definido.

Como também, pelo fato de P_T ser total, é possível decidir sempre se $U(T \circ w) > U(P_T \circ w)$ ou não.

(ii) Caso $w = {P'}_M \circ {P^{**}}_T \circ P_T \circ w'$:

Basta verificar que $U\left(P_\sum \circ {P^{**}}_T \circ P_T \circ (|{P'}_M \circ {P^{**}}_T \circ P_T \circ w'| - 1)\right)$ está bem definido. Na verdade, estar definido para todo programa com tamanho menor ou igual a $|w'|$ já bastaria. De qualquer forma, pela definição da linguagem L , $|w'| \leq |{P'}_M \circ {P^{**}}_T \circ P_T \circ w'| - 1$. Como já foi dito, quando ${P'}_M$ testar para $n = |w'|$ obrigatoriamente o output de ${\pi'}_\Omega \circ {P'}_T \circ 0^{|U(P_\Sigma \circ {P'}_T \circ n)|}1 \circ U\left(P_\sum \circ {P'}_T \circ n\right)$ será maior que $U_{{P'}_T}(w')$. Portanto, $N < |w'|$. O que nos diz que ${P'}_M$ só precisa testar os programas de tamanho $\leq |w'| \leq |{P'}_M \circ {P'}_T \circ w'| - 1$ para retornar um valor.

Pois, então, vamos provar isso. Note que, pela hipótese indutiva, $U({P^{**}}_T \circ P_T \circ w)$ está bem definido para todo w com $|w| \leq k$. Por isso, como $|{P'}_M \circ {P^{**}}_T \circ P_T \circ w'| = k+1$, então $U({P^{**}}_T \circ P_T \circ w)$ está definido para todo w , onde $|w| \leq |{P'}_M \circ {P^{**}}_T \circ P_T \circ w'| - 1 = k$.

Logo, o programa $P_{\sum} \circ {P^{**}}_T \circ P_T \circ (|{P'}_M \circ {P^{**}}_T \circ P_T \circ w'| - 1)$ se deterá, isto é, produzirá um output. O que automaticamente implica que $P_{\sum} \circ {P^{**}}_T \circ P_T \circ s$ também se detém para todos os programas s tal que $|s| \leq |w'|$.

Então, $U({P'}_M \circ {P^{**}}_T \circ P_T \circ w')$ estará bem definido.

Consequentemente, $U(T \circ w) = U(T \circ {P'}_M \circ {P^{**}}_T \circ P_T \circ w')$ também estará bem definido.

Por último, teremos que, pelo fato de P_T ser total, será possível decidir sempre se $U(T \circ w) > U(P_T \circ w)$ ou não.

(iii) Caso $w = {P''}_M \circ {P^{**}}_T \circ P_T \circ k' \circ w'$:

Novamente, precisaremos da hipótese indutiva.

Temos que, pela definição de L, para qualquer w,

$$|w| < |{P''}_M \circ {P^{**}}_T \circ P_T \circ k' \circ w|$$

Pela hipótese indutiva, $U({P^{**}}_T \circ P_T \circ w)$ está bem definido para todo w com $|w| \leq k$. Por isso, como $|{P''}_M \circ {P^{**}}_T \circ P_T \circ k' \circ w'| = k + 1$, então $U({P^{**}}_T \circ P_T \circ w')$ está definido, pois $|w'| \leq |{P''}_M \circ {P^{**}}_T \circ P_T \circ k \circ w'| - 1 = k$.

Consequentemente, $U(P_{SM} \circ {P^{**}}_T \circ P_T \circ w')$, quando rodado por ${P''}_M$, produzirá um output y.

Temos também que: verificar se y é da forma ${\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho|}1 \circ \rho$ ou não é um problema sempre decidível, dando o número y previamente. Além disso, somar um número dado outro também é.

Em nossa linguagem de programação, para qualquer k' e w', $|{P''}_M \circ {P^{**}}_T \circ P_T \circ k' \circ w'| - 1 > 1$. Então, obrigatoriamente $U\big(P_{\sum} \circ {P'}_T \circ (1)\big)$ estará também bem definido.

Temos que, pelo teorema 2.15.13, $U(P_{MS} \circ {P'}_T \circ P_T \circ |\rho|)$ e $U(P_{MS} \circ {P'}_T \circ P_T \circ k)$ estão sempre bem definidos, pois só dependem da linguagem L e de P_T ser total. Como k e $|\rho|$ já são dados dentro de w, então ${P''}_M$ sempre poderá calcular os valores de $U(P_{MS} \circ {P'}_T \circ P_T \circ |\rho|)$ e $U(P_{MS} \circ {P'}_T \circ P_T \circ k)$.

Por isso tudo, teremos que $U({P''}_M \circ {P^{**}}_T \circ P_T \circ k' \circ w')$ estará bem definido. E, por isso, $U(T \circ w) = U(T \circ {P'}_M \circ {P^{**}}_T \circ P_T \circ w')$ também.

Ademais, pelo fato de P_T ser total, será possível decidir sempre se $U(T \circ w) > U(P_T \circ w)$ ou não.

(iv) Caso w não se encaixe nos casos (i), (ii) e (iii):

Segue diretamente do fato de P_T ser total.

Logo, para qualquer w com $|w| = k + 1$, $U({P^{**}}_T \circ P_T \circ w)$ estará bem definido.

c) Finalizando a indução:

Por a) e b), teremos, por indução, que $U({P^{**}}_T \circ P_T \circ w)$ está bem definido para qualquer w tal que $|w| > 0$. Como corolário, relembrando a definição de P_{MS}, teremos que $U_{{P^{**}}_T \circ P_T}$ é uma submáquina de Turing ou um subcomputador. Pois, $U(P_{MS} \circ {P^{**}}_T \circ P_T \circ w)$ será total em w. ■

2.13 O *PROJETO INTELIGENTE* NUMA NATUREZA COMPUTÁVEL

Agora vamos provar que o tempo de evolução no *projeto inteligente* de subprogramas com a natureza computável é análogo ao *projeto inteligente* no primeiro modelo de Chaitin[149]. O que isso quer dizer especificamente? A quantidade média de mutações/subprogramas para se atingir uma aptidão $\geq {BB^+}_{{P^{**}}_T \circ P_T}(N)$ será aproximadamente N, isto é, linear.

[149] CHAITIN, G. Life as Evolving Software. In: ZENIL, H. **A Computable Universe:** Understanding and Exploring Nature as Computation. [S.l.]: [s.n.], 2012. p. 277-302. ISBN 978-9814374293.

Basicamente, no *projeto inteligente*, a natureza toma a sequência das mutações que adicionam sempre um bit de informação, ou de complexidade, ao organismo anterior.[150] Em outras palavras, ao final de k mutações, o organismo final deve ter uma aptidão $\geq BB^{+}{}_{P^{**}{}_{T}\circ P_T}(k)$. Por exemplo, se o organismo anterior possui aptidão $BB^{+}{}_{P^{**}{}_{T}\circ P_T}(k)$, a próxima mutação deve levá-lo a um outro organismo com aptidão $BB^{+}{}_{P^{**}{}_{T}\circ P_T}(k+1)$. O que é o mesmo que dizer que: se o organismo anterior possui uma aptidão que nenhum subprograma de tamanho $\leq k$ consegue calcular, então o próximo organismo deve possuir uma aptidão tão grande que nenhum programa de tamanho $\leq k+1$ pode calcular.

As mutações M_k definidas por Chaitin cumprem esse papel.[151] Por sinal, elas também são cruciais para provar a evolução no *modelo cumulativo*. Portanto, precisamos de um análogo delas para nosso modelo. É por isso que construímos o programa $P''{}_{M} \circ P^{**}{}_{T} \circ P_T \circ k \circ P'{}_{M} \circ P^{**}{}_{T} \circ P_T$ – que está em função de k, somente. Vamos chamá-lo, então, de programa $M'{}_{k}$. Por conta disso, teremos que $\pi'{}_{\Omega} \circ P^{**}{}_{T} \circ P_T \circ 0^{|\rho'|}1 \circ \rho' = U_{P^{**}{}_{T}\circ P_T}(P''{}_{M} \circ P^{**}{}_{T} \circ P_T \circ k \circ P'{}_{M} \circ P^{**}{}_{T} \circ P_T \circ w) = U_{P^{**}{}_{T}\circ P_T}(M'{}_{k} \circ w)$.

Faremos mais uma prova por indução. Para $k = 1$ o resultado se mostrará quase trivial. Caberá, então, mostrar que se é válido para k, então é válido para $k+1$.

Isso será possível porque podemos mostrar que $U_{P^{**}{}_{T}\circ P_T}$ sempre deixa certo tipo de programa – que usa aproximações sempre melhores a $\Omega_{P^{**}{}_{T}\circ P_T}$ para calcular números cada vez maiores – rodar. Essas aproximações cada vez melhores são dadas justamente pelas mutações $M'{}_{k}$, as quais adicionam às aproximações anteriores uma quantidade suficiente para determinar o próximo bit de $\Omega_{P^{**}{}_{T}\circ P_T}$. A cada vez que uma delas é aplicada, se o próximo organismo não for resultante de uma melhor aproximação inferior a $\Omega_{P^{**}{}_{T}\circ P_T}$ – isto é, se $\rho' > \Omega_{P^{**}{}_{T}\circ P_T}$ – o output desse organismo será sempre menor[152] que o do anterior. E, por isso, ele irá ser descartado pela natureza, permanecendo o organismo anterior. Caso contrário, ele obrigatoriamente produz um novo programa com complexidade maior, que calcula o próximo número da imagem da função $BB^{+}{}_{P^{**}{}_{T}\circ P_T}$.

---

[150] Essa sequência pode ser escolhida pela própria Natureza ou por algum subsistema dela, um subcomputador. Além disso, pode-se argumentar: será essa é a melhor sequência possível? Isso é uma questão em aberto.
[151] CHAITIN, Id. Ibid..
[152] No caso, fizemos com que seja sempre zero, pois $\pi'{}_{\Omega} \circ P^{**}{}_{T} \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ nunca se deterá.

Em mais detalhes, o programa M'_k foi construído para que: $\rho' \leq \Omega_{P^{**}{}_T \circ P_T}$ se, e somente se, $U\left(P_\sum \circ P^{**}{}_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right) \geq \rho'$. Isso é verdadeiro porque a mutação M'_k gera um $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ com um tamanho grande o suficiente – adicionando zeros à direita de ρ'' – para que, quando a máquina calcular $U\left(P_\sum \circ P^{**}{}_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right)$, obrigatoriamente, suas $|\rho''|$ casas binárias de $\Omega_{P^{**}{}_T \circ P_T}$ estarão corretas[153] e, logo, poderemos decidir sempre se $\rho' = \rho'' \leq \Omega_{P^{**}{}_T \circ P_T}$. [154]

Já o programa $P^*{}_T$ foi construído para que $U_{P^*{}_T \circ P^{**}{}_T \circ P_T \circ P_T}$ deixe sempre $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ rodar quando $U\left(P_\sum \circ P^{**}{}_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right) \geq \rho'$. O que faz com que: $\rho' \leq \Omega_{P^{**}{}_T \circ P_T}$ se, e somente se, $U_{P^*{}_T \circ P^{**}{}_T \circ P_T \circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) = U_{P^{**}{}_T \circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) = U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$. Pois, note que, quando $\rho' > \Omega_{P^{**}{}_T \circ P_T}$, teremos que $U_{P^{**}{}_T \circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) = U_{P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$ e $U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$ não assumirá valor nenhum, já que $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ nunca parará.[155]

Como $P^{**}{}_T \circ P_T$ é total, relembrando as definições de π'_Ω, $P_\sum$ e $BB^+{}_{P^{**}{}_T \circ P_T}$, se tivermos um organismo w tal que $U_{P^{**}{}_T \circ P_T}(w) \geq BB^+{}_{P^{**}{}_T \circ P_T}(k)$, então a mutação M'_{k+1} produzirá um organismo novo da forma $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ onde: $\rho' \leq \Omega_{P^{**}{}_T \circ P_T}$ implica $U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) \geq BB^+{}_{P^{**}{}_T \circ P_T}(k+1)$ e $\rho' > \Omega_{P^{**}{}_T \circ P_T}$ implica $U_{P^{**}{}_T \circ P_T}(w) \geq BB^+{}_{P^{**}{}_T \circ P_T}(k+1)$.

Isso, aliado aos resultados do parágrafo anterior, demonstra que se $U_{P^{**}{}_T \circ P_T}(w) \geq BB^+{}_{P^{**}{}_T \circ P_T}(k)$, então $U_{P^{**}{}_T \circ P_T}\left(U_{P^{**}{}_T \circ P_T}(M'_{k+1} \circ w)\right) \geq BB^+{}_{P^{**}{}_T \circ P_T}(k+1)$. A não ser que $U_{P^{**}{}_T \circ P_T}(w) \geq BB^+{}_{P^{**}{}_T \circ P_T}(k+1)$, o que já nos dá trivialmente que a mutação M'_{k+1} elevou a aptidão para $BB^+{}_{P^{**}{}_T \circ P_T}(k+1)$, mesmo que o novo organismo seja descartado. De qualquer forma, isso nos prova que se é válido para k, então é válido para $k+1$.

Agora só falta mostrar que é válido para $k = 1$.

---

[153] Note que o tamanho do número real binário ρ'' pode ser bem menor que o de ρ'. Porém, o valor de um é igual ao outro. O que muda é sua representação enquanto *bit string*.
[154] Ver lema 2.15.19.
[155] Ver lema 2.15.19.

Vamos olhar o que a mutação $M'_1 = P''_M \circ P^{**}_T \circ P_T \circ 1 \circ P'_M \circ P^{**}_T \circ P_T$ faz com o organismo w. Caso $U_{P^{**}_T \circ P_T}(w) \geq BB^+_{P^{**}_T \circ P_T}(1)$, nada mais temos a fazer.[156] Caso $U_{P^{**}_T \circ P_T}(w) < BB^+_{P^{**}_T \circ P_T}(1)$, pela definição de π'_Ω, P_Σ e P'_M, teremos que $P'_M \circ P^{**}_T \circ P_T \circ w$ nos dará $\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|0|}1 \circ 0$ e, por isso, $P''_M \circ P^{**}_T \circ P_T \circ 1 \circ P'_M \circ P^{**}_T \circ P_T \circ w$ retornará $\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ onde $U_{P^{**}_T \circ P_T}(\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho') = U\left(\pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|U(P_\Sigma \circ P^{**}_T \circ P_T \circ (1))|}1 \circ U(P_\Sigma \circ P^{**}_T \circ P_T \circ (1))\right) = BB^+_{P^{**}_T \circ P_T}(1)$. O que nos prova que a mutação M'_1 leva, de qualquer jeito, qualquer organismo/subprograma em outro com aptidão maior ou igual a $BB^+_{P^{**}_T \circ P_T}(1)$ – claro, quando submetido à "seleção natural" feita pela natureza/computador.

2.14 O *MODELO CUMULATIVO* NUMA NATUREZA COMPUTÁVEL.

Como originalmente no modelo de Chaitin, as mutações M'_k desempenharão o papel central na prova da evolução no nosso *modelo cumulativo*. O argumento usado é também – e, talvez, mais – análogo ao usado por Chaitin. Como a natureza nunca deixa a aptidão dos organismos diminuir, em qualquer lugar que a mutação M'_{k+1} ocorrer depois da mutação M'_k ter ocorrido, isto nunca será menos "benéfico"[157] do que se ela tivesse ocorrido logo depois[158]. Por isso, a quantidade média de mutações necessárias (ou **tempo de mutação**) será também equivalente. Ou seja, o **tempo de mutação** para se atingir a aptidão $BB^+_{P^{**}_T \circ P_T}(N)$ será aproximadamente $n^2(\log_2 n)^{1+\epsilon}$, onde ϵ é uma constante.[159]

Como calculamos a probabilidade de ocorrer a mutação M'_k? Primeiramente, lembre que $M'_k = P''_M \circ P^{**}_T \circ P_T \circ k \circ P'_M \circ P^{**}_T \circ P_T$. Logo, a única coisa que varia nesse programa é k.

Teremos, pela definição da linguagem L, que $|M'_k| = |P''_M \circ P^{**}_T \circ P_T \circ k \circ P'_M \circ P^{**}_T \circ P_T| \leq C \times 6 + |P^{**}_T| + |P_T| + |P_k| + |P'_M| + |P^{**}_T| + |P_T|$, onde P_k é o menor

---

[156] A natureza só permite a aptidão, isto é, o output, aumentar.
[157] Isto é, nunca aumentará menos a aptidão do organismo final.
[158] Como ocorre no *projeto inteligente*.
[159] Da mesma forma, isso é um limitante superior. Nada impede que essa evolução possa ser mais rápida. Quão mais rápida ela pode ser é outro problema fora do escopo deste estudo.

programa de U que calcula k. Logo, existe C' tal que $|M'_k| \leq C' + H(k)$. Porém, pela definição da linguagem L, existe constante ϵ, $H(k) \leq C'' + log_2 k + (1+\epsilon)\, log_2(log_2 k)$.

Portanto, existe outra constante C''' tal que $|M'_k| \leq C''' + log_2 k + (1+\epsilon)\, log_2(log_2 k)$. Assim, a menos de uma constante[160] C_4, já podemos estimar uma cota inferior para a probabilidade de M'_k ocorrer.

$$2^{-|M'_k|-C_4} \geq 2^{-(C''' + log_2 k + (1+\epsilon)\, log_2(log_2 k))-C_4} = {}^{1}/_{C_5 k(\log_2 k)^{1+\epsilon}}$$

A partir daqui, todo o resto se torna completamente análogo à prova chaitiniana. Basta somar o tempo médio para ocorrer cada mutação, na ordem, individualmente. Ou seja, $C_5 1(\log_2 1)^{1+\epsilon} + C_5 2(\log_2 2)^{1+\epsilon} + \cdots + C_5 n(\log_2 n)^{1+\epsilon}$

No fim, teremos que o tempo de mutação será limitado superiormente por $\sum_{k=1}^{n} C_5 k(\log_2 k)^{1+\epsilon} = O(n^2(\log_2 n)^{1+\epsilon})$. O que finaliza nossa demonstração.

2.15 DEFINIÇÕES, LEMAS E TEOREMAS

2.15.1 **Definições**

a) Dizemos que uma linguagem de programação, definida sobre uma máquina de Turing universal U, é **recursivamente funcionalizável** se existir um programa tal que, dada quaisquer *bit strings P* e *w* como inputs, ele retorna uma *bit string*, a qual vamos denotar por $P \circ w$, com a qual $U(P \circ w) =$ *o resultado da computação do programa P quando w é dado como seu input.* Lembre que tem de existir um programa que decide se uma *bit string* é da forma $P \circ w$, quaisquer que sejam P e w.

[160] Essa constante pode simbolizar algum fator de correção para fazer a linguagem *L* ser **completa**. Isto é, fazer com que o somatório de todas as probabilidades de todos os programas seja igual à unidade.

Analogamente, vamos definir a concatenação $P \circ w_1 \circ ... \circ w_k$, como o programa P recebendo w_1, ... e w_k como inputs.

b) W é o conjunto de todas as *bit strings* finitas tal que existe uma enumeração computável dessas *bit strings* da forma $l_1, l_2, l_3, ..., l_k, ...$

Para fins práticos, podemos adotar uma linguagem em que $l_1 = 0$.

c) Seja $w \in W$.
$|w|$ denota o tamanho ou a quantidade de bits que w possui.

d) Vamos simbolizar apenas por N o correspondente programa na linguagem em questão do número natural N. Por exemplo, $P \circ N$ denota o programa $P \circ w$, onde w é o número natural N na linguagem L.[161]

e) Seja $r \in \mathbb{R}$ tal que $0 \leq r \leq 1$.

Denotamos por $r|_n$ o número real no qual todos os algarismos depois da n-ézima casa depois do primeiro ponto (isto é, depois da n-ézima casa binária) são zero.

Logo, se r for um número binário entre 0 e 1, por exemplo, $r|_n$ é composto somente dos seus $n + 1$ algarismos.

f) Se uma função f é computável pelo programa P então podemos também chamar f de função P.

g) Seja x um número real e k um número natural.

---

[161] A definição da linguagem L virá logo a seguir.

Então, denotamos por $(x)|_k$ o número real composto somente pela parte inteira de x e pelas k casas binárias ou decimais depois da vírgula de x.

2.15.2 **Definição: (linguagem L)**

Seja U uma máquina de Turing universal usual sobre uma linguagem L **binária**, **autodelimitada, recursiva** e **recursivamente funcionalizável** tal que existem constantes $\in$, C e C', para todo P e w, tal que

$$|w_i| < |P \circ w_1 \circ \ldots \circ w_k|, para\ i = 1,2, \ldots ou\ k$$

e

$$|P \circ w_1 \circ \ldots \circ w_k| \leq C \times k + |P| + |w_1| + |w_2| + \cdots + |w_k|$$

e

$$H(N) \leq C' + \log_2 N + (1 + \epsilon) \log_2 (\log_2 N)$$

Além disso, ela deve ser uma linguagem **completa**, isto é, que a probabilidade de ocorrer um programa qualquer seja 1. Em outras palavras,

$$\sum_{p \in L} 2^{-|p|} = 1$$

2.15.3 **Definição: (subcomputador ou submáquina de Turing)**

Seja P_f um programa de U que computa uma função total f tal que $f: L \longrightarrow X \subseteq W$.

Definimos o subcomputador (ou uma submáquina de Turing) $U/_f$ como sendo a máquina de Turing onde, para toda *bit string* w na linguagem de U, $U/_f\,(w) = U(P_f \circ w)$.

- Observação:

A linguagem W não precisa ser, necessariamente, autodelimitada.

2.15.4 **Lema**

Existe um 0-programa T que recebe w como input e retorna o tempo computacional de $U(w)$.

- Observação:

Caso w seja um 0-programa que nunca pare, então $T \circ w$ nunca para também.

2.15.5 **Lema**

Seja P_T um 0-programa arbitrário que calcula o tempo máximo de cômputo para um programa w qualquer – ou seja, P_T é uma função total qualquer.

Então, existe uma submáquina de Turing definida pela função de tempo de cômputo P_T.

Ou seja, existe $U/_{P_{SM} \circ P_T}$ onde um P_{SM} é um 0-programa que recebe P_T e w como inputs, roda $U(P_T \circ w)$ e retorna:

(i) l_1, se $U(w)$ não parar em tempo de cômputo $\leq U(P_T \circ w)$;
(ii) l_{k+1}, se $U(w)$ parar em tempo de cômputo $\leq U(P_T \circ w)$ e $U(w) = l_k$;

- Observação:

Esse programa define um subcomputador (uma submáquina de Turing) que retorna um símbolo conhecido (no caso, zero) quando um programa de w não parar em tempo $\leq U(P_T \circ w)$ ou retorna o mesmo output (a menos de uma bijeção trivial) de $U(w)$ quando este parar em tempo $\leq U(P_T \circ w)$.

Para ser uma submáquina de Turing, ela precisa estar definida para todas os inputs. Isso ocorre pelo fato de P_T ser total.

2.15.6 **Notação**

Vamos denotar somente por U_{P_T} a submáquina de Turing ${}^{U}/_{P_{SM} \circ P_T}$, tal que:

$$\forall w(\, U_{P_T}(w) = {}^{U}/_{P_{SM} \circ P_T}\,(w) = U(P_{SM} \circ P_T \circ w))$$

Além disso, dizemos que uma função ou um número é x-computável se ele for computável pelo computador U_x. E, portanto, um 0-programa é um programa quando rodado por U. Um x-programa é um programa quando rodado por U_x.

