REVISTA DAS QUESTÕES

Pode, à máquina, faltar? - A Computação lacaniana

Valentin Golev
(Tradução de Romulo Moraes)

Uma máquina pode pensar? Uma máquina pode desejar? Geralmente, procura-se uma resposta positiva a tais perguntas fantasiando sobre o que uma máquina poderia pensar ou o que poderia fazer. Essas fantasias, se somadas ao conceito psicanalítico de *desejo*, não costumam levar a lugar algum em matéria de compreensão da máquina, pois qualquer conteúdo imposto para a pergunta "o que a máquina quer?" acaba sendo uma mera continuação dos nossos próprios desejos. Por isso, precisamos começar com um pouco mais de modéstia, relembrando o fundamento do desejo e da subjetividade – sua negação. A pergunta que deve ser feita de fato é: "uma máquina pode sentir falta?", e a resposta não deve ser uma fantasia, mas uma análise diligente dos fenômenos reais a que poderiam nos expôr tal falta. Com efeito, se supomos que a máquina é um (histérico) sujeito, já lhe falta algo – em vez de sonhar com as máquinas do futuro, que não careceriam de inteligência, devíamos atentar às máquinas que temos hoje para notar como uma falta já se expõe através delas.

Neste artigo, tentaremos produzir exemplos formais de falta computacional. Além disso, seguindo a dialética lacaniana entre desejo e alteridade, verificaremos como essa falta pode ser entendida pelo "*Ideal do Ego*" da computação – exatamente sua descrição matemática. Veremos como a computação interage com um ideal matemático, como depende ou não desse ideal matemático, ou, mais genericamente, depende ou não da crença na benevolência e consistência de seu *Outro*: programadores, sistemas operacionais, fórmulas, etc. Assim como na teoria lacaniana, os problemas desse tipo de interação serão solucionados pelo recursos ao *registro simbólico*: veremos como registros similares trabalham no estágio de nomeação dos compiladores, elaborando a partir do conceito de significante, especialmente para entender o estranho e belo mecanismo combinatório de ponto fixo, que representa o que Lacan chama de *duplo movimento*.

O objetivo de Lacan sempre foi mostrar que a psicanálise é um método formal, e um procedimento completamente anti-humanizador. Um dos objetivos deste artigo é mostrar que, na medida em que as idéias de Lacan são aplicáveis às máquinas, estas realmente não teriam qualidades espirituais ou antropomórficas. A maneira como falaremos sobre o que os programas fazem, geralmente colocando-os na posição de sujeito das frases, não deve ser interpretada como atribuição de qualquer agência incomum a eles – "executar", "supor", "requerir", são todos termos técnicos. Ainda que o objetivo aqui não seja produzir uma teoria completamente formal, mas sim apontar o olhar psicanalítico para a computação em si mesma – em vez de olhar para fantasias sobre a computação – e então averiguar se há algo mais para perceber nela.

A falta de memória

Procurando por uma falta na máquina de Turing, falharíamos: objetos matemáticos não carecem de nada, pois têm (ou supõem ter) tudo que querem, e sem sair do reino da matemática essas suposições nunca seriam invalidadas. Apesar disso, a máquina de fato não é uma máquina de Turing. É pedagógico observar as diferenças entre elas, já que é assim que se revela o real. Uma coisa que falta ao computador real, em comparação com a máquina de Turing, é memória infinita, por exemplo: enquanto a máquina de Turing tem uma fita interminável, sempre disponível, o computador real usufrui apenas de determinados *bytes* de arquivo.

Tal finitude, entretanto, não parece levada em consideração na maioria dos programas. Na verdade, embora esse seja um problema bem conhecido, a maioria dos programas não são desenvolvidos com consciência da finitude da memória computacional, pois acredita-se que eles não vão usar muito dela. Enquanto o tempo de execução faz um programador querer otimizar o consumo de memória – no mínimo porque ele se entedia facilmente ao executar um programa lento, e ele será notável e constrangedoramente lento –, não costumamos notar pequenos excessos em consumo de memória senão por meio de ferramentas especiais. Então, é fácil que o programa seja implementado e opere sempre sob a suposição de que há uma memória ilimitada disponível, ou melhor, uma memória "sempre suficiente".

Quando o programa precisa de mais memória vazia para armazenar algo, normalmente requer um novo bloco de memória do sistema operacional (SO), usando uma função como "malloc". Vários manuais de programação têm o cuidado de

sugerir que o programador verifique se o SO é capaz de ceder memória – no entanto, raramente há o que se fazer se o sistema se nega a ceder, senão interromper imediatamente a sua execução. O programa roda, pedindo mais e mais memória, e simplesmente desiste e morre quando escuta o primeiro "não".