2.15.7 **Definição**

Seja ${P'}_T$ uma função total.

Vamos definir a função $BB^{+}{}_{{P'}_T}(N)$, a função *Busy-Beaver Plus*, pelo seguinte processo:

(i) Gera uma lista de todos os outputs de $U_{{P'}_T}(w)$ tal que $|w| \leq N$;

(ii) Toma o maior número dessa lista;

(iii) Soma 1;

(iv) Retorna esse valor.

2.15.8 **Teorema**

Seja N um número natural arbitrário.

Seja P'_T uma função total.

Seja $U_{P'_T}$ uma submáquina de Turing.

Então, a função $BB^+{}_{P'_T}(N)$ é incomputável por qualquer programa de $U_{P'_T}$.

Isto é,

$$\forall P \exists N_0 \forall N (\, N \geq N_0 \;\rightarrow\; U_{P'_T}(P \circ N) < BB^+{}_{P'_T}(N))$$

– Demonstração:

Tome um programa P arbitrário.

Pela definição da linguagem L e de U, temos que:

$$|P \circ N| \leq C + |P| + C' + \log_2 N + (1+\epsilon)\log_2(\log_2 N)$$

Seja $C + |P| + C' = C''$.

Sabemos que $\exists N_0 \forall N (\, N \geq N_0 \;\rightarrow\; C'' + \log_2 N + (1+\epsilon)\log_2(\log_2 N) < N)$.

Logo, $\exists N_0 \forall N (N \geq N_0 \;\rightarrow\; |P \circ N| < N)$.

Consequentemente, pela definição de $BB^+{}_{P'_T}$, teremos que:

$$\exists N_0 \forall N (\, N \geq N_0 \;\rightarrow\; U_{P'_T}(P \circ N) < U_{P'_T}(P \circ N) + 1 \leq BB^+{}_{P'_T}(N))$$

■

2.15.9 **Lema**

Seja P'_T uma função total.

Existe um 0-programa $P_\sum$ que recebe P'_T e N como inputs e roda $U_{P'_T}(w)$, para todo $w \in L$ com $|w| \leq N$. Com isso, $P_\sum$ pega aqueles cujos outputs são diferentes de zero l_1 e soma. Por fim, retorna o valor dessa soma.

2.15.10 **Definição**

Seja P'_T uma função total.

Denotamos por $\Omega_{P'_T}$ a *time limited halting probability* dada por:

$$\Omega_{P'_T} = \sum_{p \text{ é um programa de } L \text{ que se detém em tempo de cômputo} \leq U(P'_T \circ p).} 2^{-|p|}$$

- Observação:

Quanto mais crescente for a função dada pelo programa P'_T, mais próximo de Ω chegamos.

Na verdade, não precisamos ~~realmente~~ de fato da condição de que P'_T seja total. Se $U(P'_T \circ p)$ não estiver definido, então $U(p)$ pode até não parar, pois ele estará incluso em $\Omega_{P'_T}$. Por exemplo, se P'_T for um programa que não se detém para qualquer input, então

$$\Omega_{P'_T} = \sum_{p \in L.} 2^{-|p|} = 1$$

De qualquer forma, a condição de que P'_T seja total nos garante que $\Omega_{P'_T}$ só inclua programas que param.

2.15.11 **Lema**

Seja P'_T uma função total.

Seja ρ um número binário real positivo fracionário.

Existe um 0-programa π'_Ω que recebe P'_T e $0^{|\rho|}1 \circ \rho$ como inputs. Então, ele começa a testar $U(P_\Sigma \circ P'_T \circ N)$ para $N = 1,2,3, \ldots, k, \ldots$ e para quando k for o primeiro número tal que $U\left(P_\Sigma \circ P'_T \circ k\right) \geq \rho$. Em seguida, π'_Ω pega o maior output do conjunto $\left\{U_{P'_T}(w) \mid |w| \leq k\right\}$ e soma 1. Por fim, retorna esse último valor.

2.15.12 **Lema**

Seja P'_T uma função total.

Existe um programa P'_M que recebe $P'_T \circ w$ como input, lê a si mesmo e, então, roda $U\left(P_\Sigma \circ P'_T \circ n\right)$ para $n = 1,2, \ldots$ testando em cada caso se $U\left(\pi'_\Omega \circ P'_T \circ 0^{|U(P_\Sigma \circ P'_T \circ n)|}1 \circ U\left(P_\Sigma \circ P'_T \circ n\right)\right) > U_{P'_T}(w)$. Quando achar o primeiro n que isso seja verdadeiro, toma o valor $N = n - 1$ e constrói $U\left(P_\Sigma \circ P'_T \circ N\right)$. Então, retorna $\pi'_\Omega \circ P'_T \circ 0^{|U(P_\Sigma \circ P'_T \circ N)|}1 \circ U\left(P_\Sigma \circ P'_T \circ N\right)$.

2.15.13 **Teorema**

Seja P_T uma função total.

Seja k um número natural qualquer.

Seja $H_{P^*{}_T \circ P'{}_T \circ P_T}$ o conjunto de todos os programas p de L que param em tempo de cômputo $\leq U(P^*{}_T \circ P'{}_T \circ P_T \circ p)$, caso $P'{}_T$ seja total.[162]

Existe um 0-programa P_{MS} que recebe $P'{}_T$, P_T , k como input e que sempre (independetemente de $P'{}_T$ ser total ou não) se detém retornando um valor m_S tal que, se $P'{}_T$ for total e $P^*{}_T \circ P'{}_T \circ P_T$ computar exatamente a mesma função que $P'{}_T$, para todo $m \geq m_S$ teremos que

$$\left(\sum_{|p|<m \, ; \, p \in H_{P^*{}_T \circ P'{}_T \circ P_T}} 2^{-|p|} \right)\Bigg|_k = (\Omega_{P^*{}_T \circ P'{}_T \circ P_T})\big|_k$$

– Demonstração:

Como L é uma linguagem recursiva, então existe um programa P_{MS} que recebe $P'{}_T$, P_T, k como input e, para $i = 2,3,...$, começa iterando, do menor para o maior, o somatório

$$\sum_{|p|<i \, ; \, p \in H_i} 2^{-|p|}$$

Onde H_i é o conjunto de todos os programas de L com tamanho $< i$ que param em tempo P_T ou tem a forma $P'{}_M \circ P'{}_T \circ w'$, a forma $P''{}_M \circ P'{}_T \circ k \circ w'$ ou tem a forma $\pi'{}_\Omega \circ P'{}_T \circ 0^{|\rho|}1 \circ \rho$, onde

$$\rho \leq \sum_{|p|<\left|\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho\right|=j<i \, ; \, p \in H_i} 2^{-|p|}$$

e $j \geq 1$.

Ele continua essa iteração até que ache o primeiro i' tal que

---

[162] Se P_T e $P'{}_T$ forem totais, obrigatoriamente $P^*{}_T \circ P'{}_T \circ P_T$ também será.

$$\left(\sum_{|p| \geq i' \,;\, p \in L} 2^{-|p|} + \sum_{|p| < i' \,;\, p \in H_{i'}} 2^{-|p|} \right) \Bigg|_k = \left(\sum_{|p| < i' \,;\, p \in H_{i'}} 2^{-|p|} \right) \Bigg|_k$$

Então, retorna esse i'.

Agora, primeiramente, temos que nos perguntar se esse programa sempre se detém, se ele está bem definido.

Saber se uma data *bit string* é da forma $P^*{}_T \circ P \circ P_T$ ou não é um problema decidível, pela definição da linguagem L.

Note que será somente a partir de um tamanho mínimo que aparecerá um programa da forma $P'_M \circ P'_T \circ w'$, $P''_M \circ P'_T \circ k \circ w'$ ou $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$. Logo, como P_{MS} começa iterando dos programas de tamanho menor para os maiores, até ele chegar a esse tamanho mínimo, apenas contará para o somatório os programas que param em tempo P_T.

Além disso, pode-se ver que se o somatório está definido para i, então ele estará definido para $i + 1$. Por quê? Um programa de tamanho $i + 1 - 1 = i$ pode parar em tempo P_T, ser da forma $P'_M \circ P'_T \circ w'$ ou $P''_M \circ P'_T \circ k \circ w'$. Até aqui saber se ele é somado ou não no somatório é um procedimento trivialmente recursivo, dado que P_T é total. Já se ele for da forma $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$, basta verificar se

$$\rho \leq \sum_{|p| < |\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho| = i \,;\, p \in H_i} 2^{-|p|}$$

Mas, pela hipótese indutiva, teremos que $\sum_{|p| < |\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho| = i \,;\, p \in H_i} 2^{-|p|}$,já foi calculado. Logo, P_{MS} pode verificar se ρ é menor ou igual a esse somatório.

Então, por uma indução simples, teremos que P_{MS} pode calcular $\sum_{|p| < i \,;\, p \in H_i} 2^{-|p|}$ para qualquer i.

Porém, assim, nosso programa em questão pode continuar indefinidamente a iteração sobre i e nunca parar. Resta saber, por isso, se alguma hora

$$\left(\sum_{|p|\geq i' \,;\, p\in L} 2^{-|p|} + \sum_{|p|<i' \,;\, p\in H_{i'}} 2^{-|p|} \right)\Bigg|_k = \left(\sum_{|p|<i' \,;\, p\in H_{i'}} 2^{-|p|} \right)\Bigg|_k$$

Note, pela definição da linguagem L, que

$$0 < \lim_{i\to\infty} \sum_{|p|<i \,;\, p\in H_i} 2^{-|p|} < 1$$

Como esse somatório é crescente, pela definição de limite, teremos que, para todo $\epsilon_1 > 0$, existe i_0 suficientemente grande tal que, para todo $i \geq i_0$, $\left(\lim_{i\to\infty} \sum_{|p|<i \,;\, p\in H_i} 2^{-|p|}\right) - \sum_{|p|<i \,;\, p\in H_i} 2^{-|p|} < \epsilon_1$.

Outra vez, pela definição da linguagem L, temos que

$$\lim_{i\to\infty} \sum_{|p|\geq i \,;\, p\in L} 2^{-|p|} = 0$$

Por isso, para todo $\epsilon_2 > 0$, existe i_1 suficientemente grande tal que, para todo $i \geq i_1$, $\sum_{|p|\geq i \,;\, p\in L} 2^{-|p|} < \epsilon_2$.

Como estamos lidando com um número binário estritamente entre 0 e 1, é importante notar que $Lim = \lim_{i\to\infty} \sum_{|p|<i \,;\, p\in H_i} 2^{-|p|}$ pode ser um número com um número de casas finito depois da vírgula ou, então, com o um número infinito enumerável. Como Lim é um número binário estritamente entre zero e 1, mesmo se tiver uma quantidade finita (por exemplo, n) de algarismos depois da vírgula, o somatório $\sum_{|p|<i \,;\, p\in H_i} 2^{-|p|}$ irá apresentar, a partir de um certo i, uma sequência infinita de 1´s. Em particular, ele será uma dízima da forma $(Lim - 2^{-n})|_n * 111$ ... A iteração converge para um número desse formato e não para um número com um número finito de casas depois da vírgula. No limite, esse número de infinitos algarismos (de infinitos 1´s) depois da vírgula será o mesmo Lim de antes com finitos algarismos depois da vírgula.

Em qualquer um dos casos podemos tomar um valor suficientemente pequeno ϵ_2 tal que $(Lim + \epsilon_2)|_k = Lim|_k$ ou $((Lim - 2^{-n})|_n * 111 \ldots + \epsilon_2)|_k = Lim|_k$.
Temos que

$$\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|} + \sum_{|p|\geq i\,;\,p\in L} 2^{-|p|} = Lim + \sum_{|p|\geq i\,;\,p\in L/H_i} 2^{-|p|}$$

Onde L/H_i é o conjunto de todos os programas de L que não pertencem a nenhum H_j tal que $j \geq i$.
Temos também que $\sum_{|p|\geq i\,;\,p\in L/H_i} 2^{-|p|} \leq \sum_{|p|\geq i\,;\,p\in L} 2^{-|p|}$.
Logo, para todo $i \geq i_1$, $\sum_{|p|\geq i\,;\,p\in L/H_i} 2^{-|p|} < \epsilon_2$.
Portanto, pela definição de limite, no caso de Lim ter tamanho infinito, teremos

$$\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|} \leq \sum_{|p|<i\,;\,p\in H_i} 2^{-|p|} + \sum_{|p|\geq i\,;\,p\in L} 2^{-|p|} < Lim + \epsilon_2$$

Como estamos lidando com números binários estritamente entre 0 e 1, juntamente com nossa escolha de ϵ_2,

$$\left(\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}\right)\Bigg|_k \leq \left(\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|} + \sum_{|p|\geq i\,;\,p\in L} 2^{-|p|}\right)\Bigg|_k \leq (Lim + \epsilon_2)|_k = Lim|_k$$

Por outro lado, tanto no caso Lim - o caso $(Lim - 2^{-n})|_n * 111 \ldots = Lim$ será deixado a cargo do leitor -, podemos tomar ϵ_1 suficientemente pequeno tal que $Lim - \epsilon_1 \geq Lim|_k$. Ou ainda, que $(Lim - \epsilon_1)|_k \geq Lim|_k$.
Temos, pela definição de limite, que, para todo $i \geq i_0$,

$$\left(\lim_{i\to\infty} \sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}\right) - \epsilon_1 < \sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}$$

Logo,

$$\left(\left(\lim_{i\to\infty}\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}\right)-\epsilon_1\right)\Bigg|_k \leq \left(\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}\right)\Bigg|_k$$

Mas, pela nossa escolha de ϵ_1,

$$Lim|_k \leq \left(\left(\lim_{i\to\infty}\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}\right)-\epsilon_1\right)\Bigg|_k \leq \left(\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}\right)\Bigg|_k$$

$$\leq \left(\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}+\sum_{|p|\geq i\,;\,p\in L} 2^{-|p|}\right)\Bigg|_k$$

Tome, então, $i_* = \max\{i_0, i_1\}$.
Logo, para todo $i \geq i_*$,[163]

$$Lim|_k \leq \left(\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}\right)\Bigg|_k \leq \left(\sum_{|p|<i\,;\,p\in H_i} 2^{-|p|}+\sum_{|p|\geq i\,;\,p\in L} 2^{-|p|}\right)\Bigg|_k \leq Lim|_k$$

Na verdade, no caso de Lim ser um número com finitos algarismos depois da vírgula, o qual deixamos a cargo do leitor, teremos a desigualdade:

163

$$((Lim - 2^{-n})|_n * 111\ldots)|_k \leq \left(\sum_{|p|<i\,;\, p\in H_i} 2^{-|p|} - 2^{-n} + 0\ldots0 * 1\ldots1 \right)\Bigg|_k$$

$$\leq \left(\sum_{|p|<i\,;\, p\in H_i} 2^{-|p|} + \sum_{|p|\geq i\,;\, p\in L} 2^{-|p|} - 2^{-n} + 0\ldots0 * 1\ldots1 \right)\Bigg|_k$$

$$\leq ((Lim - 2^{-n})|_n * 111\ldots)|_k$$

Portanto, para todo $i' \geq i_*$,

$$\left(\sum_{|p|\geq i'\,;\, p\in L} 2^{-|p|} + \sum_{|p|<i'\,;\, p\in H_{i'}} 2^{-|p|} \right)\Bigg|_k = \left(\sum_{|p|<i'\,;\, p\in H_{i'}} 2^{-|p|} \right)\Bigg|_k = Lim|_k$$

Ou

$$\left(\sum_{|p|\geq i'\,;\, p\in L} 2^{-|p|} + \sum_{|p|<i'\,;\, p\in H_{i'}} 2^{-|p|} - 2^{-n} + 0\ldots0 * 1\ldots1 \right)\Bigg|_k$$

$$= \left(\sum_{|p|<i'\,;\, p\in H_{i'}} 2^{-|p|} - 2^{-n} + 0\ldots0 * 1\ldots1 \right)\Bigg|_k = ((Lim - 2^{-n})|_n * 111\ldots)|_k$$

O que, de qualquer forma, nos diz que programa P_{MS} sempre para.
Além disso, para todo $i \geq i' \geq i_*$, $\left(\sum_{|p|<i'\,;\, p\in H_{i'}} 2^{-|p|}\right)\Big|_k = Lim|_k$.

A última parte do teorema é mostrar que $Lim = \Omega_{P^*{}_T \circ P'{}_T \circ P_T}$, caso $P'{}_T$ seja total e $P'{}_T$ computar a mesma função que $P^*{}_T \circ P'{}_T \circ P_T$.
O que decorre trivialmente da definição do programa $P^*{}_T$, pois o programa $P^*{}_T$ recebe $P'{}_T$, P_T e w como inputs e retorna:

(i) $U(T \circ w)$, se w for da forma $\pi'_\Omega \circ P'{}_T \circ 0^{|\rho|}1 \circ \rho$ e, além disso, $U\big(P_\Sigma \circ P'{}_T \circ (|\pi'_\Omega \circ P'{}_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big) \geq \rho$ e $U(T \circ w) > U(P_T \circ w)$;

(ii) $U(T \circ w)$, se w for da forma ${P'}_M \circ {P'}_T \circ w'$ e $U(T \circ w) > U(P_T \circ w)$;

(iii) $U(T \circ w)$, se w for da forma ${P''}_M \circ {P'}_T \circ k \circ w'$ e $U(T \circ w) > U(P_T \circ w)$;

(iv) $U(P_T \circ w)$, caso não se encaixe nos casos (i), (ii) e (iii);

Mas se ${P^*}_T \circ {P'}_T \circ P_T$ for igual a ${P'}_T$, então $U\left(P_\sum \circ {P'}_T \circ (|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho|}1 \circ \rho| - 1)\right) = U\left(P_\sum \circ {P^*}_T \circ {P'}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho|}1 \circ \rho| - 1)\right)$.

Logo, todos, e somente, os programas de L que pararem em tempo P_T, tiverem a forma ${P'}_M \circ {P'}_T \circ w'$, a forma ${P''}_M \circ {P'}_T \circ k \circ w'$ ou tiverem a forma ${\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho|}1 \circ \rho$, onde

$$\rho \leq \sum_{|p|<|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho|}1 \circ \rho|=j \,;\, p \in H_j \subseteq H_{{P^*}_T \circ {P'}_T \circ P_T}} 2^{-|p|}$$

irão ser contados em Lim. E isso independe do fato de ${P'}_T$ ser total ou não. É verdade que $\Omega_{{P^*}_T \circ {P'}_T \circ P_T}$ só estará bem definido quando ${P'}_T$ for total, porém, o valor Lim independe disso. Ele não depende do tempo que os programas ${P'}_M \circ {P'}_T \circ w'$, ${P''}_M \circ {P'}_T \circ k \circ w'$ ou ${\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho|}1 \circ \rho$ levam para rodar. O que, apesar de inusitado, é fundamental.

■

2.15.14 **Lema**

Seja ${P'}_T$ uma função total.

Seja k um número natural qualquer.

Existe um programa P''_M que recebe P'_T, k e w como inputs, onde k é um número natural[164], e roda $U_{P'_T}(w)$, obtendo y tal que $y = U_{P'_T}(w)$. Em seguida, cai numa das duas opções:

(i) Caso y seja da forma $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$, onde ρ é um número real positivo binário entre 0 e 1, P''_M soma $1/2^k$ ao número ρ gerando outro número ρ''. Especialmente se $\rho = 0$, então apenas toma $\rho'' = U\big(P_\Sigma \circ P'_T \circ (1)\big)$. Depois, adiciona uma quantidade suficiente (ou nenhum) de zeros à direita de ρ'' até que $\big|\pi'_\Omega \circ P'_T \circ 0^{|\rho''\circ 0\ldots 0|}1 \circ \rho'' \circ 0 \ldots 0\big| - 1 \geq U(P_{MS} \circ P'_T \circ P_T \circ |\rho|)$ e $\big|\pi'_\Omega \circ P'_T \circ 0^{|\rho''\circ 0\ldots 0|}1 \circ \rho'' \circ 0 \ldots 0\big| - 1 \geq U(P_{MS} \circ P'_T \circ P_T \circ k)$. Por fim, retorna $\pi'_\Omega \circ P'_T \circ 0^{|\rho''\circ 0\ldots 0|}1 \circ \rho'' \circ 0 \ldots 0$.

(ii) Caso y não seja dessa forma, retorna $U_{P'_T}(w)$.

2.15.15 **Lema**

Sejam P'_T e P_T duas funções totais.
Existe um programa P^*_T que recebe P'_T, P_T e w como inputs e retorna:

(i) $U(T \circ w)$, se w for da forma $\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho$ e, além disso, $U\big(P_\Sigma \circ P'_T \circ (|\pi'_\Omega \circ P'_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big) \geq \rho$ e $U(T \circ w) > U(P_T \circ w)$;
(ii) $U(T \circ w)$, se w for da forma $P'_M \circ P'_T \circ w'$ e $U(T \circ w) > U(P_T \circ w)$;
(iii) $U(T \circ w)$, se w for da forma $P''_M \circ P'_T \circ k \circ w'$ e $U(T \circ w) > U(P_T \circ w)$;
(iv) $U(P_T \circ w)$, caso não se encaixe nos casos (i), (ii) e (iii);

2.15.16 **Lema**

---

[164] Lembre que não há problema em se admitir números naturais como inputs, vide a notação 2.15.1.d).

Sejam P'_T e P_T funções totais tais que $P^*_T \circ P'_T \circ P_T$ computa exatamente a mesma função que P'_T.

Seja $w \in L$.

Seja $U(P''_M \circ P'_T \circ k \circ P'_M \circ P'_T \circ w) = \pi'_\Omega \circ P'_T \circ 0^{|\rho'|}1 \circ \rho'$.

Então,

$$\rho' \leq \Omega_{P^*_T \circ P'_T \circ P_T} \leftrightarrow$$
$$\leftrightarrow \rho' \leq U\left(P_\Sigma \circ P^*_T \circ P'_T \circ P_T \circ (|\pi'_\Omega \circ P'_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right)$$

- Demonstração:

Seja $s = \max\{|\rho|, k\}$.

(i) Caso $\rho' \leq \Omega_{P^*_T \circ P'_T \circ P_T} \rightarrow \rho' \leq U\left(P_\Sigma \circ P^*_T \circ P'_T \circ P_T \circ (|\pi'_\Omega \circ P'_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right)$:

Suponha que $\rho' \leq \Omega_{P^*_T \circ P'_T \circ P_T}$.

Teremos que $\rho' = \rho + {}^1/_{2^k} = \rho|_s + {}^1/_{2^k} = \rho'$, onde $\rho \leq U\left(P_\Sigma \circ P'_T \circ (|P'_M \circ P'_T \circ w| - 1)\right)$.

Temos que $U\left(P_\Sigma \circ P^*_T \circ P'_T \circ P_T \circ (|\pi'_\Omega \circ P'_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right) < \Omega_{P^*_T \circ P'_T \circ P_T}$.

E, pela definição de P''_M e pelo teorema 2.15.13, teremos que

$$\left(\sum_{|p| < |\pi'_\Omega \circ P'_T \circ 0^{|\rho'|}1 \circ \rho'| \,;\, p \in H_{P^*_T \circ P'_T \circ P_T}} 2^{-|p|} \right)\Bigg|_s = \left(\Omega_{P^*_T \circ P'_T \circ P_T}\right)\Big|_s$$

Logo, pela definição de P_Σ e pelo teorema 2.15.13, $(U\left(P_\Sigma \circ P^*_T \circ P'_T \circ P_T \circ (|\pi'_\Omega \circ P'_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right))\big|_s = \left(\Omega_{P^*_T \circ P'_T \circ P_T}\right)\big|_s$.

Como estamos lidando com números binários entre 0 e 1, pela nossa suposição teremos obrigatoriamente que $\rho' = \rho|_s + {}^1/_{2^k} \leq \left(\Omega_{P^*_T \circ P'_T \circ P_T}\right)\big|_s$. Note que $|\rho|$ é sempre finito.

Consequentemente, $\rho' = \rho|_s + {}^1\!/_{2^k} \leq (U(P_\Sigma \circ {P^*}_T \circ {P'}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)))\big|_s \leq U(P_\Sigma \circ {P^*}_T \circ {P'}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho'|}1 \circ \rho'| - 1))$.

(ii) Caso $\rho' \leq \Omega_{{P^*}_T \circ {P'}_T \circ P_T} \leftarrow \rho' \leq U(P_\Sigma \circ {P^*}_T \circ {P'}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho'|}1 \circ \rho'| - 1))$:

Suponha que $\rho' > \Omega_{{P^*}_T \circ {P'}_T \circ P_T}$:

Teremos que $\rho' = \rho + {}^1\!/_{2^k} = \rho|_s + {}^1\!/_{2^k} = \rho'|_s$, onde $\rho \leq U(P_\Sigma \circ {P'}_T \circ (|{P'}_M \circ {P'}_T \circ w| - 1))$.

O que nos traz obrigatoriamente pela nossa suposição que $\rho' = \rho + {}^1\!/_{2^k} = \rho|_s + {}^1\!/_{2^k} = \rho'|_s > (\Omega_{{P^*}_T \circ {P'}_T \circ P_T})\big|_s$.

Temos, pela definição de ${P''}_M$ e pelo teorema 2.15.13, que $(U(P_\Sigma \circ {P^*}_T \circ {P'}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)))\big|_s = (\Omega_{{P^*}_T \circ {P'}_T \circ P_T})\big|_s$.

Então, $\rho' = \rho + {}^1\!/_{2^k} = \rho|_s + {}^1\!/_{2^k} = (\rho + {}^1\!/_{2^k})\big|_s > (U(P_\Sigma \circ {P^*}_T \circ {P'}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)))\big|_s$.

Logo, como são números entre 0 e 1, $\rho' = \rho + {}^1\!/_{2^k} = \rho|_s + {}^1\!/_{2^k} = (\rho + {}^1\!/_{2^k})\big|_s > U(P_\Sigma \circ {P^*}_T \circ {P'}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho'|}1 \circ \rho'| - 1))$.

Então, por redução ao absurdo, demonstramos que $\rho' \leq \Omega_{{P^*}_T \circ {P'}_T \circ P_T} \leftarrow \rho' \leq U(P_\Sigma \circ {P^*}_T \circ {P'}_T \circ P_T \circ (|{\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho'|}1 \circ \rho'| - 1))$.

■

2.15.17 **Lema**

Seja P_T uma função total.

Existe um programa ${P^{**}}_T$ que recebe P_T e w como inputs, lê a si mesmo, a P_T e a w, monta ${P^{*}}_T \circ {P^{**}}_T \circ P_T \circ P_T \circ w$ e retorna $U({P^{*}}_T \circ {P^{**}}_T \circ P_T \circ P_T \circ w)$. Isto é, retorna:

(i) $U(T \circ w)$, se w for da forma $\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$ e, além disso, $U\big(P_\Sigma \circ {P^{**}}_T \circ P_T \circ (|\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big) \geq \rho$ e se $U(T \circ w) > U(P_T \circ w)$;

(ii) $U(T \circ w)$, se w for da forma ${P'}_M \circ {P^{**}}_T \circ P_T \circ w'$ e se $U(T \circ w) > U(P_T \circ w)$;

(iii) $U(T \circ w)$, se w for da forma ${P''}_M \circ {P^{**}}_T \circ P_T \circ k \circ w'$ e se $U(T \circ w) > U(P_T \circ w)$;

(iv) $U(P_T \circ w)$, caso não se encaixe nos casos (i), (ii) e (iii);

2.15.18 **Teorema**

Seja P_T uma função total.

Então, para qualquer $w \in L$, $U({P^{**}}_T \circ P_T \circ w)$ é um valor bem definido.

Ou seja, $U({P^{**}}_T \circ P_T \circ w)$ é uma função total em w.

– Demonstração:

a) Caso válido para todo w tal que $|w| \leq k_0$:

Como π'_Ω, ${P'}_M$ e ${P''}_M$ são programas conhecidos, eles possuem um tamanho (dentro da linguagem L de nossa escolha).

Também conhecemos o programa ${P^{**}}_T \circ P_T$, o qual também terá, por isso, um tamanho.

Logo,

$$\exists k_0 (\, k_0 = \min\{|\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ w|, |{P'}_M \circ {P^{**}}_T \circ P_T \circ w|, |{P''}_M \circ {P^{**}}_T \circ P_T \circ w|, para\ qualquer\ w \in L\} - 1)$$

Por isso,

$$\forall w(|w| \leq k_0 \rightarrow w \neq \pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho \wedge$$
$$w \neq {P'}_M \circ {P^{**}}_T \circ P_T \circ w' \wedge$$
$$w \neq {P''}_M \circ {P^{**}}_T \circ P_T \circ k \circ w')$$

Pela definição da linguagem L, saber se $w = \pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho \vee w = {P'}_M \circ {P^{**}}_T \circ P_T \circ w' \vee w = {P''}_M \circ {P^{**}}_T \circ P_T \circ k \circ w' \vee (w \neq \pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho \wedge w \neq {P'}_M \circ {P^{**}}_T \circ P_T \circ w' \wedge w \neq {P''}_M \circ {P^{**}}_T \circ P_T \circ k \circ w')$ se torna um problema decidível. Logo, ${P^{**}}_T$ pode sempre decidir corretamente sobre qualquer uma dessas opções.
Então,

$$\forall w(|w| \leq k_0 \rightarrow U({P^{**}}_T \circ P_T \circ w) = U(P_T \circ w))$$

Como, por suposição inicial, P_T é uma função total, então

$$\forall w \exists y(|w| \leq k_0 \rightarrow U({P^{**}}_T \circ P_T \circ w) = y)$$

Isso finaliza a primeira parte da demonstração.

b) Caso válido para todo w tal que $|w| \leq k$:

Queremos demonstrar que será válido também para qualquer w com $|w| = k + 1$.
Primeiramente, tome um programa w arbitrário de tamanho $k + 1$.
Temos que w é da forma $\pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$, ${P'}_M \circ {P^{**}}_T \circ P_T \circ w'$, ${P''}_M \circ {P^{**}}_T \circ P_T \circ w' \circ k$ ou de qualquer outra forma.
Igualmente ao caso a, saber se $w = \pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho \vee w = {P'}_M \circ {P^{**}}_T \circ P_T \circ w' \vee w = {P''}_M \circ {P^{**}}_T \circ P_T \circ w' \circ k \vee (w \neq \pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho \wedge w \neq {P'}_M \circ {P^{**}}_T \circ P_T \circ w' \wedge w \neq {P''}_M \circ {P^{**}}_T \circ P_T \circ w' \circ k)$ é um problema decidível. Logo, ${P^{**}}_T$ pode sempre decidir corretamente sobre qualquer uma dessas opções. Vamos analisar abaixo cada possibilidade em separado.

(i) Caso $w = \pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$:

Basta $P^{**}{}_T$ calcular $U\big(P_{\sum} \circ P^{**}{}_T \circ P_T \circ (|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big)$ e verificar se $U\big(P_{\sum} \circ P^{**}{}_T \circ P_T \circ (|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big) \geq \rho$.

O cerne da questão é que $U(P^{**}{}_T \circ P_T \circ w)$ precisa estar definido para todo w, onde $|w| \leq \big|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho\big| - 1$, para que $P_{\sum} \circ P^{**}{}_T \circ P_T \circ (|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)$ sempre pare.

Porém, pela hipótese indutiva, $U(P^{**}{}_T \circ P_T \circ w)$ está bem definido para todo w com $|w| \leq k$. Por isso, como $\big|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho\big| = k + 1$, então $U(P^{**}{}_T \circ P_T \circ w)$ está definido para todo w, onde $|w| \leq \big|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho\big| - 1 = k$.

Logo, quando $P^{**}{}_T$ rodar $P_{\sum} \circ P^{**}{}_T \circ P_T \circ (|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)$, ele irá parar, nunca entrando em loop.

Ademais, como ρ é dado por w, $U\big(P_{\sum} \circ P^{**}{}_T \circ P_T \circ (|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big)$ é calculável e decidir se um número é maior ou igual ao outro é um problema decidível, então $P^{**}{}_T$ poderá verificar corretamente se $U\big(P_{\sum} \circ P^{**}{}_T \circ P_T \circ (|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big) \geq \rho$.

Caso não seja, então iremos ao caso (iv).

Caso seja, pela definição de π'_{Ω}, teremos que, para todo ρ', se $U\big(P_{\sum} \circ P^{**}{}_T \circ P_T \circ (|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho'| - 1)\big) \geq \rho'$, então $\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho'$ sempre irá parar.

Logo, se $U\big(P_{\sum} \circ P^{**}{}_T \circ P_T \circ (|\pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho| - 1)\big) \geq \rho$, $U(T \circ w) = U\big(T \circ \pi'_{\Omega} \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho\big)$ sempre estará bem definido.

Com também, pelo fato de P_T ser total, é possível decidir sempre se $U(T \circ w) > U(P_T \circ w)$ ou não.

(ii) Caso $w = P'_M \circ P^{**}{}_T \circ P_T \circ w'$:

Basta verificar se $U\big(P_{\sum} \circ P^{**}{}_T \circ P_T \circ (|P'_M \circ P^{**}{}_T \circ P_T \circ w'| - 1)\big)$ está bem definido.

Para isso, note que, pela hipótese indutiva, $U({P^{**}}_T \circ P_T \circ w)$ está bem definido para todo w com $|w| \leq k$. Assim, como $|{P'}_M \circ {P^{**}}_T \circ P_T \circ w'| = k + 1$, então $U({P^{**}}_T \circ P_T \circ w)$ está definido para todo w, onde $|w| \leq |{P'}_M \circ {P^{**}}_T \circ P_T \circ w'| - 1 = k$.

Logo, o programa $P_\sum \circ {P^{**}}_T \circ P_T \circ (|{P'}_M \circ {P^{**}}_T \circ P_T \circ w'| - 1)$ se deterá, isto é, produzirá um output.

Então,

$$U({P'}_M \circ {P^{**}}_T \circ P_T \circ w') =$$
$$= {\pi'}_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{\left|U\left(P_\sum \circ P^{**}{}_T \circ P_T \circ (|{P'}_M \circ P^{**}{}_T \circ P_T \circ w'|-1)\right)\right|}1 \circ$$
$$\circ U\left(P_\sum \circ {P^{**}}_T \circ P_T \circ (|{P'}_M \circ {P^{**}}_T \circ P_T \circ w'| - 1)\right)$$

Consequentemente, $U(T \circ w) = U(T \circ {P'}_M \circ {P^{**}}_T \circ P_T \circ w')$ estará bem definido.

Por último, teremos que, pelo fato de P_T ser total, será possível decidir sempre se $U(T \circ w) > U(P_T \circ w)$ ou não.

(iii) Caso $w = {P''}_M \circ {P^{**}}_T \circ P_T \circ k \circ w'$:

Outra vez, precisaremos da hipótese indutiva apenas numa das operações de ${P''}_M$. As outras são trivialmente bem definidas em qualquer situação. Ou seja, precisamos verificar se $U_{{P^{**}}_T \circ P_T}(w')$ produz output ou se não se detém.

Temos que, pela definição de L, para qualquer w, $|w| < |{P''}_M \circ {P^{**}}_T \circ P_T \circ k \circ w|$.

Pela hipótese indutiva, $U({P^{**}}_T \circ P_T \circ w)$ está bem definido para todo w com $|w| \leq k$. Por isso, como $|{P''}_M \circ {P^{**}}_T \circ P_T \circ k \circ w'| = k + 1$, então $U({P^{**}}_T \circ P_T \circ w')$ está definido, pois $|w'| \leq |{P''}_M \circ {P^{**}}_T \circ P_T \circ k \circ w'| - 1 = k$.

Consequentemente, $U(P_{SM} \circ {P^{**}}_T \circ P_T \circ w')$, quando rodado por ${P''}_M$, produzirá um output y.

Temos também que verificar se y é da forma ${\pi'}_\Omega \circ {P'}_T \circ 0^{|\rho|}1 \circ \rho$ ou não é um problema sempre decidível, dando o número y previamente. Além disso, somar um número dado outro, também é.

Já provamos que $U(P_{MS} \circ {P^{**}}_T \circ P_T \circ |\rho|)$ e $U(P_{MS} \circ {P^{**}}_T \circ P_T \circ k)$ estão sempre bem definidos, pois só dependem da linguagem L e do fato de P_T ser total. Como k e $|\rho|$, já

são dados dentro de w, então P''_M sempre poderá calcular os valores de $U(P_{MS} \circ |\rho|)$ e $U(P_{MS} \circ k)$.

Logo, $U(P''_M \circ P^{**}_T \circ P_T \circ k \circ w')$ estará bem definido. E, por isso, $U(T \circ w) = U(T \circ P'_M \circ P^{**}_T \circ P_T \circ w')$ também.

Ademais, pelo fato de P_T ser total, será possível decidir sempre se $U(T \circ w) > U(P_T \circ w)$ ou não.

(iv) Caso w não se encaixe nos casos (i), (ii) e (iii):

Segue diretamente do fato de P_T ser total.

Logo, para qualquer w com $|w| = k + 1$, $U(P^{**}_T \circ P_T \circ w)$ estará bem definido.

c) Finalizando a indução:

Por a) e b), teremos, por indução, que $U(P^{**}_T \circ P_T \circ w)$ está bem definido para qualquer w tal que $|w| > 0$.

■

2.15.19 **Teorema**

Seja P_T uma função total.

Seja $w \in L$.

Seja $U_{P^{**}_T \circ P_T}(P''_M \circ P^{**}_T \circ P_T \circ k \circ P'_M \circ P^{**}_T \circ P_T \circ w) = \pi'_\Omega \circ P^{**}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$.

Então,

$$\rho' \leq \Omega_{P^{**}_T \circ P_T} \leftrightarrow$$

$$\leftrightarrow\ U_{P^{**}{}_T\circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) = U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$$

- Demonstração:

Como, pelo teorema 2.15.18, $P^{**}{}_T \circ P_T$ é total, então $U_{P^{**}{}_T\circ P_T}(P''_M \circ P^{**}{}_T \circ P_T \circ k \circ P'_M \circ P^{**}{}_T \circ P_T \circ w) = U_{P^{*}{}_T\circ P^{**}{}_T\circ P_T\circ P_T}(P''_M \circ P^{**}{}_T \circ P_T \circ k \circ P'_M \circ P^{**}{}_T \circ P_T \circ w) = U(P''_M \circ P^{**}{}_T \circ P_T \circ k \circ P'_M \circ P^{**}{}_T \circ P_T \circ w)$.
Então, pelo lema 2.15.16 temos que

$$\rho' \leq \Omega_{P^{*}{}_T\circ P^{**}{}_T\circ P_T\circ P_T}\ \leftrightarrow$$
$$\leftrightarrow\ \rho' \leq U\left(P_\sum \circ P^{*}{}_T \circ P^{**}{}_T \circ P_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right)$$

Mas, pela construção de $P^{**}{}_T$, temos que,

$$\rho' \leq U\left(P_\sum \circ P^{*}{}_T \circ P^{**}{}_T \circ P_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right)$$
$$= U\left(P_\sum \circ P^{**}{}_T \circ P_T \circ (|\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right) \rightarrow$$
$$\rightarrow\ U_{P^{**}{}_T\circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) = U_{P^{*}{}_T\circ P^{**}{}_T\circ P_T\circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$$
$$= U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$$

Logo,

$$\rho' \leq \Omega_{P^{**}{}_T\circ P_T} = \Omega_{P^{*}{}_T\circ P^{**}{}_T\circ P_T\circ P_T} \rightarrow\ U_{P^{**}{}_T\circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$$
$$= U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$$

Para provar o sentido inverso, note que

$$\rho' > \Omega_{P^{*}{}_T\circ P^{**}{}_T\circ P_T\circ P_T} = \Omega_{P^{**}{}_T\circ P_T} \rightarrow\ U_{P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$$
$$\neq U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$$

Isso se dá porque, quando π'_Ω começar a calcular o somatório de todas as probabilidades de todos os programas que param em tempo $\leq U(P^{**}{}_T \circ P_T \circ w)$ para averiguar se este se iguala

ou ultrapassa ρ', esse processo nunca cessará. Pois, o somatório de todas as probabilidades de todos os programas que param em tempo $\leq U({P^{**}}_T \circ P_T \circ w)$ é igual a $\Omega_{{P^{**}}_T \circ P_T}$ e este é, por suposição, menor que ρ'.

Na verdade, como $\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'$ será um programa que nunca para, então $U\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$ não assumirá valor nenhum, não está bem definido. Isso já nos daria que $U_{{P^*}_T \circ {P^{**}}_T \circ P_T \circ P_T}\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) \neq U\left(\pi'_\Omega \circ {P'}_T \circ 0^{|\rho'|}1 \circ \rho'\right)$, pois, ${P^*}_T \circ {P^{**}}_T \circ P_T \circ P_T$ é total quando ${P^{**}}_T \circ P_T$ e P_T também o são.

Porém, se quisermos, ainda assim usar o Lema 2.15.16, teremos que

$$\begin{aligned}
\rho' > \Omega_{{P^*}_T \circ {P^{**}}_T \circ P_T \circ P_T} = \Omega_{{P^{**}}_T \circ P_T} \rightarrow\ & U_{P_T}\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) \\
& \neq U\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) \wedge\ \rho' \\
& > U\left(P_\Sigma\ \ \circ {P^{**}}_T \circ P_T \circ (|\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right) \\
& = U\left(P_\Sigma\ \ \circ {P^*}_T \circ {P^{**}}_T \circ P_T \circ P_T \circ (|\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right)
\end{aligned}$$

Contudo, pela definição de ${P^{**}}_T$, temos que

$$\begin{aligned}
U_{P_T}\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) & \neq U\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) \wedge\ \rho' \\
& > U\left(P_\Sigma\ \ \circ {P^{**}}_T \circ P_T \circ (|\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'| - 1)\right) \\
& \rightarrow U_{{P^{**}}_T \circ P_T}\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) \\
& = U_{{P^*}_T \circ {P^{**}}_T \circ P_T \circ P_T}\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) \\
& = U_{P_T}\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)
\end{aligned}$$

Logo,

$$\begin{aligned}
\rho' > \Omega_{{P^{**}}_T \circ P_T} \rightarrow U_{{P^{**}}_T \circ P_T}\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) & = U_{P_T}\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) \\
& \neq U\left(\pi'_\Omega \circ {P^{**}}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)
\end{aligned}$$

Então, por redução ao absurdo,

$$\rho' \leq \Omega_{{P^{**}}_T \circ P_T}$$

$$\leftarrow U_{P^{**}{}_T \circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right) = U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho'|}1 \circ \rho'\right)$$

O que finaliza nossa demonstração. ■

2.15.20 **Teorema**

Seja N um número natural.

Seja P_T uma função total.

Então, para todo N, existe um programa da forma $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho$, tal que $U_{P^{**}{}_T \circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|}1 \circ \rho\right) \geq BB^+{}_{P^{**}{}_T \circ P_T}(N)$.

- A ideia da prova:

Vamos usar o fato de $\Omega_{P^{**}{}_T \circ P_T}$ ser um número que contém informação necessária sobre os programas de $U_{P^{**}{}_T \circ P_T}$ para que se compute a função $BB^+{}_{P^{**}{}_T \circ P_T}(N)$. O que, propositalmente, é análogo a Ω e $BB(N)$. Além disso, $P^{**}{}_T$ foi construído para permitir que aproximações inferiores a $\Omega_{P^{**}{}_T \circ P_T}$ sejam utilizadas para $U_{P^{**}{}_T \circ P_T}$ poder calcular algum valor da imagem da função $BB^+{}_{P^{**}{}_T \circ P_T}(N)$.

- Demonstração:

Pelo teorema anterior, provamos que, para qualquer input w, $P^{**}{}_T \circ P_T \circ w$ se detém. Isto é, $\forall w \exists y(\ \mathrm{U}(P^{**}{}_T \circ P_T \circ w) = \mathrm{y}\ )$.
Segue, de imediato, que a função $U(P_\Sigma \circ P^{**}{}_T \circ P_T \circ N)$ também estará bem definida para qualquer N.

Tomemos, então, N como sendo um número natural arbitrário.

Teremos que $U(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N)$ nos dá um número real binário entre 0 e 1, o qual é uma aproximação inferior a $\Omega_{{P^{**}}_T \circ P_T}$.

Temos, também, pela definição do programa π'_{Ω}, que

$$U\left(\pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|U(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N)|} 1 \circ U\left(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N\right)\right) = {BB^{+}}_{{P^{**}}_T \circ P_T}(N)$$

A cláusula (i) do programa ${P^{**}}_T$ nos garante que se

$$U\left(P_{\sum} \circ {P^{**}}_T \circ P_T \circ (|\pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|} 1 \circ \rho| - 1)\right) \geq \rho$$

então

$${U_{{P^{**}}_T \circ P_T}}\left(\pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|} 1 \circ \rho\right) = U(\pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|\rho|} 1 \circ \rho)$$

Vamos, então, aumentar o tamanho de $U(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N)$, caso seja necessário, pelo procedimento seguinte:

> Tome o número real entre 0 e 1 dado por $U(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N)$ e adicione (caso seja necessário) zeros à direita o suficiente até que

$$\left|\pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|U(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N) \circ 0 \ldots 0|} 1 \circ U\left(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N\right) \circ 0 \ldots 0\right| - 1 \geq N$$

Temos que, para quaisquer k, k', se $k' \geq k$, então $U\left(P_{\sum} \circ {P^{**}}_T \circ P_T \circ k'\right) \geq U\left(P_{\sum} \circ {P^{**}}_T \circ P_T \circ k\right)$.

Logo,

$$U\left(P_{\sum} \circ {P^{**}}_T \circ P_T \circ (|\pi'_{\Omega} \circ {P^{**}}_T \circ P_T \circ 0^{|U(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N) \circ 0 \ldots 0|} 1 \circ U\left(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N\right) \circ 0 \ldots 0| - 1)\right) \geq U\left(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N\right) = U\left(P_{\sum} \circ {P^{**}}_T \circ P_T \circ N\right) \circ 0 \ldots 0$$

Então, pela cláusula (i),

$$U_{P^{**}{}_T \circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|U(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N)\circ 0 \ldots 0|} 1 \circ U\left(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N\right) \circ 0 \ldots 0\right) = U(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|U(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N)\circ 0 \ldots 0|} 1 \circ U\left(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N\right) \circ 0 \ldots 0)$$

Mas, como $0^{|U(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N)\circ 0 \ldots 0|} 1 \circ U\left(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N\right) \circ 0 \ldots 0$ e $U\left(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N\right)$ são dois números reais iguais, então

$$U_{P^{**}{}_T \circ P_T}\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|U(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N)\circ 0 \ldots 0|} 1 \circ U\left(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N\right) \circ 0 \ldots 0\right) = U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|U(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N)\circ 0 \ldots 0|} 1 \circ U\left(P_\Sigma \ \circ P^{**}{}_T \circ P_T \circ N\right) \circ 0 \ldots 0\right) = U\left(\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|U(P_\Sigma \circ P^{**}{}_T \circ P_T \circ N)|} 1 \circ U\left(P_\Sigma \circ P^{**}{}_T \circ P_T \circ N\right)\right) = BB^+{}_{P^{**}{}_T \circ P_T}(N)$$

Dessa forma, conseguimos o nosso desejado programa da forma $\pi'_\Omega \circ P^{**}{}_T \circ P_T \circ 0^{|\rho|} 1 \circ \rho$ que calcula $BB^+{}_{P^{**}{}_T \circ P_T}(N)$ quando rodado pela submáquina $U_{P^{**}{}_T \circ P_T}$.