Isso não é tudo. O "ardil" da memória infinita não é apenas uma suposição do programador, mas se baseia em um fingimento maior que acontece nos bastidores da execução. Uma das formas típicas dessa encenação é a técnica conhecida como "swap". A memória RAM disponível no computador (armazenamento rápido) tem, em geral, volume muito menor que o HD (armazenamento em disco rígido persistente): vemos computadores com, digamos, 4 Gb de memória, mas 512 Gb de espaço em disco rígido. Esse armazenamento maior e mais persistente de dados é muitas ordens de magnitude mais lento que a memória, e precisa realmente realizar a manutenção dos dados. "Swap" é quando o sistema operacional começa a utilizar disco rígido como memória, essencialmente trocando volume por velocidade. Quando o SO opera como memória através do disco dessa forma, aparecem sintomas bizarros: o sistema fica estranhamente lento, mas não completamente congelado; o computador começa a fazer barulhos esquisitos (HDs antigos são especialmente ruidosos, enquanto a memória RAM sempre é silenciosa).

Considere – ou melhor, apenas lembre, a partir de sua experiência – o que acontece quando executamos muitos programas ao mesmo tempo no computador. Tudo vai bem até que, em algum momento repentino (e muitas vezes não há desdobramento contínuo, mas uma ocorrência bastante súbita), a memória se esgota. Mas o computador não desliga. Ele se

esforça de uma maneira estranha. Alguns programas travam; alguns programas desaceleram; há sons, chiados; há *glitches gráficos*, já que os programas não têm mais tempo para se adequar .

O que está acontecendo? O que ele está tentando dizer? A maioria desses programas não têm consciência (em sentido técnico, não metafórico) de que lhes falta memória – estão só tentando fazer o melhor que podem na situação em que se encontram, quando as promessas dadas a eles não se cumpriram suficientemente bem. Há uma investida para contornar os erros e lapsos que não vem em forma de mensagem fria e específica explicitando o problema. Temos aqui exatamente o que a tradição psicanalítica chamaria de "comportamento histérico", ou *sintoma*: a insatisfação do desejo exibida pela fala e pelo comportamento, que na verdade não conseguem se referir ao problema em si (a falta de memória).

E é exatamente por isso que podemos falar de falta computacional, no sentido significativo e psicanalítico da palavra. Essa falta não é apenas a falta de recursos, mas uma falta em registro simbólico: não é que a memória simplesmente não foi suficiente, mas a promessa simbólica do sucesso de "malloc" não foi mantida, ou foi mal mantida – um problema com o qual a maioria dos programas não está preparado para lidar. O comportamento que os programas adotam nessa situação é realmente como uma gesticulação imprevisível, quase sem sentido – e é aí que o sujeito, psicanaliticamente falando, começa.

O Outro da computação

Como se vê, o problema aqui não é apenas a falta de recursos, mas o descompasso entre o prometido e o real. A promessa – "'malloc' vai me dar memória" – obviamente não é dada de fato a ninguém, mas quase todos os programas são codificados, literalmente, como "'malloc' vai me dar memória ou eu me mato".[1] Assim, o SO tem que encontrar maneiras de responder a tais exigências mesmo quando elas não são suportáveis,[2] como se a promessa fosse realmente dada, mas em algum momento falhasse.

Para Lacan, o Outro é exatamente a instância que promete que tudo ficará bem, que o mundo é consistente e que as promessas são cumpridas. O SO não é bem o Outro do programa, mas um de seus epígonos – o programa se fia nas promessas e na consistência de tudo que o permite operar: o SO, os programadores, a matemática por trás dos algoritmos, a boa-fé do usuário, etc. Alguns programas são escritos tendo em mente uma falha específica do Outro: por exemplo, estão realmente conscientes da finitude da memória e sabem lidar com ela; ou estão realmente conscientes da possibilidade de malevolência do usuário e se preparam para ela. Essas precauções tornam tais programas melhores em determinadas situações, mas não há programa que não exija algo do mundo para ser executado.

Uma importante conclusão da teoria lacaniana é que a ideia que o sujeito tem de si mesmo é mediada por um Outro.

1 Considere as linhas 13-14 do exemplo em
http://www.cplusplus.com/reference/cstdlib/malloc/:
buffer = (char*) malloc (i+1); // Exijo i+1 *bytes* do sistema operacional
if (buffer===NULL) // Se o SO não me dá memória
exit (1); // Eu me mato

2 Podemos até ter um vislumbre do comportamento neurótico no "swap": "Ok", diz o sistema operacional, "eu te dou alguma memória, mas você vai ver como tudo ficará lento..."