■

3 A EVOLUÇÃO DE HIPERPROGRAMAS

3.1 INTRODUÇÃO

Podemos entender, de forma geral, a metabiologia como sendo uma teoria matemática para a evolução de programas, de software. Ela se inspirou nas teorias da evolução biológica, assim como, pelos conceitos de complexidade e de probabilidade algorítmica, ambos da teoria da informação algorítmica. Organismos evoluindo seriam programas que aumentam de complexidade. O que, *grosso modo*, quer dizer que se pode ir computando problemas que necessitariam cada vez mais de programas maiores para computar.

Dessa forma, já nos ambientamos com o panorama geral que iremos tratar neste capítulo e não nos aprofundaremos no que seja a metabiologia, nos seus fundamentos e nem sobre seus resultados. Queremos focar num desdobramento dela, ou melhor, num vislumbre do limite ante o qual consideramos comumente como verdadeiro: a matemática.

Antes de chegar – ou de se aproximar – nesse ponto, vale a pena dizer rapidamente o que se entende por um programa: é uma *bit string* (uma cadeia finita de 0´s e 1´s) numa fita de uma máquina de Turing universal. Essa é a forma mais geral de definirmos matematicamente um programa – ou, se desejar, um algoritmo. Em essência, o que é computável o é por um programa, ou seja, existe um programa que pode realizar a operação desejada (por exemplo, decidir se um número natural pertence ou não a um conjunto).

Sabemos, como mostrado por Chaitin, que é possível a evolução de programas, levando-os a se aproximarem cada vez mais da complexidade de Ω, a qual é infinita.[165] Nesse modelo já apresentado na metabiologia, os seres vivos e as mutações (estas, aleatoriamente geradas) são programas. A natureza é capaz de saber todos os dígitos de Ω, isto é, ela seria o equivalente a uma máquina oracular de Turing, e é assim justamente porque é capaz de decidir sempre se um programa arbitrário se detém ou não.[166]

Isso posto, podemos dizer que conseguimos um resultado positivo para uma evolução por mutações algorítmicas aleatórias de sistemas computáveis. Porém, sempre devemos nos

[165] O número ômega é um número real entre 0 e 1 cujos dígitos depois da vírgula são incomputáveis.
[166] Note que computar o número ômega e resolver o problema da parada se equivalem.

perguntar se esse modelo de fato pode nos ajudar na compreensão da evolução da vida e quais seriam os seus pontos fracos. Afinal, como o próprio nome induz, a metabiologia, como teoria, tem pelo menos "um olho voltado" para a biologia. Tal assunto irá ser discutido um pouco melhor na seção 3.7.

Avançando sobre esse pensamento, podemos nos perguntar: e aquilo que não é computável? Se os seres vivos, em alguma instância ou estágio evolutivo, não forem sistemas computáveis, a metabiologia cairia por terra ou ficaria sem propósito empírico como entendimento da vida e sua evolução? Se sim, não faria sentido um modelo matemático de evolução dos seres vivos baseado na evolução de programas.

Contudo, podemos estudar matematicamente a evolução de sistemas capazes de "computar" o não computável? O objetivo principal do presente estudo é mostrar que a metabiologia ainda dá resultados úteis e profícuos, mesmo nesse caso – talvez – menos palpável e mais abstrato: quando "engolimos" o não computável para dentro de uma formalização. Na verdade, essa possibilidade tem sua retumbância na matemática. Não é nada inédita a formalização de máquinas teóricas (por exemplo, as máquinas oraculares de Turing) capazes de executar tais tarefas, mas, no que concerne à metabiologia, precisamos saber se é possível esses sistemas incomputáveis irem ganhando complexidade suficiente para serem capazes de resolver mais e mais problemas indecidíveis.

De início, iremos definir o que chamamos de hiperprogramas, que nada mais são que programas de máquinas de Turing oraculares. Então, discutiremos se é possível os organismos/hiperprogramas irem evoluindo[167] de forma a conseguirem computar problemas aritméticos cada vez mais não computáveis.

Vale lembrar que existe uma hierarquia de problemas indecidíveis. Por exemplo, da mesma forma que existe o problema da parada para os computadores comuns, existe um segundo problema da parada relativo às maquinas oraculares, capazes de computar o primeiro problema da parada. De modo análogo, podemos montar um terceiro problema da parada relativo às máquinas oraculares que são capazes de computar o segundo problema da parada. E assim por diante, formando uma hierarquia de problemas cada vez mais indecidíveis. A subida dessa escada do inalcançável caracteriza o adjetivo "cantoriano" – que será mais bem explicado adiante – no espírito de todo nosso texto. Note, então, que computar Ω ainda não é

[167] Isto é, ganhando complexidade e/ou ordem.

suficiente para nos dar todas as verdades matemáticas. Na verdade, podemos montar uma hierarquia de problemas indecidíveis e uma hierarquia de números ômega. Ambas, estritamente ligadas, uma a uma.[168]

Pode-se dizer, inclusive, que a demonstração desse capítulo também se sustenta sob o fenômeno da incomputabilidade relativa, assim como a da evolução de subprogramas. Na verdade, a incomputabilidade está intimamente ligada à forma como medimos o ganho de complexidade dos seres metavivos, através da função de aptidão. A diferença é que a versão chaitiniana lida com a incomputabilidade clássica. A evolução de subprogramas lida com a incomputabilidade relativa recursiva e a evolução de hiperprogramas lida com a incomputabilidade relativa oracular, a qual já é bem conhecida na literatura.

Em outras palavras, neste capítulo estudaremos uma evolução de sistemas não computáveis em que os seres "vivos" – metavivos – vão sendo capazes de resolver mais e mais problemas indecidíveis **sobre a aritmética**. Queremos, com isso, saber se tal modelo evolutivo pode "dar conta" de qualquer verdade aritmética. **Estamos procurando uma evolução metabiológica que leve, no limite, à completude da aritmética**.

Da mesma forma que, no modelo proposto originalmente por Chaitin, a natureza é capaz de computar o objetivo final dos organismos (no caso, saber todos os bits de Ω), nosso modelo precisa que a natureza seja capaz de computar todas as verdades aritméticas, nosso objetivo final.

Nosso desejo é formalizar uma prova que pareça uma extensão natural do modelo que já se conseguiu na metabiologia. O que também fizemos nos capítulos 1 e 2. Em particular, a prova desse capítulo será uma versão da **busca exaustiva** para hiperprogramas. Não faremos provas para o **projeto inteligente** nem para a **evolução cumulativa**.

Se conseguirmos que esses hiperprogramas evoluam, toda a hierarquia de problemas indecidíveis será contemplada por esse processo interminável de evolução dos hiperprogramas. O que possibilita aos hiperprogramas/organismos responderem a qualquer pergunta sobre a aritmética, conforme a evolução vai ocorrendo. A evolução dos hiperprogramas seria completa (no sentido lógico matemático).

Como já dissemos, o que almejamos é chegar a algum resultado análogo aos já obtidos na metabiologia, o que pode ser traduzido pelas perguntas: para todo n, existem sequências de

[168] CHAITIN, G. J. **Exploring Randomness**. Londres: Springer-Verlag London Limited, 2001. ISBN 1-85233-417-7.

mutações hiperalgorítmicas que levem qualquer organismo inicial a um organismo final que seja capaz de decidir se qualquer hiperprograma de ordem $\leq n$ se detém ou não? Qual a quantidade média de tentativas que devemos esperar para que ocorra uma dessas sequências de mutações?

Uma demonstração positiva para responder a essas perguntas acarretaria o fato de que a evolução dos seres vivos metabiológicos (podemos chamá-los de seres metavivos) eleva a complexidade dos seres para que eles sejam capazes, eventualmente, de saber se qualquer sentença aritmética é falsa ou verdadeira. Mesmo que a cada estágio de evolução não se possa ter a resposta para todas elas, teríamos apenas que esperar um tempo razoável a fim de que a evolução nos levasse a poder decidir uma nova sentença, antes indecidível. Um processo interminável e progressivo que nos daria todas as verdades aritméticas conforme as mutações vão sendo aplicadas ao longo do tempo.

Isso não nos lembra do sentimento – e que está bem longe de ser uma constatação – de que sempre podemos resolver os problemas matemáticos dando tempo suficiente? Sentimento tão defendido por muitos como sendo uma das coisas que nos diferencia das máquinas. Numa natureza desse tipo que estamos lidando aqui, talvez, os seres humanos não sejam capazes de serem completos, mas a evolução iria nos levar assimptoticamente a isso - nós, seres vivos como um todo. Sugerimos que o leitor veja o contraponto disso na proposta da seção 1.4.2.

Caso contrário[169], mesmo com uma natureza dessa magnitude, os seres metavivos nunca chegariam à completude da aritmética – apesar de serem capazes de ultrapassar a barreira do computável. Eles nunca poderiam nem se aproximar assimptoticamente da complexidade da natureza. O que é analogamente diferente do que ocorre, por exemplo, com Ω, pois existe um programa que converge para ele se for deixado rodar de modo indefinido. E, por isso, diferente do que ocorre com os modelos já estabelecidos por Chaitin. Um resultado desse tipo colocaria a completude ainda mais distante da computação.

Enfim, dada essas duas opções e mais uma terceira, que seria a de que não podemos demonstrar "nem que podemos, nem que não podemos", ficaríamos esperando uma resposta vinda da matemática da metabiologia. Porém, parece que essas duas últimas opções não se aplicam. A partir do exposto neste capítulo, de fato, pode existir a evolução de

[169] Caso se demonstre que não há tais sequências de mutações hiperalgorítmicas para todo n.

hiperprogramas em direção à completude aritmética. É para esse intuito que as demonstrações abaixo se devotam.

Basicamente, nossos organismos e mutações serão sistemas não computáveis que vamos chamar de hiperprogramas. A natureza será um hiperprograma – de ordem superior a qualquer organismo e mutação – capaz de resolver qualquer problema aritmético.

Assim, dada uma sentença aritmética de ordem qualquer, queremos estimar qual a quantidade média de mutações para que o organismo final seja capaz de decidir se essa sentença é verdadeira ou falsa. Para isso precisaremos definir uma hierarquia dessas sentenças e um objetivo final para os organismos análogo à função $BB'(N)$: em nosso caso será a função $BB_{\omega}(N)$. Uma função *à la Busy-Beaver* que nos dá uma maximização de cômputo em relação a todos os hiperprogramas de ordem finita. Diferentemente da *Busy-Beaver* usual, que dá uma maximização de cômputo em relação a todos os programas (hiperprogramas de ordem zero). Usaremos, então, essa nova função como função de aptidão, a mesma que usamos para medir a complexidade ou criatividade de nossos organismos/hiperprogramas.[170]

De início, definiremos os hiperprogramas e a probabilidade sobre o espaço de todos eles. A natureza será o hiperprograma de ordem maior convenientemente escolhido para que seja capaz de saber se qualquer hiperprograma de ordem finita se detém ou não. O que será a mesma coisa que resolver qualquer problema aritmético, ou seja, qualquer problema incomputável por hiperprogramas de ordem finita.

A demonstração consiste em que, para todo valor de $BB_{\omega}(N)$, existe uma mutação – em particular, algorítmica – com certa probabilidade que leva qualquer organismo num outro capaz de retornar o valor de $BB_{\omega}(N)$ e cuja ordem é suficiente para resolver os problemas aritméticos contemplados em $BB_{\omega}(N)$. Esta é a **nossa versão do modelo de busca exaustiva**. É, basicamente, um problema de nomeação recursiva de máquinas oraculares de Turing a partir da hierarquia aritmética, porém, numa "roupagem" da teoria da informação algorítmica e usando conceitos da metabiologia.

Nesta parte da tese, as demonstrações matemáticas serão mais informais. Espera-se que não haja dificuldade em entendê-las e verificá-las tomando como base a literatura sugerida. Por esse motivo, nos abstivemos de provar alguns lemas. Ora porque esmiuçá-los nos levaria a um formalismo que tiraria o leitor do foco do texto, ora porque eles são bastante

[170] Mais sobre o assunto na seção 3.5.

triviais, e ora porque consideramos esses resultados como "esperados"[171]. De qualquer forma, o leitor é sempre convidado a verificar formalmente cada passo.

3.2 CONSIDERAÇÕES INICIAIS

3.2.1 **Ultrapassando o computável**

Vamos ampliar a noção de programa, estendendo a noção do que seja uma máquina de Turing universal U – diremos que esta é uma máquina de Turing oracular de ordem 0 ($U = U_0$).

Então, uma máquina de Turing oracular de ordem 1 (U_1) será uma máquina de Turing oracular de ordem 0, a qual tem acesso a um oráculo que pode dar as respostas para o problema da parada relativo a U_0. E, assim por diante.

Por exemplo, podemos usar a hierarquia dos números Ω de Chaitin, criador também da metabiologia, para servir como oráculos. Assim, os hiperprogramas nada mais são do que programas com sub-rotinas capazes de consultar qualquer bit de números reais (na forma binária) aos quais ele tem acesso (no nosso caso, os números ômega). Note que, como hiperprogramas são *bit strings* finitas, a quantidade máxima de números reais a que eles têm acesso tem que ser enumerável. Assim, um hiperprograma de ordem 1 tem acesso – ou pode chamar – a qualquer bit do número Ω. Um hiperprograma de ordem 2 tem acesso a qualquer bit dos números Ω e Ω_2. Um hiperprograma de ordem 3 tem acesso a qualquer bit dos números Ω, Ω_2 e Ω_3. Esse processo continua para toda ordem finita. Dessa forma, $U_1 = U_\Omega$, $U_2 = U_{\Omega,\Omega_2}$, $U_3 = U_{\Omega,\Omega_2,\Omega_3}$, ...

Portanto, um hiperprograma será um programa de uma U_n, para qualquer $n \geq 0$. Possivelmente, a visualização de um hiperprograma se torne mais fácil usando os números Ω como oráculos, justamente, porque consultar bits de números reais (no caso, números Ω) pode

[171] Com certeza, menos inusitados do que os resultados dos capítulos 1 e 2.

ser mais intuitivo do que consultar funções. A formalização que usaremos aqui para as máquinas oraculares de Turing será bem próxima da original, criada por ele.[172]

Máquinas universais de Turing com diversas fitas são bem conhecidas na teoria da computação, sabemos muito bem como defini-las.[173] Portanto, uma máquina oracular de ordem 1 é apenas uma máquina universal de Turing com duas fitas. No entanto, só uma delas recebe dígitos do "programador"[174]. A chamaremos de fita principal e nela reside o programa da máquina oracular de Turing. A outra fita tem que já estar preenchida – nesse caso, ela será preenchida pelos dígitos do número Ω. Quando a fita extra estiver preenchida por um número oracular, como Ω, passaremos a chamar o programa na fita principal de **hiperprograma** e a máquina de Turing composta por essas duas fitas de **máquina oracular de Turing de primeira ordem** ou **hipercomputador de primeira ordem**.

Conforme formos adicionando nelas fitas extras e colocando, respectivamente, Ω_2, Ω_3, ..., construiremos hipercomputadores de segunda ordem, terceira ordem, ou de qualquer outra ordem finita.[175] Note que o que varia de uma máquina oracular para outra de mesma ordem é apenas o conteúdo na fita principal, o qual é sempre um programa finito ou autodelimitado. Este usa as sub-rotinas contidas na tabela de estado definida para a máquina universal de Turing de múltiplas fitas para consultar o que está escrito nas outras fitas. E o que está escrito nelas é sempre um número real que serve de oráculo para algum grau de Turing correspondente.

Em essência, os graus de Turing denotam que nível de oráculos precisamos para computar (ou decidir) um conjunto de números naturais. O grau 0 é aquele de todos os conjuntos naturais que podem ser computados (ou que são recursivos). O grau 1 ou 0′ é aquele de todos os conjuntos de números naturais que podem ser computados se for dado um oráculo para o problema da parada. Por sua vez, o grau 2 ou 0′′ é aquele de todos os conjuntos de números naturais que podem ser computados se for dado um oráculo para o

---

[172] TURING, A. Systems of logic based on ordinals. **Proceedings of the London Mathematical Society**, 1939. 161-228.
[173] LEWIS, H. R.; PAPADIMITRIOU, C. H. **Elementos de Teoria da Computação**. 2ª. ed. Porto Alegre: Bookman, 2000.
[174] Na metabiologia, o papel de programador pode ser feito pela natureza juntamente com as mutações aleatórias. Ou seja, não importa quem ou o que coloca os bits na fita.
[175] Na verdade, a quantidade máxima de fitas que podemos adicionar é enumerável e não apenas finita. Nossa natureza será uma máquina de Turing com acesso a infinitas fitas, por exemplo. Explicaremos isso melhor adiante.

problema da parada relativo a U_1. Essa ordenação de graus continua e corresponde, intencionalmente, à nossa hierarquia de hipercomputadores U_1, U_2, U_3, ...

Se um ser vivo for um sistema capaz de computar o problema da parada, por exemplo, podemos representá-lo por um hiperprograma de uma U_Ω. Se um ser vivo for um sistema capaz de computar o problema da parada para U_Ω, podemos representá-lo por um hiperprograma de uma U_2. Esse processo interminável nos leva a construir uma hierarquia de hiperprogramas, cada vez com ordens maiores que as outras, capazes de computar cada vez mais problemas incomputáveis para os anteriores. Note que Ω_{n+1} é tão aletório para U_n quanto Ω é para U. Logo, uma mutação subir a ordem de um hiperprograma/organismo em 1 é equivalente[176] a se injetar uma quantidade infinita de informação de uma só vez.

Se conhecermos razoavelmente bem essa teoria, não será difícil ver que os modelos para a evolução metabiológica, dados por Chaitin[177], podem ser provados também para uma natureza sendo um hiperprograma de ordem $n+1$ e os organismos e mutações sendo hiperprogramas de ordem n. Basta notar que Ω_{n+1} é tão aletório para U_n quanto Ω é para U.

Isso já nos permitiria a evolução de sistemas complexos o bastante para superar qualquer computador, como se conhece hoje em dia. Note que, ainda no caso acima, teríamos um limite (um número natural n) para a ordem que os hiperprogramas poderiam alcançar. Daí, talvez, sejamos impelidos a ir além; talvez sejamos tomados por um espírito cantoriano provocador. Podemos mesmo ultrapassar este limite?

3.2.2 **Em direção à completude**

O matemático Georg Cantor foi o criador dos números transfinitos. E o fez visando "subir as escadas" até "Deus", até o "tudo". Basta notar a ordenação de infinitos cada vez maiores que ele nos deu com alguns dos mais famosos de seus teoremas. Tal processo interminável culmina em algo que é tão grande que não pode ser nem um conjunto, como ele

[176] Porém, não é exatamente a mesma coisa. Lembre que hiperprogramas são finitos em qualquer ordem e que eles apenas consultam os números oraculares. Explicaremos melhor adiante.
[177] CHAITIN, G. Life as Evolving Software. In: ZENIL, H. **A Computable Universe:** Understanding and Exploring Nature as Computation. [S.l.]: [s.n.], 2012. p. 277-302. ISBN 978-9814374293.

mesmo demonstrou.[178] Sem nos alongar sobre esse assunto, queremos "pegar carona" nessa ideia e perguntar se é possível os organismos da metabiologia serem capazes de atingir uma complexidade necessária para "resolver tudo". Obviamente, precisamos explicar melhor o que queremos dizer por 'resolver tudo'.

Assim, vamos importar, também da lógica matemática, o conceito de completude. Apesar de o desejo de se conseguir um sistema completo para toda a matemática dos números ser, em geral, atribuído à escola hilbertiana, os processos e as ideias aqui expostos estão longe desse projeto de Hilbert.[179] A noção de sistema axiomático finito é totalmente atropelada pelos hiperprogramas, e a completude advinda da "subida" pelos problemas indecidíveis já é, em si, algo contraditório com a completude dos axiomas de Peano[180]. Poderíamos dizer, talvez com mais pertinência, que o espírito deste trabalho está mais próximo de um "programa cantoriano de completude". Podemos dizer que para o primeiro programa de pesquisa se quer o todo a partir do finito e para este último se quer o todo a partir do infinito.

Enfim, sabe-se que não há programa que possa nos dar todas as respostas para todas as perguntas sobre aritmética, sobre a teoria dos conjuntos ou sobre qualquer sistema axiomático finito que seja poderoso o suficiente para lidar com grande parte da matemática[181]. Mas e nossos hiperprogramas? Eles poderiam nos dar essas respostas?

Um sistema completo, em nosso caso, seria aquele capaz de responder sempre a qualquer pergunta sobre a aritmética ou sobre qualquer computação. Mas, acontece também que, para conseguir esse poder, nenhuma máquina de Turing oracular de ordem finita é suficiente. Nenhuma sequência de números ômega $\Omega,\ \Omega_2, \Omega_3, \ldots, \Omega_n$, com n sendo um número finito, é capaz de dar todas as respostas. Por conseguinte, precisamos estabelecer qual a ordem mínima do hiperprograma que faria o "papel" da natureza, e que deve, por suposição, nos dar a completude da aritmética.

Resta, então, perguntar: saber os primeiros enumeráveis números ômega bastaria para termos todas as respostas que queremos? E mais: seria possível que os organismos podem ir evoluindo e ganhando complexidade suficiente para responder a qualquer pergunta que possa ser feita sobre a natureza? Lembre que a natureza também é um hiperprograma e que estamos

---

[178] É possível formalizar esse infinito como sendo uma classe própria de elementos, por exemplo.
[179] De modo resumido, tal projeto consistia em provar a consistência dos axiomas de Peano.
[180] Os axiomas de Peano são a referência comum para uma axiomática da aritmética.
[181] Esses resultados vêm dos teoremas de incompletude de Gödel.

partindo do princípio de que qualquer pergunta precisa ser formulada nessa mesma linguagem, envolvendo apenas computações (ou números naturais).

Uma coisa imprescindível ao analisar os modelos de evolução de Chaitin é lembrar que a natureza sempre tem de ter complexidade irredutível a qualquer outro ser vivo ou mutação. Seja de forma relativa, como no capítulo 1, ou de forma absoluta. Isto é, a natureza tem quer ser uma máquina de Turing oracular de ordem superior a qualquer organismo ou mutação. Por conseguinte, precisamos estabelecer qual a ordem mínima do hiperprograma que seria a natureza, na nossa versão do modelo por busca exaustiva, que nos daria a completude da aritmética.

Usando o resultado que liga a hierarquia aritmética aos graus de Turing (que são, *grosso modo*, a ordem de nossas máquinas oraculares)[182], para toda pergunta na linguagem da aritmética existe um n finito tal que existe um conhecido hiperprograma de U_{n} que pode decidir se ela é verdadeira ou falsa no modelo standard da aritmética. Portanto, uma natureza que pudesse consultar qualquer oráculo de ordem finita seria capaz de decidir qualquer sentença aritmética e seria um sistema completo. Com esse intuito, usando índices ordinais, podemos considerar a natureza como sendo um hiperprograma de $U_{\Omega,\,\Omega_2,\Omega_3,\ldots} = U_{\omega}$, onde ω é o primeiro ordinal limite, o primeiro ordinal maior que qualquer número natural.[183] A natureza será um programa de uma máquina universal de Turing com acesso a um número infinito enumerável de fitas extras, cada uma delas preenchida com os seus respectivos números ômega, configurando, portanto, uma máquina oracular de Turing de ordem ω.

Temos que dizer que máquinas com um número infinito de fitas não é algo usual na literatura. Porém, não é difícil verificar que o mesmo procedimento recursivamente enumerável nas sub-rotinas da tabela de estado para indexar cada fita, usado no caso com finitas fitas, pode ser aplicado para indexar uma quantidade enumerável de fitas.

A esse hiperprograma que fará o "papel"[184] metabiológico de natureza chamaremos de **hipernatureza**. Tal natureza seria, assim, uma máquina oracular capaz de computar qualquer número ômega de ordem finita e, portanto, capaz de decidir se qualquer mutação ou

---

[182] ROGERS, H. **Theory of recursive functions and effective computability**. 3 ed. [S.l.]: MIT Press, 1992. ISBN 0-262-68052-1.

[183] Como a quantidade de números oraculares de Ω até Ω_{ω} é enumerável, os hiperprogramas de ordem ω ainda podem ter tamanho finito.

[184] O análogo para natureza do ser metavivos para os organismos.

organismo (hiperprogramas de ordem finita) se detém ou não. O que nos permite assumir uma natureza, ou melhor, uma **hipernatureza,** completa.

Os organismos ou seres vivos metabiológicos evoluiriam por meio de mutações algorítmicas aleatórias para chegar cada vez mais perto do poder computacional da natureza? As mutações são capazes, como já sabemos pela metabiologia, de adicionar complexidade aos seres, porém, como as mutações iriam mudar a ordem dos hiperprogramas? Note que pular a ordem em 1 é equivalente a injetar uma quantidade infinita de complexidade (ou informação algorítmica) de uma só vez. Isso, à primeira vista, poderia servir como um empecilho a tais mutações. Contudo, o truque está no fato de que os hiperprogramas são *strings* finitas, logo, por definição, eles não carregam toda essa complexidade com eles. Apenas é necessário que as mutações mudem a forma da *bit string* de um hiperprograma de ordem n para que ela passe a ser da forma de um hiperprograma de ordem $n + 1$. Quando este for rodado, a natureza dará cabo de informar os bits dos números oraculares.

Outra questão seria como definir uma probabilidade sobre os hiperprogramas, mesmo estes sendo autodelimitados e completos[185]. Da teoria algorítmica da informação, teríamos que a soma das probabilidades de todos os hiperprogramas de ordem n daria 1. Logo, a soma das probabilidades de todos os hiperprogramas de qualquer ordem finita daria $1 + 1 + \cdots + 1 + \cdots = \infty$. Portanto, assim como no início da teoria algorítmica da informação foi necessário introduzir o conceito de programa autodelimitado, aqui, possivelmente, se fará necessário o conceito de hiperprograma auto-ordenado. Isto é, aquele que informa sua própria ordem em sua *bit string*. O que será explicado logo adiante.

3.3 DEFINIÇÕES

3.3.1 Um hiperprograma é uma *bit string* finita de uma máquina oracular de Turing de qualquer ordem. Em outras palavras, um hiperprograma é um programa de máquina oracular de Turing.

---

[185] Ver seção 3.4.

Nós podemos definir mais precisamente os hiperprogramas apenas por programas com sub-rotinas que podem chamar qualquer $n-ézimo$ bit de um número real. Esses números reais precisam ser oráculos, correspondendo aos graus de Turing. Pode-se formalizar uma máquina universal de Turing de várias fitas,[186] em que todas, menos a primeira, são preenchidas com números reais oraculares. O número de fitas, além da primeira, dita a ordem da máquina oracular de Turing.[187]

Por exemplo, um hiperprograma de ordem 1 pode ter acesso ao número Ω de Chaitin como seu oráculo. Um hiperprograma de ordem 2 tem acesso a Ω e a Ω_2. E assim por diante.

3.3.2 Dado um grau de Turing n, nós denotamos a máquina oracular de Turing de ordem n por U_n.

3.3.3 Dizemos que uma máquina oracular de Turing de ordem 0 é a máquina de Turing universal usual.

3.3.4 Dada uma hierarquia de números ômega de Chaitin $\Omega_0, \Omega_1, \Omega_2, ...$ nós definimos a correspondência: $U = U_0$, $U_1 = U_\Omega$, $U_2 = U_{\Omega,\Omega_2}$, $U_3 = U_{\Omega,\Omega_2,\Omega_3}$ e assim por diante. Os respectivos números ômega podem ser usados como oráculos para seus respectivos hiperprogramas.

3.3.5 Denotamos um hiperprograma h de uma máquina oracular de Turing de ordem n por $h^{(n)}$.

---

[186] O que é bem conhecido na literatura.
[187] No caso do problema abordado neste capítulo, todos os "organismos" terão acesso somente a um número finito de fitas. Já a natureza pode ter acesso a um número infinito enumerável de fitas.

3.3.6 Dizemos que um conjunto de hiperprogramas é auto-ordenado se existir um programa (hiperprograma de ordem zero) que pode sempre decidir qual é a ordem de cada hiperprograma desse conjunto (que são *bit strings* finitas) quando cada um é dado como input a esse programa.

3.3.7 Seja $S_{HP} = (h_1, h_2, h_3, \dots)$ uma sequência recursiva de hiperprogramas de ordem finita tal que estes são autodelimitados, auto-ordenados e[188]

$$\sum_{i=1}^{\infty} 2^{-|h_i|} = 1$$

3.3.8 Dizemos que um hiperprograma $h^{(n)}$ se detém se existir uma *bit string* x tal que $U_n\left(h^{(n)}\right) = x$. Isto é, um hiperprograma para se, e somente se, ele parar em sua respectiva máquina oracular de Turing.

3.3.9 Dizemos que um hiperprograma h' de S_{HP} é **recursivamente construtível** a partir de x se existir um programa que retorna a *bit string* h' como output quando x é dado como input. Isto é, existe $h^{(0)}$ tal que $U_0\left(h^{(0)} \circ x\right) = h'$.

3.4 PRELÚDIO DA PROVA

Antes de começarmos a demonstração, é imprescindível entender a definição 3.3.7. As noções de programa autodelimitado e a soma das probabilidades algorítmicas já são

[188] Mostraremos adiante que essa definição não é vácua, pois podemos definir uma sequência de hiperprogramas satisfazendo a esses três requisitos.

conhecidas e facilmente estendíveis à definição de hiperprogramas, já que estes são também *bit strings* finitas.

Mas introduzimos um termo novo: hiperprograma auto-ordenado. Assim como a autodelimitação, a auto-ordenação também é uma cláusula de "boa formação" das *bit strings* (algo análogo às *well-formed formula*s da lógica matemática). Da mesma forma que existe um programa tal que, dado qualquer programa autodelimitado como input, ele pode decidir se este é um programa autodelimitado, dadas certas regras de linguagem previamente, e informar seu tamanho (sem ter que rodar o programa[189]), existe um programa tal que, dado qualquer hiperprograma auto-ordenado como input, ele pode decidir se este é um programa auto-ordenado conforme as regras da linguagem e informar sua ordem sem ter que rodar o hiperprograma.

De qualquer forma, um exemplo trivial de linguagem base de programação satisfazendo as três condições seria a concatenação de $0\ldots01\circ P$, onde P é um hiperprograma autodelimitado não auto-ordenado de uma linguagem completa e o número de zeros antes do primeiro número 1 diz a ordem de P.[190] Como a soma de *bit strings* autodelimitadas de uma linguagem completa dá 1 e vale a soma $\frac{1}{2^1}+\frac{1}{2^2}+\frac{1}{2^3}+\cdots=1$, então a soma das probabilidades algorítmicas de todos os hiperprogramas da forma $0\ldots01\circ P$ dará também 1.

Para especificarmos mais ainda nossa linguagem de programação, fazemos com que U_n reconheça sempre hiperprogramas de ordem $>n$ como *bit strings* "mal formadas", que não são hiperprogramas válidos para serem rodados. Obviamente, um hiperprograma de ordem n pode computar qualquer hiperprograma de ordem $\leq n$. Note que, como a natureza é um hiperprograma de ordem ω, ela reconhece qualquer hiperprograma de ordem $<\omega$ como uma *bit string* bem formada.

3.5 A COMPLEXIDADE DA COMPLETUDE

[189] Como é mais comum de se definir a linguagem. Ver CHAITIN, G. J. **Algorithmic Information Theory**. 3ª. ed. Yorktown Heights: IBM, P O Box 218, 2003.

[190] Portanto, um hiperprograma de ordem zero teria a forma $1\circ P$, um hiperprograma de ordem 1 teria a forma $01\circ P$, um de ordem 2 teria a forma $001\circ P$ e assim por diante.

Assim como a natureza nos primeiros modelos de Chaitin tem sua função de aptidão $BB'(N)$, nossa natureza terá a função $BB_\omega(N)$.

A função de aptidão $BB_\omega(N)$ é definida justamente para lançar mão da mesma ideia que a *Busy-Beaver* possui de nos dar uma forma de medir a complexidade de um programa. Cabe lembrar também da seção 1.2.3 desta tese. Ali, podem ser encontradas maiores explicações sobre o assunto. Se um hiperprograma de ordem finita possui um output maior ou igual a $BB_\omega(N)$, quer dizer que nenhum programa de tamanho $\leq N$ pode produzir um output que seja maior que o dele. Além disso, se um hiperprograma de ordem finita possui um output estritamente maior que $BB_\omega(N)$, quer dizer que nenhum programa de tamanho $\leq N$ pode produzir um output que seja maior ou igual ao dele. Ou seja, para atingir uma aptidão maior que $BB_\omega(N)$, obrigatoriamente, precisamos de um hiperprograma de ordem finita com tamanho maior que N. E $BB_\omega(N)$ não poderá ser comprimido por um hiperprograma de tamanho muito menor que N. Portanto, a partir da comparação do output de um hiperprograma com a função $BB_\omega(N)$, podemos "medir" a complexidade ou "criatividade" desse hiperprograma.

É importante relembrar que à medida que o valor de BB_ω cresce, maiores hiperprogramas serão obrigatoriamente contemplados e, por isso, também hiperprogramas, de ordem finita, cada vez maiores. Com isso, conforme essa função cresce, obrigatoriamente passaremos por um hiperprograma com ordem e complexidade suficientes para resolver qualquer problema em qualquer grau de Turing finito. Em outras palavras – fazendo a correspondência com a hierarquia aritmética –, conforme essa função cresce, obrigatoriamente, passaremos por um hiperprograma com ordem e complexidade suficientes para resolver qualquer problema aritmético.

Logo, se a evolução de hiperprogramas por mutações aleatórias levá-los a calcularem maiores e maiores valores de BB_ω, então essa evolução estará sempre nos levando em direção à completude da aritmética. O que é nosso objetivo.

Feito isso, para qualquer N que se queira, vamos nos focar em estimar a quantidade média de mutações necessárias (o tempo de mutação) para se chegar a uma aptidão $BB_\omega(N)$.

3.5.1 **Definição**

Seja $P_{HP}^{(0)}$ o programa que enumera o conjunto S_{HP}.

- Nota:

Lembre que S_{HP} também é recursivamente enumerável. Logo, a definição não é vácua.

3.5.2 **Definição**

Seja $BB_{\omega}(N)$ uma função que retorna o maior output de qualquer hiperprograma $h \in S_{HP}$ tal que $|h| \leq N$.

- Nota:

Para essa função estar bem definida basta que as máquinas oraculares de Turing U_0, U_1, U_2, ..., U_{ω} também estejam bem definidas sobre a linguagem S_{HP}.

3.6 AS IDEIAS CHAVES DA PROVA

Podemos dividir o problema central da prova em dois. Primeiro, precisamos construir um hiperprograma de ordem $o_m + 1$ que seja capaz de decidir se qualquer hiperprograma de ordem $\leq o_m$ se detém ou não, e seja capaz de calcular $BB_{\omega}(N)$.[191] Essa ordem o_m é dada pelo programa no lema 3.9.1 Basicamente, a maior ordem que os hiperprogramas de tamanho $\leq N$ possuem. Dessa forma, $o_m + 1$ se torna a ordem mínima necessária para calcular, em todos os casos, $BB_{\omega}(N)$. Segundo, precisamos mostrar que um programa, com N dado como input, pode construir – ou nomear – (por isso o termo "construído recursivamente") esse

[191] Iremos definir a ordem máxima o_m logo abaixo.

hiperprograma de ordem $o_m + 1$ (para construir uma *bit string* não quer dizer que o programa precise ser capaz de rodá-la).

Podemos começar perguntando se existe um programa que, dado N como input, pode retornar a ordem o_m necessária para cobrir todos os hiperprogramas que param envolvidos em $BB_\omega(N)$. A resposta é sim. Para tal, basta saber qual a ordem máxima dos hiperprogramas de tamanho $\leq N$. É justamente isso que o programa $P_{MO}^{(0)}$ faz.

Em seguida, definimos outro programa $P_{nN}^{(0)}$ que, com N e com o valor dado por $P_{MO}^{(0)}$ como inputs, retorna todos os hiperprogramas de S_{HP} com tamanho $\leq N$ e ordem $\leq o_m$. Pela construção de o_m, esse programa retornará todos os hiperprogramas de tamanho $\leq N$.

Assim, tomemos agora um hiperprograma $P_H^{(o_m+1)}$ que pega essa sequência de hiperprogramas dada por $P_{nN}^{(0)}$, roda todos esses hiperprogramas e retorna o maior output que eles calcularem. Caso algum deles nunca pare, o oráculo de ordem $o_m + 1$, ao qual $P_H^{(o_m+1)}$ tem acesso, pode responder.

Como $P_H^{(o_m+1)}$ é capaz de contemplar todos os hiperprogramas de tamanho $\leq N$, podemos montar[192] outro programa $P_T^{(o_m+1)}$ que, com N como input e baseado na quantidade de hiperprogramas dada por $P_H^{(o_m+1)}$, nos retorne um valor $\geq BB_\omega(N)$.[193] Em notação mais precisa, teremos:

$$U_{U_0\left(P_{MO}^{(0)}\circ N\right)+1}\left(P_T^{\left(U_0\left(P_{MO}^{(0)}\circ N\right)+1\right)}\circ N\right) = U_{U_0\left(P_{MO}^{(0)}\circ N\right)+1}\left(P_H^{(n+1)}\circ\left(U_0\left(P_{nN}^{(0)}\circ U_0\left(P_{MO}^{(0)}\circ N\right)\circ N\right)\right)\right) \geq BB_\omega(N)$$

O que finaliza a primeira parte da prova.

Para demonstrar a segunda parte, isto é, que $P_T^{(o_m+1)}$ é uma *bit string* que pode ser construída por um programa $P_{P_T}^{(0)}$ a partir de N como input, basta notar que os processos que efetuamos (e que deixamos omitidos por já serem conhecidos na literatura) para provar que existe tal e tal hiperprograma, nos diz como deve ser o programa ou quais sub-rotinas que

[192] Ver definição 3.4.9.

[193] A rigor esse hiperprograma será da forma $P_T^{\left(U_0\left(P_{MO}^{(0)}\circ N\right)+1\right)}$.

chamam números oraculares. Além disso, o_m é um valor construtivo de forma trivial a partir de um programa com N como input. Note que para montar o hiperprograma $P_H^{(o_m+1)}$ não é necessário saber os bits dos números oraculares, só se precisa saber o programa que chama os n primeiros bits de um número real binário. Isto é, só precisamos montar um programa de uma máquina universal de Turing com várias fitas (a principal, onde vai o programa, e outras extras, as quais podem conter *bit strings* infinitas arbitrárias), de forma que as fitas extras sejam preenchidas, no nosso caso, por números oraculares.

Por fim, seja $P' \circ P_{P_T}^{(0)} \circ N$ um hiperprograma (de ordem zero) que ignora seu input e que retorna $U_0\left(P_{P_T}^{(0)} \circ N\right)$. Ou seja, retorna a *bit string* $P_T^{(o_m+1)} \circ N$, a qual passará a ser o próximo organismo.[194] Como $P' \circ P_{P_T}^{(0)} \circ N$ é um hiperprograma (em particular, de ordem zero), ele pode ser uma mutação algorítmica. Seu tamanho será da ordem de $C + H(N)$, onde C é uma constante. Por uma estimação já conhecida na teoria da informação algorítmica, teremos que a probabilidade desse hiperprograma de ordem zero ocorrer será

$$> \frac{1}{C' * N * (\log_2 N)^{1+\epsilon}}$$

onde C' e ϵ são constantes. Logo, a quantidade média de tentativas para que ocorra essa mutação será $< C' * N * \log_2 N^{1+\epsilon}$. Um tempo bem próximo de ser linear, o que é bem rápido se comparado com o primeiro modelo de *busca exaustiva,* e mesmo com o modelo de *evolução cumulativa*, os quais Chaitin já apresentou. Ou seja, o modelo de *busca exaustiva para hiperprogramas* é, aproximadamente, tão ou mais rápido que uma função linear em N.

3.7 PROVOCAÇÕES A UM UNIVERSO NÃO COMPUTÁVEL

[194] É bom lembrar que todo organismo é um hiperprograma de ordem finita a ser rodado pela natureza, um hipercomputador.

A primeira coisa que um leitor poderia, agora, se perguntar é que tipo de biologia ou natureza suportaria seres vivos com tamanha capacidade de cômputo? Tomando como verdadeira a hipótese de que o modelo para a evolução da vida proposto neste capítulo é fidedigno à biologia do "mundo real", os organismos seriam, no mínimo, sistemas não computáveis. Seriam sistemas físico-químicos tão complexos que nenhum computador, por mais poderoso que fosse, poderia dar conta de emulá-los.

Nesta seção lançaremos algumas provocações acerca dos resultados alcançados neste capítulo. Por essa razão, torna-se mais importante para nós apresentá-las e colocá-las em relação com o presente trabalho do que discutir a fundo cada item aqui mencionado. Ademais, muitas delas, por serem posições filosóficas e/ou objetos de investigação científica, ainda propagam profundas discussões em suas respectivas áreas.

Primeiramente, devemos voltar a dizer que, aqui, estamos deixando de lado qualquer outra propriedade qualitativa que possa emergir dos sistemas biológicos, da mesma forma que discutimos na seção 1.4, a qual também está intimamente ligada com as questões de agora. Por exemplo, deixamos de lado propriedades hipotéticas como "alma", "espírito" ou qualquer tipo de dualismo que não entenda um ser vivo apenas como um sistema de relações entre objetos, conhecidos ou não – enfim, que não entenda os seres vivos como sistemas.

Como já falamos no capítulo anterior, esse tipo de visão pode ser chamada de 'funcionalismo'. Mas não assumimos essa postura filosófica porque desconsideramos as outras e sim porque é a que compete ao campo de discussão que esta tese abrange. Em última instância, tanto o cômputo quanto o "hipercômputo" são processos que correspondem à modificação de estados de um sistema. Estados estes que são compostos apenas por "coisas", objetos ou entes. Não importaria do que esse sistema é "feito", mas somente as relações entre as "coisas" do que ele é "feito". Tudo isso vale, com certeza, uma investigação filosófica mais aprofundada tanto na biologia, física e ciência cognitiva, porém, tal propósito extrapola o escopo de presente estudo.

É claro que uma das hipóteses que perpassam todo o texto é a tese de Church-Turing. Aqui, nós a assumimos. O que nos permite passar do âmbito algorítmico para o recursivo dos sistemas formais da lógica matemática e, então, para o computacional. O caminho inverso também é válido. Por tratar de algoritmos, ela tem um dos pés fincados na informalidade da "não cientificidade". A noção de algoritmo permanece como algo autoevidente para a nossa intuição, ao mesmo tempo que permanece " não formalizável". De qualquer forma, por exemplo, o conceito de máquina oracular de Turing e a própria equivalência entre os graus de Turing e a hierarquia aritmética – a qual vamos utilizar largamente a partir de agora –

necessitam, em sua fundamentação filosófica, que esta tese seja considerada um ponto de partida, tácito ou não.

Dito isto, voltemos à pergunta: os seres vivos são sistemas incomputáveis? A primeira coisa a ser considerada é que a incomputabilidade relativa e os resultados do Capítulo 1 corroboram com uma biologia computável, ao mesmo tempo em que ela nos pareça não ser. A segunda, não menos importante, é que não é porque um resultado corrobora com alguma hipótese que essa hipótese é verdadeira ou mais provável. Tratar o assunto desse modo seria uma "falácia epistemológica" de nossa parte.

Em particular, o fenômeno da incomputabilidade relativa metabiológica serve como uma contra-resposta a algumas críticas à computabilidade da vida, i.e., dos organismos. Essa crítica em geral é feita levantando argumentos a favor de uma não computabilidade dos fenômenos biológicos. Suscitaremos alguns deles.

Por exemplo, a vida poderia ser analógica. Por isso, as grandezas que medem as propriedades físico-químicas assumiriam valores dentro dos números reais, o que, em muitos casos, tornaria impossível para qualquer cômputo capturar todas as relações. Se estas tiverem uma cardinalidade não enumerável, não discreta, enquadram-se nesse caso. Um exemplo matemático dessa possibilidade seria considerar as redes neuronais dos animais como sendo redes neurais (aquelas largamente utilizadas em computação) cujos pesos podem assumir valores reais. Isso permitiria a elas realizarem hipercômputos ou, nas palavras de Siegelmann e Sontag, permitiria que elas fossem "*super-Turing machines*".[195]

Obviamente, a vida sendo analógica implica o Universo ser também analógico, pois o último contém a primeira. Aí, se iniciam também novas discussões sobre a continuidade ou "discretude" da física. Por um sentido inverso de argumentação, o mundo físico sendo, de alguma maneira[196], incomputável permitiria que os sistemas vivos também o fossem. Uma possibilidade de construir um hipercomputador real, caso essa hipótese seja suficientemente forte, pode ser vista em Doria (2006)[197]. Neste caso, um sistema físico que seja capaz de realizar cômputos e solucionar certo tipo de cálculo integral, poderá superar o limite do computável. Mesmo que isso permaneça como uma hipótese – inclusive, sendo de suma

[195] SIEGELMANN, H. Neural and Super-Turing Computing. **Minds and Machines**, n. 13, 2003. 103-114
[196] Por exemplo, seguindo a hipótese de Penrose. Ver SZUDZIK, M. The Computable Universe Hypothesis. **arXiv**, Dezembro 2010.
[197] DA COSTA, N. C. A.; DORIA, F. A. Some thoughts on hypercomputation. **Applied Mathematics and Computation 178**, 2006.

importância um teste experimental – ainda cabe a pergunta: por que os seres vivos não poderiam ter encontrado um meio de realizar essa ou outra tarefa não computável?

Na própria biologia existe a afirmação famosa de Jaques Monod[198] de que a sequência de aminoácidos em uma proteína se apresenta de forma aleatória. Dada uma cadeia de 200 aminoácidos em uma proteína, mesmo sabendo os 199 primeiros, não haveria procedimento nem regra que possa prever qual será o último aminoácido. Independentemente da extensão da validade dessa tese científica,[199] vamos supô-la, por um instante, como verdadeira, ou supor que, pelo menos, é, de fato, dificílimo para nós e nossos computadores decodificarmos as proteínas. Então, como as proteínas são sintetizadas pelo próprio organismo, podemos dizer que grande parte dela foi codificada pelo "programa"[200] do organismo ou, em nossas palavras, *codificadas pelo ser metavivo*. Algo aleatório para nós e para nossa computação, mas que é "computado" pelo ser vivo nos levando a crer que o ser vivo não pode ser computável. Ele deveria, então, hipercomputar.

Tudo isso são campos de discussões, quem sabe, infindáveis do mundo científico e filosófico atual. Todavia, o que podemos dizer neste trabalho é que os resultados do Capítulo 1 apenas colocam de volta o papel do observador em todas essas questões: é possível nós, seres vivos, "observarmos" o mundo biológico ou físico como não sendo computável, embora o mundo biológico ou físico seja computável. Por quê? Poderíamos ser subcomputadores, ou subsistemas, contidos em um sistema físico-químico que suporta a vida e que é computável, porém, num nível acima do nosso.[201] O que, por mais esquisito ou irônico que possa parecer, faria a **hipótese da vida computável**[202] ser consistente, inclusive, com os argumentos contra a computabilidade da vida.

Contudo, podemos defender a posição do parágrafo anterior – ou apostar nela? Não. Nosso resultado só diz que existe mais uma possibilidade, mas não afirma que é a mais provável ou a correta. Portanto, sim, os seres vivos podem não ser computáveis, eles podem ser hipercomputadores. Eles podem não parecer computáveis e "realmente" não ser

---

[198] MONOD, J. **Chance and Necessity:** An Essay on the Natural Philosophy of Modern Biology. New York: Alfred A. Knopf, 1971. ISBN 0-394-46615-2.
[199] HUNTER, C. Fred Sanger, Protein Sequences and Evolution Versus Science. **Darwin´s God**, 2013. Disponivel em: <http://darwins-god.blogspot.com.br/2013/11/fred-sanger-protein-sequences-and.html>. Acesso em: 17 Março 2015.
[200] As aspas estão aqui porque pode ser, na verdade, um hiperprograma.
[201] O qual, por exemplo, poderia ser capaz de computar funções que seriam incomputáveis para nós.
[202] A hipótese de que os seres vivos seriam sistemas computáveis.

computáveis. É uma possibilidade que devemos considerar de forma equivalente. Pois, essas duas possibilidades antagônicas nos parecem, até o momento, "equiconsistentes" com a biologia.

E agora? Como lidamos com isso? Como dito no início deste capítulo, é nessa linha que a investigação sobre a evolução de hiperprogramas se torna necessária. E foi essa nossa intenção: mostrar que é possível uma evolução – por sinal, muito rápida – de **hiperprogramas** ou **hiperorganismos** de forma a cobrir todo o espaço de problemas sobre verdades matemáticas usuais, como a aritmética. Uma tarefa consagrada na literatura como altamente incomputável.

Assim, escolhemos trabalhar com hiperprogramas de ordem finita, correspondendo-os aos problemas na hierarquia aritmética. A natureza precisa se tornar um hiperprograma mais poderoso ainda, de ordem infinita[203], justamente porque ela precisa saber se qualquer hiperorganismo produz um output ou não, e saber calcular qual é esse output. A nossa **hipernatureza** precisa ter acesso a todos os oráculos de ordem finita, precisa ser capaz de rodar qualquer hiperprograma de ordem finita.

Vamos fazer uma ressalva agora.

Para um leitor atento, é natural se perguntar: por que usamos hiperprogramas auto-ordenados? Sabemos que, na literatura, essa não é a forma usual de se trabalhar com máquinas oraculares de Turing (programas com sub-rotinas que podem consultar oráculos para realizar seus cômputos). Nossa demonstração necessita dessa propriedade para funcionar. Do contrário, os hiperprogramas poderiam ser autodelimitados, com a soma da probabilidade de todos resultando em 1, mas que, somente quando rodados por máquinas oraculares de Turing[204] elas possam saber qual a ordem destes.[205] Assim, o argumento matemático deste capítulo não se aplicaria. Talvez, para esse caso, possamos provar semelhante resultado, ao qual chegamos, utilizando mutações que sejam hiperprogramas de qualquer ordem finita e não apenas mutações algorítmicas (hiperprogramas de ordem zero). Esse é um problema matemático a ser investigado no futuro.

Cabe lembrar que a mesma coisa vale para a autodelimitação de hiperprogramas. Podemos, sim, trabalhar com hiperprogramas cujo tamanho só pode ser determinado quando

---

[203] Sua ordem é o primeiro ordinal limite: ω.
[204] De ordem igual ou superior.
[205] Quando só se pode saber a ordem de um hiperprograma rodando-o, diz-se que ele é ordenado *on the fly*.

rodados por sua respectiva máquina oracular e ainda assim serem autodelimitados. No entanto, para o presente trabalho, um caso ou o outro não influencia no resultado final, pois, S_{HP} é recursivo.[206]

De qualquer forma, a auto-ordenação não nos é, de forma alguma, uma escolha arbitrária para fazer a prova funcionar. Pelo contrário, está calcada em hipóteses e teses além da metabiologia. Podemos elencar alguns argumentos que podem corroborar com a auto-ordenação dos hiperorganismos, porém, com certeza, esses são temas que ainda carecem de muita discussão. Pode ser que um modelo satisfatório para a evolução de hiperseres vivos não se baseie em hiperprogramas auto-ordenados. Todavia, vamos deixar essa possibilidade para o futuro e nos fixar em tentar fundamentar um pouco melhor a hipótese contrária.

A primeira questão a favor da auto-ordenação é bastante pragmática em relação à matemática. Note que, desconsiderando somente a propriedade auto-ordenativa, a soma de todas as probabilidades de todos os hiperprogramas autodelimitados de ordem n será 1. A soma de todas as probabilidades de todos os hiperprogramas de ordem $n + 1$ será 1. E assim por diante. Logo, a soma de todas as probabilidades de todos os hiperprogramas de ordem finita dará infinito e não 1. O que é tudo que não queremos ao definir um espaço de probabilidades. Por causa disso, realmente, adicionar um prefixo autodelimitado que informa a ordem máxima dos oráculos de ordem finita que podem ser consultados pelo hiperprograma à frente é uma excelente forma de fazer a soma final dar 1 e não infinito.[207] Um problema matemático interessante para o futuro se torna, também, averiguar se a soma final resultar 1 implica ou não algum tipo de auto-ordenação.

Se formos, agora, pensar em termos de engenharia da computação – ou engenharia da hipercomputação, se um dia houver –, podemos adquirir uma intuição sobre o assunto que estamos falando. Vamos fazer, então, uma experiência mental. Se chegarmos a construir um hipercomputador de ordem 1, o qual "provarmos"[208] que é um hipercomputador de ordem 1 por meios teóricos e científicos, não é absurdo dizer que "saberemos" diferenciar esse hipercomputador de um computador normal já conhecido.

---

[206] Claro, S_{HP} pode não ser recursivo num modelo mais adequado para os organismos. E isso também invalidaria os teoremas deste capítulo. Porém, muitas das justificativas para a auto-ordenação também têm respaldos análogos para a recursividade de S_{HP}. Principalmente a do último parágrafo desta seção.
[207] Ver seção 3.4.
[208] Que seja algo amplamente aceito pela comunidade científica ou geral.

Mas como “provamos” tal coisa? Se os próprios métodos e hipóteses que nos levarem a essa convicção forem algo que reconhecemos e for algo que consigamos classificar, eles mesmos já serviriam como uma forma de diferenciar um hipercomputador de um computador. Portanto, em tese, poderíamos elaborar um procedimento recursivo – ou algorítmico, se preferir – que possa diferenciar tal hipercomputador de um computador comum. Se esse procedimento recursivo existir, então os hiperprogramas desses hipercomputadores hipoteticamente reais serão auto-ordenados. Em particular, pois, existe um programa que calcula a ordem de seus hiperprogramas. O que cai na definição de auto-ordenação. Esse programa é justamente dado pelo procedimento recursivo acima.[209]

Continuando nessa linha de pensamento, se construirmos, posteriormente um hipercomputador de ordem 2 que “sabemos”, pelos mesmos[210] meios científicos e teóricos, que é um hipercomputador de ordem 2, então seus hiperprogramas serão também auto-ordenados, por motivos análogos ao caso do parágrafo anterior.

Fazendo essa experiência mental para qualquer hipercomputador de ordem finita, conseguimos uma hierarquia de hipercomputadores hipoteticamente reais para os quais um modelo com hiperprogramas auto-ordenados faz total sentido. Agora, troque o papel de “construtor”. Coloque a natureza no lugar de “nós”, do início da vida até hoje. Um modelo com hiperprogramas auto-ordenados também faria total sentido.

Vamos para um caso extremo agora.

Suponha um espaço de hiperprogramas / hiperorganismos de ordem finita tal que não exista nenhum hiperprograma de ordem finita que possa dizer a ordem dos hiperorganismos. Isto é, a ordem deles só poderia ser dita pela própria hipernatureza, uma máquina oracular de Turing com acesso a todos os oráculos de ordem finita. Mas lembre-se que a hipernatureza só permite hiperprogramas de ordem finita como organismos, portanto, ela precisa excluir todos aqueles que chamam uma quantidade infinita de oráculos, pois, estes hiperprogramas nunca parariam. Seria a hipernatureza capaz de decidir sempre se um hiperprograma chama infinitos oráculos de ordem finita ou não? Esse é mais um problema matemático aberto à investigação. Se a resposta for negativa, a hipernatureza não poderá cumprir seu papel em diversos casos. Em particular, naqueles em que o hiperorganismo chama infinitos oráculos e ela não consegue

---

[209] Lembre-se da tese de Church-Turing.

[210] Não necessariamente os mesmos. Mas a teoria envolvida para “provar” a ordem dos hipercomputadores hipoteticamente reais precisaria ser axiomatizável ou computável. O que é não é uma hipótese absurda.

saber que ele o faz. Ela precisaria “matar” ou “descartar” esses hiperprogramas, mas não saberia que deve fazê-lo. Portanto, no caso dessa resposta negativa, se faria necessário algum tipo de auto-ordenação.

São hipóteses e possibilidades. De qualquer modo, a auto-ordenação que usamos evita esses problemas de forma econômica e simples. Somente se requer que exista um programa que possa saber a ordem de qualquer hiperprograma sem precisar rodar esse hiperprograma numa máquina oracular de Turing. Além disso, tudo o que hiperprogramas de ordem finita não auto-ordenados podem calcular, os hiperprogramas auto-ordenados também podem, e vice-versa. À guisa de verificação, basta adicionar um prefixo de auto-ordenação – como fizemos neste capítulo – com a ordem máxima de um hiperprograma de ordem finita não auto-ordenado a este último. O hipercômputo será o mesmo. Do ponto de vista de simplicidade e economia de recursos computacionais, não haveria mais vantagem para a natureza de ordem infinita criar a vida com organismos que são hiperprogramas não auto-ordenados do que com auto-ordenados. Mas não fazemos a mínima ideia se a natureza biológica possui, de fato, essas duas propriedades. Por enquanto, só poderíamos dizer que, para nós, isso é mais razoável, e apenas isso.

3.8 O QUE PODEMOS PENSAR A PARTIR DAQUI?

Nós mostramos que é possível a evolução de hiperprogramas. Em particular, provamos que existe um programa, com N como seu input, que retorna um organismo/hiperprograma que pode calcular um número maior ou igual a $BB_{\omega}(N)$ como seu output. Tal organismo/hiperprograma tem ordem e complexidade suficientes para saber o output de qualquer hiperprograma de tamanho $\leq N$, i.e., se eles param ou não.

Para calcular valores cada vez maiores de BB_{ω} precisamos de hiperprogramas/organismos de tamanhos cada vez maiores também. Para todo problema aritmético existe um grau de Turing finito correspondente capaz de contê-lo; e para todo grau de Turing finito existe um hiperprograma de ordem finita que é capaz de dar conta dele. Por isso, conforme BB_{ω} cresce, passaremos por todos os hiperprogramas capazes de resolver problemas aritméticos. Em consequência disso, mostramos que é possível uma evolução de hiperprogramas capaz de nos levar em direção à completude da aritmética – e ela ocorre com bastante rapidez.

Como o **tempo de mutação** (a quantidade média de mutações algorítmicas aleatórias necessárias) para o programa/mutação que construímos aparecer aleatoriamente é $< C * N * (\log N)^{1+\epsilon}$, podemos dizer que a evolução metabiológica de hiperprogramas que definimos aqui ocorre em tempo quase linear em N, ou até mais rápido.

É sempre importante lembrar que o caso que estudamos neste capítulo foi análogo à **busca exaustiva** de Chaitin[211]. No entanto, nesta os programas evoluem em direção a complexidade de Ω em tempo de mutação exponencial[212] em N. O próprio modelo de **evolução cumulativa** chaitiniano, apesar de mais rápido que o exaustivo, evolui em tempo de mutação pouco maior que quadrático em N. Ao que parece, quando se põem oráculos no "jogo da evolução" de seres metavivos, a própria evolução[213] ocorre com mais rapidez. Obviamente, estudar por que tanta diferença do caso chaitiniano para o caso de "hiperorganismos", o qual mesmo com mutações "cegas" pode evoluir mais rápido que em tempo linear, pode ser um tópico interessante de investigação teórica.

Outra questão que se pode extrair deste capítulo – talvez até mais filosófica que puramente matemática – é que pudemos enumerar hiperprogramas que se aproximam sempre da completude da aritmética. Mas os problemas aritméticos não podem ser resolvidos por nenhuma função recursiva. Então, como pode existir um programa que enumera hiperprogramas (os quais podemos entender também como funções) capazes de resolver o problema aritmético desejado? Nenhum programa pode ir resolvendo todos os problemas na hierarquia aritmética, porém, existe uma enumeração de hiperprogramas que, por sua vez, podem! No fundo, essa enumeração de hiperprogramas se calca numa enumeração de oráculos.

Fazendo uma metáfora: não podemos dar infinitos passos, porém, podemos falar sobre alguém que pode. Talvez, isso esteja intimamente ligado ao "fato" de que mesmo que um matemático não possa sempre saber se uma sentença aritmética é falsa ou verdadeira, ele pode definir ou construir um sistema que pode – e isso tem sido feito bastante em lógica matemática.[214] O capítulo 3 acentua essa hipótese. Programas também podem falar sobre a

---

[211] CHAITIN, G. Life as Evolving Software. In: ZENIL, H. **A Computable Universe:** Understanding and Exploring Nature as Computation. [S.l.]: [s.n.], 2012. p. 277-302. ISBN 978-9814374293.
[212] Especificamente, de base 2.
[213] De hiperorganismos, sistemas incomputáveis.
[214] Mais sobre o tema pode ser visto em ABRAHÃO, F. S. **Demonstrando a Consistência da Aritmética**. Dissertação de mestrado. ed. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2011.

completude, sem serem, na verdade, completos. Aqui, outra vez, nos entrelaçamos com as questões do capítulo 1. Além disso, note que mais uma vez nos deparamos com um "dança" entre **língua** e **metalíngua.** Pode ser que tais coisas também tenham a ver com nossos **pseudoparadoxos metalinguísticos**. Um campo de investigação futuro poderia ser demonstrar se essa hierarquia de hiperprogramas pode ser relativizada da mesma forma que relativizamos a incomputabilidade no capítulo 1. Algo que já chamamos de **hipercomputação relativa recursiva.**[215] E qualquer semelhança com os problemas da filosofia da linguagem sobre semiótica e teoria dos conceitos não é mera coincidência.

Vamos "pensar mais alto" então: nós ainda conseguiríamos ir além da complexidade suficiente para obtermos a completude da aritmética? Existiriam funções ainda não computáveis por essa "hipernatureza" – um hiperprograma de ordem ω? Note que nesse capítulo os organismos/hiperprogramas estão limitados para serem sempre de ordem $< \omega$.

De fato, por exemplo, existe um número não enumerável de subconjuntos dos números naturais que **não podem** ser definidos por sentenças aritméticas. Isto é, o número de subconjuntos dos números naturais sobre os quais não podemos perguntar é muito maior que o daqueles que podemos. Nossa "hipernatureza" de ordem ω só é capaz de computar uma quantidade enumerável de problemas, o que estaria muito longe ainda do que desejamos agora. Isso tudo está refletido também nos graus de Turing, pois existe uma quantidade não enumerável (do tamanho do contínuo) desses graus.[216]

Logo, pulando várias outras questões técnicas da matemática, chegaríamos à necessidade de um hiperprograma/natureza de ordem, minimamente, ω^{ω}. O que supera todos os hiperprogramas de ordem transfinita que computam subconjuntos dos números naturais.

Pois, então, montamos outro problema maior para a metabiologia: considerando a natureza um hiperprograma de $U_{\Omega,\,\Omega_2,\Omega_3,\ldots,\Omega_{\omega^1},\Omega_{\omega^2},\Omega_{\omega^3},\ldots} \equiv U_{\omega^{\omega}}$, conseguiremos uma evolução de hiperprogramas (de ordem $< \omega^{\omega}$) capaz de ir computando todos os subconjuntos dos números naturais? Note que isso é equivalente a ir computando todos os números reais. Seria o análogo (e mais abrangente) do que fizemos para a aritmética no presente capítulo: uma evolução capaz de ir computando todas as verdades aritméticas.

---

[215] Vide seção 1.4.1.
[216] ROGERS, H. **Theory of recursive functions and effective computability**. 3 ed. [S.l.]: MIT Press, 1992. ISBN 0-262-68052-1.

Se o leitor, embarcado no espírito cantoriano desse capítulo, ainda estiver insatisfeito com tamanho nível de complexidade para a natureza – uma natureza capaz de computar todo número real – e se predispor, inclusive, a ultrapassar a própria concepção majoritária da física moderna sobre o quão complicado é o universo, ainda podemos ir além: subir a "escada" de todos os ordinais, como era desejo de Cantor. Qual ordem de hiperprograma seria capaz de computar todos os conjuntos que a matemática, ou o matemático, poderia conceber? Essa super "hipernatureza" precisaria ter uma ordem muito maior que ω, ω^{ω}, $\omega^{\omega^{\omega^{\omega}}}$ ou além. Poderíamos chegar a "computar" todos os conjuntos? Até onde a evolução de hiperprogramas pode nos levar? Qual a maior classe de questões indecidíveis da matemática que a evolução metabiológica é capaz de contemplar usando o conceito de hiperprogramas?

Este capítulo alonga muito mais o diâmetro do "abraço" que a metabiologia pode nos dar ao entendimento da vida e da Natureza. E o faz trazendo para nós a possibilidade de estudar matematicamente a evolução de sistemas tão complexos que podem ultrapassar, o quanto se queira, a fronteira do computável. É evidente também que essas questões reverberam muito nos problemas fundamentais da biologia e, também, da física. Justamente, elas perpassam a discussão sobre vida artificial indo até a computabilidade do Universo.

3.9 EM DETALHES MAIS FORMAIS

3.9.1 **Lema**

Existe um programa $P_{MO}^{(0)}$ que recebe N como input, enumera todo hiperprograma de S_{HP} com tamanho $\leq N$ usando $P_{HP}^{(0)}$ e retorna a maior ordem o_m que estes hiperprogramas têm.

- Notas:

Lembre que S_{HP} é, por definição, recursivo. Como todo hiperprograma é uma *bit string*, existirá um programa que dá todas as *bit strings* de tamanho $\leq N$ que pertencem a S_{HP}.

3.9.2 **Lema**

Seja $P_{nN}^{(0)}$ o programa que recebe n e N como inputs e retorna um *array* de hiperprogramas de S_{HP} que contém todos os hiperprogramas de S_{HP} com tamanho $\leq N$ e ordem $\leq n$.

- Notas:

Lembre que, por definição, S_{HP} é um conjunto recursivo, autodelimitado e auto-ordenado. Logo, decidir o tamanho e a ordem de um hiperprograma é uma tarefa recursiva. Posto isso, construir esse *array* como output se torna possível.

Podemos entender esse *array* como uma sequência de hiperprogramas. Por exemplo, a primeira *bit string* autodelimitada informa a quantidade de hiperprogramas subsequentes, então, concatenamos o primeiro hiperprograma com essa primeira *bit string;* daí, concatenamos o segundo hiperprograma; e assim por diante. O resultado é uma *bit string* finita contendo os hiperprogramas de S_{HP} que tem tamanho $\leq N$ e ordem $\leq n$. No entanto, qualquer que seja o processo recursivo para se montar esse *array* ele tem que ser o mesmo que o programa $P_H^{(n+1)}$, a seguir, é programado para reconhecer ao ler seu input.

3.9.3 **Lema**

Existe um hiperprograma $P_H^{(n+1)}$ recursivamente construtível a partir de n que recebe como input um *array* de hiperprogramas e retorna o maior output dos hiperprogramas de S_{HP} com ordem $\leq n$ que pertencem a esse *array*.

- As principais ideias dessa prova:

Esse *array* de hiperprogramas nada mais é que uma sequência concatenada de hiperprogramas de S_{HP} tal que ela informa quantos hiperprogramas estão nela. Caso contrário, $P_H^{(n+1)}$ não saberia quando parar de ler seu input.

Assim, o hiperprograma $P_H^{(n+1)}$ lê a si mesmo identificando sua própria ordem. Então, lê todos os elementos do *array* e seleciona apenas aqueles hiperprogramas com ordem $\leq n$. Lembre-se da definição de S_{HP}. Como $P_H^{(n+1)}$ é de ordem $n+1$ ele pode saber sempre se um hiperprograma de ordem $< n+1$ se detém ou não. Daí, pode-se calcular qual o maior output daqueles que produziram outputs.

Para provar que $P_H^{(n+1)}$ é recursivamente construtível a partir de n, basta primeiro notar que programar um programa para ler a si mesmo e identificar sua própria ordem, a qual já está dentro dos moldes da linguagem S_{HP} – que é recursiva por definição –, é, trivialmente recursivo. Identificar as ordens de outras *bit strings* de S_{HP} também será, por motivos parecidos, um processo recursivo. Portanto, somente precisamos construir um processo que consulte os oráculos de ordem $n+1$ para calcular os outputs desses hiperprogramas de ordem $\leq n$ dentro do *array*. Esse processo de "chamada" não depende dos oráculos nem do que eles dizem, depende apenas de como eles estão indexados (obrigatoriamente de forma recursivamente enumerável) na linguagem de programação.

Vamos explicar melhor. Conhecemos bem o programa H_Ω que usa Ω, por exemplo, para resolver o problema da parada. É com ele que provamos que é equivalente saber os dígitos de Ω e resolver o problema da parada.

O programa H_{Ω_2} que usa Ω_2 para resolver o problema da parada para U_1 é essencialmente o mesmo, só que relativo a U_1 ao invés de U. Em outras palavras, para enumerar os hiperprogramas de ordem 1 que param, H_{Ω_2} roda os hiperprogramas de U_1 no lugar dos programas de U (que é o que H_Ω faz) – e eles são basicamente programas de U que podem consultar uma fita extra qualquer, arbitrária[217]. No nosso caso, por construção, convencionamos que o que preenche essa fita são dígitos de Ω, porém, essa fita não pertence ao programa H_{Ω_2}. Se pertencesse, ele não seria um programa e sim uma máquina oracular de Turing. Quando a fita extra for, de fato, preenchida por Ω, então esse programa passa a satisfazer a nossa definição de hiperprograma.

[217] Lembrar da definição 3.3.1.

Por sua vez, teremos o programa H_{Ω_3} que será o mesmo que H_Ω só que relativo a U_2 ao invés de U.[218] Para enumerar os hiperprogramas de ordem 2 que param, H_{Ω_3} roda os hiperprogramas de U_2 no lugar dos programas de U (que é o que H_Ω faz), os quais são basicamente, programas de U que podem consultar, agora, duas fitas extras quaisquer.

Esses argumentos valem para todo Ω_n com n finito, formando uma cadeia H_Ω, H_{Ω_2},... de programas[219] que são recursivamente construtíveis. Portanto, por um argumento de indução matemática, esse processo de "chamada" e decisão também se torna um processo recursivo para qualquer n em $P_H^{(n+1)}$. Isto é a mesma coisa que dizer que, sim, para qualquer n, $P_H^{(n)}$ é uma *bit string* diferente de $P_H^{(n+1)}$. Porém, existe um programa que sabe qual é essa diferença.

Uma vez resolvida a questão de quais hiperprogramas $h^{(n)}$ de ordem n param ou não por um hiperprograma de ordem $n+1$ recursivamente construtível a partir de n, basta outro hiperprograma de ordem $n+1$ também recursivamente construtível a partir de n que pega esses $h^{(n)}$´s que se detêm e os roda em U_n. E, depois, seleciona o maior valor.

Portanto, dado n, existe um programa que retorna a *bit string* $P_H^{(n+1)}$. É este que atesta que $P_H^{(n+1)}$ é recursivamente construtível a partir de n.

3.9.4 **Lema:**

Seja $P_T^{(n+1)}$ o hiperprograma de ordem $n+1$ que recebe N como input e enumera todos os hiperprogramas de S_{HP} com tamanho $\leq N$ e ordem $\leq n$ usando o programa $P_{nN}^{(0)}$. Então, informa esses hiperprogramas como input para $P_H^{(n+1)}$ e retorna o valor desse último cômputo. Ou seja, $P_T^{(n+1)}$ retorna $U_{n+1}\left(P_H^{(n+1)} \circ (U\left(P_{nN}^{(0)} \circ n \circ N\right))\right)$.

- Nota:

---

[218] E o mesmo vale para H_{Ω_2}.
[219] Pois, estão "despidos" de seus números oraculares.

A existência desse programa depende somente de $P_H^{(n+1)}$ e $P_{nN}^{(0)}$. É importante lembrar que o output de $U\left(P_{nN}^{(0)} \circ n \circ N\right)$ já é colocado na forma que $P_H^{(n+1)}$ recebe seus inputs.

É natural perguntar agora como o programa $P_T^{(n+1)}$ se relaciona com $BB_\omega(N)$. Na verdade, ele está aqui justamente para cumprir o papel de calcular $BB_\omega(N)$. Porém, isso só acontecerá se n for suficientemente grande, pois precisamos que n seja maior ou igual a maior ordem presente entre os hiperprogramas de tamanho $\leq N$. Para isso, vamos usar o programa $P_{MO}^{(0)}$.

3.9.5 **Lema:**

$P_T^{(n+1)}$ é construtível recursivamente a partir de n.

- Demonstração:

O programa $P_{nN}^{(0)}$ é recursivamente construtível de forma trivial, pois é um programa. É um procedimento também recursivo o programa $P_T^{(n+1)}$ poder ler a si mesmo e a sua própria ordem para formar a *bit string* $P_{nN}^{(0)} \circ n \circ N$.

Dar o output de $U\left(P_{nN}^{(0)} \circ n \circ N\right)$ também é recursivo.

Lembre, pelo lema 3.9.3, que $P_H^{(n+1)}$ é também recursivamente construtível a partir de n.

Logo, o programa que monta o programa que recebe N como input e monta $P_H^{(n+1)} \circ (U\left(P_{nN}^{(0)} \circ n \circ N\right))$ também será recursivamente construtível a partir de n.

Temos que um hiperprograma h' que roda outro hiperprograma h de mesma ordem tal que este é recursivamente construtível também é recursivamente construtível, pois, ele só precisa executar as mesmas tarefas que outra *bit string* na fita principal de U_{n+1} executa. Como as sub-rotinas de U_{n+1} são bem conhecidas – em particular, trata-se de uma máquina universal

de Turing com uma fita principal e outras n fitas extras –, pode-se programar um algoritmo que emule U_{n+1} rodando h.[220] Note, também, que nenhum oráculo – ou nenhuma fita extra – é consultado até que $P_T^{(n+1)}$ comece a rodar $U_{n+1}\left(P_H^{(n+1)} \circ (U\left(P_{nN}^{(0)} \circ n \circ N\right))\right)$, i.e., até que comece a rodar esse h'.

Portanto, um hiperprograma que recebe N como input e depois roda $U_{n+1}\left(P_H^{(n+1)} \circ (U\left(P_{nN}^{(0)} \circ n \circ N\right))\right)$ é recursivamente construtível a partir de n.

3.9.6 **Teorema:**

Existe um hiperprograma $P_T^{\left(U_0\left(P_{MO}^{(0)} \circ N\right)+1\right)}$ que recebe N como input tal que $U_{U_0\left(P_{MO}^{(0)} \circ N\right)+1}\left(P_T^{\left(U_0\left(P_{MO}^{(0)} \circ N\right)+1\right)} \circ N\right) \geq BB_\omega(N)$.

– Prova:

Basta notar que todos os hiperprogramas de S_{HP} de tamanho $\leq N$ estarão no output de $U\left(P_{nN}^{(0)} \circ U_0\left(P_{MO}^{(0)} \circ N\right) \circ N\right)$. Logo, pela definição do hiperprograma $P_H^{(n+1)}$, teremos que o valor final retornado por $P_T^{\left(U_0\left(P_{MO}^{(0)} \circ N\right)+1\right)} \circ N$ vai ser obrigatoriamente o maior output dos hiperprogramas de S_{HP} com tamanho $\leq N$. ■

3.9.7 **Corolário:**

[220] Óbvio, caso esse algoritmo seja rodado por uma máquina oracular de mesma ordem.

$P_T^{\left(U_0\left(P_{MO}^{(0)} \circ N\right)+1\right)} \circ N$ é recursivamente construtível a partir de N.

- Prova:

Pelo lema 3.9.5, temos que $P_T^{(n+1)}$ é recursivamente construtível a partir de n.
Como o valor $U_0\left(P_{MO}^{(0)} \circ N\right)$ é também recursivamente construtível a partir de N, então a composição $P_T^{\left(U_0\left(P_{MO}^{(0)} \circ N\right)+1\right)} \circ N$ também será. ■

3.9.8 **Lema:**

Existe um programa $P_{P_T}^{(0)}$ que recebe N como input e retorna o hiperprograma $P_T^{\left(U_0\left(P_{MO}^{(0)} \circ N\right)+1\right)} \circ N$ como output.

- Prova:

Como, pelo corolário 3.9.7, $P_T^{\left(U_0\left(P_{MO}^{(0)} \circ N\right)+1\right)} \circ N$ é recursivamente construtível a partir de N, podemos tomar $P_{P_T}^{(0)}$ como sendo justamente esse programa que constrói $P_T^{\left(U_0\left(P_{MO}^{(0)} \circ N\right)+1\right)}$ a partir de N, e depois o concatena com a representação de N. ■

3.9.9 **Corolário:**

Existe um programa $P' \circ P_{P_T}^{(0)} \circ N$ que ignora seu input e que retorna $U_0\left(P_{P_T}^{(0)} \circ N\right)$. Além disso, $U_{U_0\left(P_{MO}^{(0)} \circ N\right)+1}\left(U_0\left(P_{P_T}^{(0)} \circ N\right)\right) \geq BB_\omega(N)$.

- Notas:

Esse resultado final se baseia no fato de $P_T^{\left(U_0\left(P_{MO}^{(0)} \circ N\right)+1\right)}$ ser recursivamente construtível a partir de N para sempre garantir que, onde quer que $P' \circ P_{P_T}^{(0)} \circ N$ ocorra como mutação, o próximo organismo terá aptidão $\geq BB_\omega(N)$. Vide o teorema 3.9.6. E isso acontece mesmo $P' \circ P_{P_T}^{(0)} \circ N$ sendo um hiperprograma de ordem zero, um programa comum. A ideia é que um programa pode nos dizer "alguém" que pode resolver um problema incomputável, apesar de não ser capaz de resolver esse problema. É um problema de nomeação de máquinas oraculares de Turing numa "roupagem" da teoria da informação algorítmica.

CONCLUSÃO

Para concluir, vamos primeiro resumir o que foi obtido em cada capítulo. Os capítulos 1 e 2 tratam do mesmo problema e abarcam um modelo de evolução diferente do terceiro. Portanto, pode-se pensar que fomos a duas direções "opostas" nesta tese. Antes, porém, vale lembrar que os modelos de evolução chaitinianos de **busca exaustiva**, **projeto inteligente** e **evolução cumulativa** se baseiam em assumir os organismos e mutações como programas ou como sistemas computáveis. A natureza é também um sistema, porém, incomputável: um hipercomputador de primeira ordem. Assim, nosso trabalho tratou de estudar esses modelos de evolução em configurações diferentes: primeiro, consideramos a natureza como um sistema computável, quando nossos organismos e mutações passaram a ser subprogramas; depois, consideramos os organismos como sistemas incomputáveis, como hiperprogramas de ordem finita, em que a natureza passou a um ser hiperprograma de ordem infinita.

Nos capítulos 1 e 2 começamos definindo o que é **subcomputação** e a **incomputabilidade relativa recursiva**. Como o próprio nome já indica, um subcomputador ou uma submáquina de Turing é um subsistema de um computador ou máquina universal de Turing, de forma que este último sempre sabe tudo que seu subsistema pode fazer, justamente, porque o "contém". **Subprogramas** são programas rodados por subcomputadores. Daí, de forma totalmente análoga à incomputabilidade clássica na literatura da teoria da computação, surge a incomputabilidade relativa entre o que pode ser computado por uma máquina universal de Turing e, ao mesmo tempo, não pode ser computado por qualquer submáquina de Turing.

Usamos esse novo fenômeno de incomputabilidade para substituir a incomputabilidade clássica da **função de aptidão Busy-Beaver**, que é crucial para os teoremas apresentados por Chaitin na metabiologia. Construímos uma versão relativamente incomputável em relação a $U_{P^{**}{}_{T^{\circ}P_T}}$ dessa função, denotada por $BB^{+}{}_{P^{**}{}_{T^{\circ}P_T}}(N)$. Para o output de qualquer subprograma superar ou igualar o valor $BB^{+}{}_{P^{**}{}_{T^{\circ}P_T}}(N)$, ele precisa ter necessariamente tamanho maior que N. Portanto, **atingir uma aptidão** $BB^{+}{}_{P^{**}{}_{T^{\circ}P_T}}(N)$ ainda quer dizer atingir N bits de criatividade ou complexidade, os quais são imcompressíveis por qualquer subprograma de tamanho menor. A submáquina de Turing $U_{P^{**}{}_{T^{\circ}P_T}}$ também foi construída para conseguir "carregar" (isto é, rodar certos programas) os análogos dos

programas cruciais que fazem o papel de organismos ou de mutação nas demonstrações chaitinianas dos modelos **projeto inteligente** e **busca exaustiva**.

Por isso, nos baseando no conceito de complexidade ou *program-size complexity* da teoria da informação algorítmica, só que aplicado à subcomputação, pudemos construir dois teoremas, isomorfos aos do **projeto inteligente** e **busca exaustiva** originais, respectivamente. Ou seja, os **tempos de mutação** (a quantidade média de mutações necessárias) para os organismos/subprogramas atingirem N bits de criatividade ou complexidade são, respectivamente, limitados superiormente por: N para o projeto inteligente, ou seja, linear; $N^2(\log_2 N)^{1+\epsilon}$, onde ϵ é uma constante, para a evolução cumulativa. Os quais são os mesmos resultados originais mostrados por Chaitin em seus teoremas.

Como tanto os nossos como os resultados originais tratam de limitantes superiores para o tempo de mutação, não podemos afirmar que o tempo de mutação com natureza/hipercomputador de primeira ordem e com natureza/computador são os mesmos. Como foi dito no capítulo 1, **somente algumas propriedades** que aparecem entre um hipercomputador de primeira ordem e um computador foram totalmente relativizadas para aparecerem também entre um computador e nossos subcomputadores. Justamente, foram aquelas necessárias para fazermos nossos teoremas *à la* Chaitin. Talvez, a forma melhor para se apresentar a incomputabilidade recursiva relativa teria sido, primeiro, definir, de forma geral – se é que é possível relativizar todas as propriedades entre dois graus de Turing -, um **grau negativo de Turing** na subcomputação. E só então construir uma natureza e organismos baseando-se nesse arcabouço teórico já dado. No entanto, pulamos essa definição mais geral e tentamos mostrar que pouco se precisa para conseguir formalizar uma evolução metabiológica computável "como se fosse incomputável".

De qualquer forma, nosso resultado da evolução cumulativa de subprogramas também serve como algo totalmente a favor da possibilidade de evolução de seres vivos, i.e., de evolução de sistemas computáveis, mesmo se a Natureza for computável. Essa evolução também ocorre movida apenas por mutações algorítmicas, as quais são aleatórias, e, mesmo assim, os organismos/subprogramas conseguem atingir qualquer nível de complexidade (irreduzível para os anteriores[221]) em um tempo médio bastante razoável (no máximo, em tempo praticamente quadrático).

---

[221] Para os subprogramas menores.

Já no capítulo 3, tomamos outra direção. Deixamos o "reino das coisas computáveis". Permitindo que os **seres metavivos**, nossos organismos matemáticos, sejam sistemas que não podem ser emulados por nenhuma máquina universal de Turing, provamos que é possível eles irem evoluindo só por meio de mutações algorítmicas aleatórias de modo a aumentarem de complexidade e, atrelado a isso, poderem resolver mais e mais problemas na **hierarquia aritmética**. Pudemos fazer essa ligação com a hierarquia aritmética por meio do teorema já conhecido que faz a correspondência desta com os **graus de Turing**. Por exemplo, para uma sentença qualquer na linguagem da aritmética usual que esteja na ordem n da hierarquia, existe um grau de Turing correspondente, em particular também n, tal que qualquer máquina oracular de Turing com acesso a um oráculo de ordem n pode decidir sempre se essa sentença é verdadeira ou falsa.

Começamos definindo nossos **hiperprogramas:** programas autodelimitados de máquinas oraculares de Turing. E os definimos de forma que eles são sempre *bit strings* finitas que podem ser rodados por uma máquina de Turing com acesso a números oraculares. Cada hiperprograma tem uma ordem correspondente a um **grau** na **hierarquia de Turing**. E eles precisam dizer em suas próprias *bit strings* a qual ordem pertencem (são **auto-ordenados**). Os organismos/hiperprogramas podem ter qualquer ordem finita. Daí, termos seres metavivos abrangendo toda a hierarquia aritmética: para qualquer problema aritmético, existe um organismo/hiperprograma que pode resolvê-lo. Em consequência disso, a natureza, por precisar poder saber sempre se qualquer organismo produz output ou não, passou a ser um hiperprograma de ordem maior que qualquer ordem finita, i.e., de ordem ω, o primeiro ordinal limite. Essa natureza/hiperprograma tem acesso a qualquer oráculo de ordem finita, podendo assim resolver qualquer problema aritmético.

Assim como na evolução de subprogramas, também construímos uma versão análoga à função de aptidão Busy-Beaver, no caso, a função $BB_{\omega}(N)$. Em resumo, ela é a função Busy-Beaver, só que calcula o maior output obtido por qualquer **hiperprograma** de ordem finita com tamanho $\leq N$. Essa demonstração também se baseia no fenômeno da incomputabilidade, só que no da **incomputabilidade relativa oracular** (a que ocorre entre diferentes graus de Turing). O modelos chaitinianos se baseiam na **incomputabilidade clássica** e a evolução de subprogramas, como mostramos, na relativa recursiva. A incomputabilidade desse tipo de função é, de fato, o que nos permite "medir" a complexidade ou criatividade dos organismos, porém, talvez, isso também nos indique que a incomputabilidade tem um papel central para a evolução de sistemas.

Nosso intuito também foi o de fazer uma demonstração parecida às da metabiologia inicial. Fizemos uma versão do modelo de **busca exaustiva**. Porém, chegamos a um resultado final diferente para o tempo de mutação. Assim, provamos que existe um programa que recebe N como input e que retorna como output a *bit string* correspondente a um hiperprograma. Este possui ordem e, por isso, complexidade suficientemente grandes para retornar como output um valor superior ou igual a $BB_{\omega}(N)$. Ou seja, o tempo de mutação para se atingir N bits de **criatividade aritmética** (uma complexidade aritmeticamente irreduzível) é $< C'N(\log_2 N)^{1+\epsilon}$ – também um limitante superior, que é muito menor se compararmos com a busca exaustiva chaitiniana e ainda também menor que sua evolução cumulativa. Em relação ao tempo de mutação, o projeto inteligente chaitiniano e a busca exaustiva de hiperorganismos são praticamente os mesmos. O que pode ser um resultado instigante.

É, em essência, um problema de nomeação recursiva de máquinas oraculares de Turing a partir da hierarquia aritmética. Porém, numa "roupagem" da teoria da informação algorítmica e usando conceitos da metabiologia. Portanto, a ideia básica desse resultado é poder estimar a probabilidade de surgir aleatoriamente um programa que faz essa nomeação.

Mesmo com mutações algorítmicas aleatórias "cegas", que descartam completamente seu input (ou seja, o organismo anterior), a evolução de hiperprogramas ocorre muito rápido. Dada qualquer sentença aritmética, não precisaríamos esperar um tempo grande para que a evolução por mutações algorítmicas aleatórias levassem os organismos/hiperprogramas a poder resolvê-la. A evolução "cega" de hiperprogramas em direção à **completude da aritmética** ocorre em tempo quase linear.

Com os dois resultados dos nossos três capítulos, podemos defender que supor que os seres vivos são sistemas que não podem ser emulados por computadores ou supor que a Natureza é emulável por um computador – que são suposições incompatíveis entre si e com a metabiologia original –, não servem para criticar a metabiologia como inadequada para a teoria da evolução da vida. Obtemos teoremas utilizando os fundamentos da metabiologia que mostram que os organismos evoluem mesmo que a Natureza seja computável ou, diametralmente opondo, mesmo que os organismos não sejam computáveis. O que nos coloca na direção de entender a metabiologia como uma teoria matemática para evolução de qualquer **sistema** que caia sob nossa definição geral e ubíqua de sistema e em nossa definição de aptidão - seja ele computável, subcomputável ou hipercomputável. Esse foi nosso objetivo principal.

As contribuições desses resultados em si já nos dão novos campos de investigação matemática. A primeira e, talvez, mais importante, questão, é estimar os tempos de mutação

médios[222] em cada um dos três casos (os chaitinianos e o dois desta tese) e não somente um limitante superior.

Poderíamos, assim, comparar entre *evolução cumulativa*, *projeto inteligente* e *busca exaustiva*, o que já seria auspicioso. Sobretudo, para a discussão quanto se a evolução da vida é **direcionada** (se a natureza ou "Deus" escolhe as mutações) ou **aleatória** (se é feita por mutações que ocorrem aleatoriamente). Saber os tempos de mutações médios nos permitiria averiguar qual é mais rápido ou se podem ser aproximadamente iguais: nos indicaria que a evolução direcionada é diferente da aleatória ou nos indicaria que elas são indiferenciáveis, respectivamente – nesse último caso, que "Deus" é contingente em relação à velocidade da evolução.

Além disso, poderíamos comparar esses tempos de mutações médios com os outros nos quais os organismos são subcomputáveis, computáveis ou hipercomputáveis. Dessa forma, dando-nos, pelo menos, pistas para diferenciar qual dos três casos se encaixa melhor para um modelo da biologia. Por exemplo, já mencionamos isso quando falamos, aqui mesmo na Conclusão, do tempo de mutação da busca exaustiva de hiperprogramas em comparação com os três modelos chaitinianos.

Indo além da parte matemática da tese, também fizemos várias discussões que subjazem, permeiam e se desdobram dos resultados obtidos. Tivemos que nos abster de perscrutar mais a fundo os conceitos e termos, envolvidos nelas, para satisfazer nosso desejo que mostrar a relação dos mesmos com as teorias e teoremas por nós desenvolvidos. Essas problemáticas, de qualquer forma, defendem, como nosso objetivo secundário, a relevância dos temas abordados na tese em relação a outros campos da ciência ou filosofia.

Por exemplo, no capítulo 1, fomos inspirados a usar o fenômeno da incomputabilidade relativa recursiva para solucionar nosso problema da evolução sob uma natureza computável. Para recriar a **incomputabilidade clássica** da função de aptidão Busy-Beaver, necessária para medir o ganho de criatividade ou complexidade (incompressível) dos seres metavivos ao longo da evolução, tivemos, basicamente, que fazer com Turing, Radó e Chaitin a mesma coisa que Skolem fez com Cantor: relativizar a incomputabilidade da função de aptidão atrelada a Ω. Mostramos que depende de estar olhando "de fora" ou "de dentro". Para tal, construímos um subcomputador definido sobre um programa que "se **autodiagonaliza**".

---

[222] Provavelmente, estimar limitantes inferiores também ajudaria.

Assim, analogamente ao "paradoxo" de Skolem, mostramos que existe o **"paradoxo" da computabilidade**: "de dentro" uma função pode ser incomputável, mas "de fora" ela pode ser computável. Como o de Skolem, essa antinomia linguística, na verdade, não constitui um paradoxo verdadeiro, por isso, as aspas. Além disso, é interessante notar que o fato de não haver contradição vem da constatação matemática de um fenômeno linguístico, o qual permite que algo seja dito na "língua" e o contrário seja dito na metalíngua. Portanto, chamamos todos os nossos "paradoxos" de **pseudoparadoxos metalinguísticos**. Eles derivam de tomar como hipótese subsistemas como subcomputadores de outro sistema. Dessa forma, aparece o fenômeno da incomputabilidade relativa entre o sistema e o subsistema. Existiriam problemas que o sistema pode resolver e que o subsistema não pode.

Esse caminho nos levou, em seguida, ao **"paradoxo" da intuição matemática**. Ele trata de explicar por que, mesmo que a mente, no final, seja computável, os matemáticos teriam a sensação ou convicção [223] de que sempre podem criar ou decidir sentenças indecidíveis por qualquer teoria axiomática formal, as quais eles mesmos tenham criado. Em resumo, discutimos acerca da pergunta: **o matemático é completo?** Sob as devidas hipóteses, portanto, para nós – matemáticos e conscientes –, pareceria que nós mesmos (nossa mente) somos incomputáveis, mesmo que não realmente sendo (caso algo ou alguém possa nos olhar "de fora"). Seríamos programas destinados (ou programados) a nunca achar a nossa "computabilidade". Apesar de existir um sistema axiomático que faz tudo que o matemático possa fazer, ele poderia **nos** ser inalcançável.

Também fizemos provocações sobre a subcomputação ser uma das propriedades constitutivas do fenômeno da consciência humana, podendo, inclusive, ser responsável por fazer o compatibilismo entre **computabilidade da mente** e **livre-arbítrio**.

O próximo **"paradoxo"** foi o da **computabilidade biológica**. Seguindo o raciocínio análogo ao de antes, ele levantaria como possibilidade a biologia ou os sistemas vivos parecerem incomputáveis para nós (tomando a consciência como um subsistema do organismo), mesmo que, "de fora", não sejam de fato. Então, relacionamos esse fenômeno hipotético com a busca de uma biologia teórica completa e com **vida artificial.**

Estendendo mais ainda o que fizemos para a computabilidade biológica, argumentamos também sobre o **"paradoxo" da computabilidade do Universo.**

[223] Pois, quando estudamos lógica-matemática e os teoremas de incompletude ficamos, no mínimo, tentados a acreditar nisso.

Analogamente ao da biologia, as leis[224] da física poderiam se apresentar a nós (tomando "nós" como um subsistema do Universo ou do sistema regido por todas as leis da física) como incomputáveis, mesmo que elas sejam, na verdade, computáveis. Relacionamos isso a uma disjunção exclusiva de cunho epistemológico: teríamos que escolher entre conhecer todas as leis da física **ou** poder usá-las para prever (ou explicar) os fenômenos. Nunca as duas coisas ao mesmo tempo. Isso poderia ser caracterizado como uma "**lei de ignorância**" ou uma espécie de **"princípio de incerteza"** epistemológico.

No capítulo 3, tratamos da possibilidade de a vida ser verdadeiramente incomputável, em que os seres vivos poderiam ser sistemas não emuláveis por qualquer computador. Apresentamos alguns argumentos contra a **hipótese da vida computável** (a hipótese de que os seres vivos seriam sistemas computáveis), relacionando com as discussões do capítulo 1. Inclusive, mesmo a evolução de hiperprogramas indo numa direção oposta à da evolução de subprogramas (estão em hierarquias totalmente diferentes), levantamos a possibilidade de os dois resultados matemáticos estarem ligados pela **hipercomputação relativa recursiva** – que é um campo ainda a ser investigado na busca de ampliar a incomputabilidade relativa, já obtida nesta tese, para a criação de uma **hierarquia negativa de graus de Turing**.

Outro campo de investigação matemática proposto foi averiguar se é possível construir uma evolução de hiperprogramas não só em direção à completude da aritmética, mas também em direção a "computar" todos os números reais ou a "computar" todos os conjuntos. Enfim, estudar a evolução de sistemas que ultrapassam o quanto se queira a "barreira" do computável.

Para finalizar, todas essas provocações constituem nosso objetivo secundário. Em decorrência dos conceitos novos – ou atualizados – elaborados para resolver o problema da evolução de subprogramas e o da evolução de hiperprogramas, mostramos que as consequências científicas e filosóficas dos nossos resultados matemáticos se alastram pelas questões fundamentais da biologia e vida artificial, como também, da física e da inteligência artificial. Tudo o que foi discutido floresce em nossos anseios como contribuições e/ou provocações para o futuro.

---

[224] Mesmo as que não conhecemos ou as que nunca conheceremos.

REFERÊNCIAS

ABDALLA, M. La Crisis Latente del Darwinismo. **Asclepio. Revista de Historia de la Medicina y de la Ciencia**, LVIII, 2006. 43-94.

ABDALLA, M. **La Crisis Latente del Darwinismo**. Sevilla: Publidisa, S.A., 2010. ISBN ISBN 978-84-937871-1-0.

ABRAHÃO, F. S. **Demonstrando a Consistência da Aritmética**. Dissertação de mestrado. ed. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2011.

ABRAHÃO, F. S. Questões em Metabiologia. **Scientiarum Historia**, Rio de Janeiro, 2011.

ABRAHÃO, F. S. Metabiologia Cantoriana. **Scientiarum Historia**, Rio de Janeiro, 2013.

ABRAHÃO, F. S. Teologia da Informação. **Scientiarum Historia** , 2013.

ABRAHÃO, F. S. "Paradoxo" da Computabilidade. **Scientiarum Historia**, Rio de Janeiro, 2014.

ADAMI, C. Information-Theoretic Considerations Concerning the. **ARXIV**, 2014.

ALTER, T.; HOWELL, R. **A dialogue on consciousness**. New York: Oxford University Press, Inc., 2009. ISBN ISBN 978-0-19-537529-9.

ARKOUDAS, K. Computation, hypercomputation, and physical science. **Journal of Applied Logic**, 2008. 461-475.

BANATHY, B. **Guided Evolution of Society:** A Systems View. [S.l.]: Springer US, 2000. ISBN ISBN 978-1-4419-3342-3.

BARWISE, J. (Ed.). **Handbook of Mathematical Logic**. Amsterdam: Elsivier Science Publisher B.V., 1977. ISBN 0444863885.

BASU, S. On the Structure of Subrecursive Degrees. **Journal of Computer and System Sciences**, 1970.

BEKLEMISHEV, L. D. Reflection principles and provability. **Russian Mathematical Surveys**, v. 60:2, p. 197–268, 11 Janeiro 2005.

BERLINSKI, D. **The Atheism and its Devil´s Scientific Pretensions Delusion**. [S.l.]: Basic Books, 2009. ISBN ISBN 978-0-465-01937-3.

BERTALANFFY, L. V. **General System Theory:** Foundations, Development, Applications. New York: George Braziller, 1968. ISBN ISBN 0-8076-0453-4.

BOOLOS, G.; BURGESS, J.; JEFFREY, R. **Computability and Logic**. New York: Cambridge University Press, 2007. ISBN ISBN 978-0-521-87752-7.

BOVYKIN, A. **On order-types of models of arithmetic**. Tese de Doutorado. ed. Birmingham: The University of Birmingham, 2000.

CALUDE, C. (Ed.). **Randomness and Complexity**. Singapore: World Scientific Publishing Co. Ptc. Ltd., 2007.

CAPRA, F. **A teia da vida**. 1ª. ed. [S.l.]: Pensamento-Cultrix LTDA, 1997. ISBN ISBN 978-85-316-0556-7.

CHAITIN, G. **Epistemology as Information Theory: From Leibniz to Omega**. Alan Turing Lecture on the European Computing and Philosophy Conference at Mälardalen University. Västerás. 2005.

CHAITIN, G. **Meta Math!:** the quest for omega. [S.l.]: Vintage Books, 2005. ISBN 978-1-4000-7797-7.

CHAITIN, G. **The Halting Probability Omega: Irreducible Complexity in Pure Mathematics**. University of Milan. [S.l.]. 2006.

CHAITIN, G. Metaphysics, Metamathematics and Metabiology. In: ZENIL, H. **Randomness through Computation**. [S.l.]: World Scientific, 2011. p. 93-103.

CHAITIN, G. Life as Evolving Software. In: ZENIL, H. **A Computable Universe:** Understanding and Exploring Nature as Computation. [S.l.]: [s.n.], 2012. p. 277-302. ISBN 978-9814374293.

CHAITIN, G. **Proving Darwin:** making biology mathematical. [S.l.]: Vintage Books, 2012. ISBN 978-1-4000-7798-4.

CHAITIN, G. J. **Exploring Randomness**. Londres: Springer-Verlag London Limited, 2001. ISBN 1-85233-417-7.

CHAITIN, G. J. **Algorithmic Information Theory**. 3ª. ed. Yorktown Heights: IBM, P O Box 218, 2003.

CHAITIN, G. J. To a mathematical theory of evolution and biological creativity. **Randomness, Structure and Causality: Measures of Complexity from Theory to Applications**, Santa Fe, January 2011.

CHAITIN, G. J. **Conceptual Complexity and Algorithmic Information**. Federal University of Rio de Janeiro. [S.l.]. 2014.

CHAITIN, G. J.; CHAITIN, V. M. F. G.; ABRAHÃO, F. S. Metabiología: los orígenes de la creatividad biológica. **Investigación y Ciencia**, n. 448, Janeiro 2014.

CHAITIN, G.; CHAITIN, V.; KUBRUSLY, R. É possível matematizar a biologia?. **Revista Carbono**, Rio de Janeiro, n. 02. ISSN ISSN 2358-8047.

CHAITIN, G.; DA COSTA, N.; DORIA, F. A. **Gödel´s Way:** Exploits into an undecidable world. Londres: Taylor & Francis Group, 2012. ISBN ISBN 978-0-415-69085-0.

CHAITIN, V. Criatividade, aleatoriedade e complexidade: a matemática na vida. **Scientiarum Historia III**, Rio de Janeiro, 2010.

CHAITIN, V. O corpo metabiológico. **Scientiarum Historia IV**, Rio de Janeiro, 2011.

CHAITIN, V. Metafísica, metamatemática e metabiologia. **Revista Tempo Brasileiro**, n. nº 189/190, p. 101-114, 2012.

CHAUVIN, R. **O Darwinismo ou o fim do Mundo**. Lisboa: Instituto Piaget, 1997. ISBN ISBN 972-771-155-3.

CHURCH, G.; REGIS, E. **Regenesis:** how synthetis biology will reinvent nature and ourselves. New York: Basic Books, 2012. ISBN ISBN 978-0-465-02175-8.

COOPER, B. Definability as hypercomputational effect. **Applied Mathematics and Computation**, n. 178, 2006. 72-82.

COPELAND, J. Hypercomputation. **Minds ad Machines**, 2002. 461-502.

COPELAND, J. The Church-Turing Thesis. **The Stanford Encyclopedia of Philosophy**, 2008. Disponivel em: <http://plato.stanford.edu/archives/fall2008/entries/church-turing/>. Acesso em: 23 Março 2015.

DA COSTA, N. C. A.; DORIA, F. A. Some thoughts on hypercomputation. **Applied Mathematics and Computation 178**, 2006.

DAVIES, P. The epigenome and top-down causation. **Interface focus**, 2011.

DAY, T. Computability, Gödel´s incompleteness theorem, and a inherent limit on the predictability of evolution. **J. R. Soc. Interface**, Agosto 2011.

DENNETT, D. **Darwin´s dangerous idea:** evolution and the meanings of life. New York: Simon & Schuster Paperbacks, 1995.

DENNETT, D. **Freedom Evolves**. [S.l.]: Viking Penguin, 2003. ISBN ISBN 0-670-03186-0.

DÓRIA, F. A. **An introduction to the Shannon coding theorem**. [S.l.]. 2007.

DORIA, F. A.; CARNIELLI, W. A. Are the Foundations of Computer Science Logic–Dependent? **Dialogues, Logics and Other Strange Things**, 20 Outubro 2008.

DOVAL, D. H. **Redes, Sistemas y Evolución:** Hacia una nueva biología. Madrid: Universidad Autónoma de Madrid, 2013.

ENDERTON, H. **A mathematical introduction to logic**. 2ª. ed. Boston, MA: Academic Press, 2001. ISBN 978-0-12-238452-3.

ETESI, G.; NÉMETI, I. Non-Turing Computations Via Malament-Hogarth Space-Times. **International Journal of Theoretical Physics**, n. 41, 2002.

EWERT, W.; DEMBSKI, W.; MARKS II, R. J. Active Information in Metabiology. **Bio-Complexity**, Dezembro 2013.

FERNANDES, A. M. D. R. **Inteligência Aritificial:** noções gerais. Florianópolis: Visualbooks Editora, 2008. ISBN ISBN 85-7502-114-1.

FODOR, J.; PIATTELLI-PALMARINI, M. **What Darwin got Wrong**. New York: Picador, 2010. ISBN ISBN 978-0-374-28879-2.

FOUKS, J.-D. Towards an algorithmic theory of adaptation. **Theoretical Computer Science**, 1999. 121-142.

FRANZÉN, T. Transfinite Progressions: a second look at completness. **The Bulletin of Symbolic Logic**, v. 10:2, Setembro 2004.

GOULD, S. J. **Wonderful Life:** The burgess Shale and the Nature of History. New York: W. W. Norton & Company, Inc., 1989.

GRÜNWALD, P.; VITÁNYI, P. **Shannon Information and Kolmogorov Complexity**. [S.l.]. 2010.

HORST, S. The Computational Theory of Mind. **The Stanford Encyclopedia of Philosophy**, 2011. Disponivel em: <http://plato.stanford.edu/archives/spr2011/entries/computational-mind/>. Acesso em: 23 Março 2015.

HORSTEN, L. Philosophy of Mathematics. **The Stanford Encyclopedia of Philosophy**, 29 Agosto 2008. Disponivel em: <http://plato.stanford.edu/archives/fall2008/entries/philosophy-mathematics/>. Acesso em: 17 fev. 2011.

HULL, D.; RUSE, M. (Eds.). **The Cambridge Companion to the Philosophy of Biology**. [S.l.]: Cambridge University Press, 2007. ISBN ISBN 978-0-521-61671-3.

HUNTER, C. Fred Sanger, Protein Sequences and Evolution Versus Science. **Darwin´s God**, 2013. Disponivel em: <http://darwins-god.blogspot.com.br/2013/11/fred-sanger-protein-sequences-and.html>. Acesso em: 17 Março 2015.

JOHNSON-LAIRD, P. N. **Mental Models**. 6. ed. Cambridge: Harvard University Press, 1995.

KAUFFMAN, S. **The Origins of Order:** Self-oraganization and selection in evolution. New York: Oxford University Press, Inc. , 1993.

KIM, J. **Philosophy of Mind**. 3ª. ed. Boulder: Westview Press, 2011.

KUNEN, L. A Ramsey Theorem in Boyer-Moore Logic. **Journal of Automated Reasoning**, 1995. 217-235.

LEISTCH, A.; SCHACHNER, G.; SVOZIL, K. How to Acknowledge Hypercomputation? In: _____ **Complex systems**. [S.l.]: Complex Systems Publications, Inc, 2008.

LEWIS, H. R.; PAPADIMITRIOU, C. H. **Elementos de Teoria da Computação**. 2ª. ed. Porto Alegre: Bookman, 2000.

LOGAN, R. **Que é informação?:** a propagação da informarção na biosfera, na simbolosfera, na tecnosfera e na econosfera. Rio de Janeiro: Contraponto Editora, 2012. ISBN ISBN 978-85-7866-046-8.

LÓPEZ, M. **Diseño Inteligente:** Hacia un nuevo Paradigma Científico. [S.l.]: Organización Internacional para el Avance Científica del Diseño Inteligente, 2010. ISBN ISBN 10-1451514972.

MACLENNAN, B. Natural computation and non-Turing models of computation. **Theoretical Computer Science**, n. 317, 2004. 115-145.

MARGOLIS, E.; SAMUELS, R.; STICH, S. (Eds.). **The Oxford Handbook of Philosophy of Congnitive Science**. New York: Oxford University Press, Inc., 2012.

MARGULIS, L.; PUNSET, E. **Mind, Life, and Universe**. [S.l.]: Chelsea Green Publishing Company, 2007. ISBN ISBN 978-1-933392-61-5.

MARGULIS, L.; SAGAN, D. **Acquiring genomes:** a theory of the origins of species. [S.l.]: Basic Books, 2002.

MATURANA, H.; VARELA, F. **Autopoiesis and Cognition**. Dordrechet: D. Reidel Publishing Company, 1972. ISBN ISBN 90-277-1015-5.

MATURANA, H.; VARELA, F. **A árvore do cohecimento:** as bases biológicas da compreensão humana. são Paulo: Palas Athena Editora, 1984. ISBN ISBN 978-85-7242-032-7.

MCKENNA, M.; COATES, J. Compatibilism. **The Stanford Encyclopedia of Philosophy**, 2015. Disponivel em: <http://plato.stanford.edu/archives/sum2015/entries/compatibilism/>. Acesso em: 14 Abril 2015.

MCLAUGHLIN, B.; BECKERMANN, A.; WALTER, S. (Eds.). **The Oxford Handbook of Philosophy of Mind**. New York: Oxford University Press Inc., 2009. ISBN ISBN 978-0-19-926261.

MENDELSON, E. **Introduction to Mathematical Logic**. 2ª. ed. Princeton, New Jersey: D. VAN NOSTRAND COMPANY, INC., 1964.

MEYER, S. **Signature in the cell:** Dna and evidence for Intelligent Design. New York: HarperCollins, 2009.

MITCHELL, M. **Complexity:** a guided tour. New York: Oxford University Press, Inc., 2009. ISBN ISBN 978-0-19-512441-5.

MONOD, J. **Chance and Necessity:** An Essay on the Natural Philosophy of Modern Biology. New York: Alfred A. Knopf, 1971. ISBN 0-394-46615-2.

MURAWSKI, R. On Proofs of the Consistency of Arithmetic. **STUDIES IN LOGIC, GRAMMAR AND RHETORIC 5**, 2002.

MURPHY, M.; O´NEILL, L. **"O que é vida?" 50 anos depois:** especulações sobre o futuro da biologia. [S.l.]: Fundação editora UNESP (FEU), 1997. ISBN ISBN 85-7139-168-8.

ORD, T. **Hypercomputation: computing more than the Turing machine**. Department of Computer Science, University of Melbourne. [S.l.].

PITT, D. Mental Representation. **The Stanford Encyclopedia of Philosophy**, 2008. Disponivel em: <http://plato.stanford.edu/archives/fall2008/entries/mental-representation/>. Acesso em: 6 Setembro 2011.

POTGIETER, P. Zeno machines and hypercomputation. **Theoretical Computer Science**, n. 358, 2006. 23-33.

PYLYSHYN, Z. Is vision continuous with cognition? The case for cognitive impenetrability of visual perception. **Behavioral and Brain Sciences**, n. 22, 1999. 341-423.

RIBEIRO, H. D. M. Da metamatemática para a ciência cognitiva. **Scielo**, 1999.

RIDLEY, M. **The red quenn:** sex and the evolution of human nature. Londres: Harper Perennial, 1993.

ROGERS, H. **Theory of recursive functions and effective computability**. 3ª. ed. [S.l.]: MIT Press, 1992. ISBN 0-262-68052-1.

ROSE, H. E. Subrecursion: Functions and Hierarchies. **The Journal of Symbolic Logic**, 52, Nº 2, Junho 1987.

RUSE, M. (Ed.). **The Oxford Handbook of Philosophy of Biology**. New York: Oxford University Press, Inc., 2008.

SACKS, O. **O homem que confundiu sua mulher com um chapéu**. São Paulo: Companhia das Letras, 1970. ISBN ISBN 978-85-7164-689-6.

SACKS, O. **O Olhar da Mente**. São Paulo: Editora Schwarcz LTDA, 2010. ISBN ISBN 978-85-359-1769-7.

SANDÍN, M. **La Transformación de la Evolución**. Real Sociedad Española de Historia Natural. [S.l.], p. 139-167. 2005.

SARKAR, S.; PLUTYNSKI, A. (Eds.). **A Companion to the Philosophy of Biology**. [S.l.]: Blackwell Publishing Ltd, 2008. ISBN ISBN 978-1-4051-2572-7.

SCHMIDHUBER, J. Gödel Machines: Self-Referential Universal Problem solvers Making Provably Optimal Self-Improvements. **arXiv**, Setembro 2003.

SHANNON, C. A Mathematical Theory of Communication. **The Bell System Technical Journal**, 1948.

SHAPIRO, S. (Ed.). **The Oxford Handbook of Philosophy of Mathematics and Logic**. New York: Oxford University Press, Inc. , 2005.

SIEGELMANN, H. Neural and Super-Turing Computing. **Minds and Machines**, n. 13, 2003. 103-114.

SIEGELMANN, H.; SONTAG, E. Analog computation via neural networks. **Theoretical Computer Science**, 131, 1993. 331-360.

SINGER, E. Under Pressure, Does Evolution Evolve? **Quanta Magazine**, Janurary 2014.

SMITH, J. M. **Mathematical Ideas in biology**. New York: Cambridge University Press, 1968.

SMITH, J. M. **Shaping Life:** genes, embryos, and evolution. [S.l.]: Weidenfeld & Nicolson, 1998. ISBN ISBN 0-300-08022-0.

SMITH, J. M.; SZATHMÁRY, E. **The origins of life**. New York: Oxford University Press, Inc., 1999.

SONG, D. Non-Computability of Consciousness. **NeuroQuantology**, 5, 2007. 382-391.

STANNETT, M. Computation and Hypercomputation. **Minds and Machines**, n. 13, 2003. 115-153.

SUBER, P. What is Software? **Journal of Speculative Philosophy**, 1988.

SZUDZIK, M. The Computable Universe Hypothesis. **arXiv**, Dezembro 2010.

TURING, A. Systems of logic based on ordinals. **Proceedings of the London Mathematical Society**, 1939. 161-228.

TURNER, R. The Philosophy of Computer Science. **Stanford Encyclopedia of Philosohy**, 2014. Disponivel em: <http://plato.stanford.edu/archives/win2014/entries/computer-science/>. Acesso em: 17 Março 2015.

WALKER, S. I.; DAVIES, P. the Algorithmic Origins of Life. **arXiv**, 2012.

WIKIPEDIA CONTRIBUTORS. Program-size complexity. **Wikipedia**, 2005. ISSN 29200492. Disponivel em: <http://en.wikipedia.org/w/index.php?title=Program-size_complexity&oldid=29200492>. Acesso em: 17 Março 2015.

WIKIPEDIA CONTRIBUTORS. Inductive reasoning. **Wikipedia, The Free Encyclopedia**, 2009. Disponivel em: <http://en.wikipedia.org/w/index.php?title=Inductive_reasoning&oldid=309075447>. Acesso em: 6 Setembro 2011.

WIKIPEDIA CONTRIBUTORS. Philosophy of mind. **Wikipedia, The Free Encyclopedia**, 2009. ISSN 308403498. Disponivel em: <http://en.wikipedia.org/w/index.php?title=Philosophy_of_mind&oldid=308403498>. Acesso em: 17 Fevereiro 2011.

WIKIPEDIA CONTRIBUTORS. Chaitin's constant. **Wikipedia, The Free Encyclopedia**, 2011. ISSN 413788003. Disponivel em: <http://en.wikipedia.org/w/index.php?title=Chaitin%27s_constant&oldid=413788003>. Acesso em: 17 Fevereiro 2011.

WIKIPEDIA CONTRIBUTORS. Deductive reasoning. **Wikipedia, The Free Encyclopedia**, 2011. Disponivel em: <http://en.wikipedia.org/w/index.php?title=Deductive_reasoning&oldid=448498019>. Acesso em: 6 Setembro 2011.

WIKIPEDIA CONTRIBUTORS. History of scientific method. **Wikipedia, The Free Encyclopedia**, 2011. Disponivel em: <http://en.wikipedia.org/w/index.php?title=History_of_scientific_method&oldid=446884020>. Acesso em: 6 Setembro 2011.

WIKIPEDIA CONTRIBUTORS. Hypercomputation. **Wikipedia**, 2015. Disponivel em: <http://en.wikipedia.org/w/index.php?title=Hypercomputation&oldid=598466518>. Acesso em: 17 Março 2015.

WIKIPEDIA CONTRIBUTORS. Monolith (Space Odyssey). **Wikipedia**, 2015. Disponivel em: <http://en.wikipedia.org/w/index.php?title=Monolith_(Space_Odyssey)&oldid=645736730>. Acesso em: 17 Março 2015.

WKIPEDIA CONTRIBUTORS. Turing degree. **Wikipedia**. ISSN 651758268. Disponivel em: <http://en.wikipedia.org/w/index.php?title=Turing_degree&oldid=651758268>. Acesso em: 17 Março 2015.

WOLCHOVER, N. A New Physics Theory of Life. **Quanta Magazine**, Januart 2014.

ZENIL, H. (Ed.). **Randomness through Computation:** some answers, more questions. Singapore: World Scientific Publishing Co. Ptc. Ltd., 2011.

ZENIL, H. (Ed.). **A Computable Universe**. Singapore: World Scientific Publishing Co. Pte. Ltd., 2013. ISBN ISBN 978-981-43-74-29-3.

ZENIL, H. Hypercomputation in A Computable Universe. **Anima Ex Machina**. Disponivel em: <http://www.mathrix.org/liquid/archives/hypercomputation-in-a-computable-universe>. Acesso em: 29 ago. 2013.

ZIMMER, C. The Surprising Origins of Life's Complexity. **Quanta Magazine**, Julho 2013.