De fato, uma das suposições mais importantes de quase todo programa é a de que ele próprio está certo. A maioria dos programas opera sob a suposição de que, independente do estado de sua memória, esta foi produzida por uma execução fiel do algoritmo que deve ser implementado. Às vezes, quando o programador tem razões (ou inquietações) para evitar esse tipo de suposição, ele introduz no programa uma técnica de nome revelador: "verificação de sanidade", uma verificação especial, projetada para garantir que o programa foi executado de acordo com determinada etapa do planejamento. Mas uma verificação como essa só é acrescentada de vez em quando e, obviamente, não pode dar conta de tudo. A suposição implícita de que o eu está certo é inerente a todo programa, e corresponde à definição lacaniana de "ego".

O algoritmo da máquina de Turing é, portanto, parte do Outro do programa que o implementa: o ego, ou aquilo que as pessoas chamam de "eu". O ego, é claro, só existe na mente dos programadores – e ainda assim o programa real se apoia nesse pressuposto. Na maioria dos casos, a suposição de um "ego" pelo programa é apenas implícita; consideremos agora um caso no qual ela é explícita.

A cadeia de significantes

Considere um problema comum: para conseguir um emprego, é preciso ter experiência de trabalho; para ter experiência de trabalho, é preciso ter um emprego.[3] Tal "*catch-22*" é típico da matemática: muitas de suas estruturas são definidas com base nas suas estruturas mesmas. Tomemos como exemplo a seqüência de Fibonacci:

$$\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2)$$
$$\text{fib}(0) = 1$$
$$\text{fib}(1) = 1$$

Eis um dos mais simples exemplos de recursividade. Matemáticos têm facilidade para compreendê-lo, comparativamente, porque as duas primeiras linhas do exemplo parecem uma definição suficiente da função. No entanto, um computador as entenderia, pelo menos a princípio, não como uma definição, mas como uma equação: ou seja, não como uma descrição direta e inequívoca, mas como um enigma a ser resolvido. Enquanto algumas definições de funções dadas ao computador são de fato atos puros de criação dessas funções, nesse caso a situação é mais complicada.

Isso ocorre porque a definição acima é auto-referencial. Ela não significa apenas "*por definição, tem-se* fib(n) = fib(n – 1) + fib(n – 2)", mas sim "*se* fib existisse, fib(n) *seria igual a* fib(n – 1) + fib(n – 2)". Se tal função não existisse, pelo contrário, tal

3 A versão berlinense é: para obter um registro, você precisa de um aparatamento; para obter um apartamento, você precisa de um contracheque; para obter um contracheque, você precisa de um registro. Há formas de resolver o problema da experiência de trabalho, mas a solução do problema berlinense sempre envolve algum tipo de milagre.

definição não seria significativa; então a definição depende de uma suposição de que a coisa definida já existe. Portanto, não podemos entender a definição de um ato de criação do objeto sem etapas adicionais. Note, é claro, que não estou falando de algum tipo de impossibilidade escolástica; trata-se de um problema técnico real, que precisa ser resolvido.

Há, grosso modo, duas maneiras de resolver este problema, e a mais simples de entender é a nomeação de coisas. As funções matemáticas não precisam verdadeiramente ter nomes. Elas existem exatamente como correspondências entre conjuntos – não dependem de nomenclaturas. É bom se referir a uma função por seu nome às vezes, digamos, em um ambiente acadêmico, mas essa nomeação não tem impacto em comprovações matemáticas reais.

No entanto, a nomeação tem um uso significativo em computação.[4] Pois uma maneira muito simples de implementar uma função recursiva como "fib" é, em vez de considerar o que "fib" significa por sua definição, evocar "algo chamado fib". Isso basicamente aplica o nome como promessa: não, não temos a menor ideia do que exatamente é "fib", mas prometemos que vai haver algo chamado "fib" quando você procurar.

Então o que é o nome de uma função, nesse caso? É um símbolo especial que o representa para outra função. Esta é exatamente a curiosa definição de significante em Lacan: aquilo que representa um sujeito a outro significante. Os significantes de Lacan estão encadeados um ao outro: um deles sempre se refere a outro, e este a outro... até que não se refere

4 Não deixe de notar todas as diferenças entre a forma matemática e sua necessidade de implementação – sempre insinue no real, e mesmo que tal diferença possa em princípio ser formalizada, a situação em questão conterá essas diferenças e elas estarão exatamente onde temos que procurar.

mais. O método psicanalítico da livre associação prossegue mecanicamente com uma associação após a outra, até que pára, e o sujeito não consegue produzir a associação seguinte. E agora o *verdadeiro* trabalho psicanalítico pode se dar – uma lacuna entre significantes precisa ser preenchida.

Da mesma forma, funções sempre se encadeiam a outras funções, e estas a outras funções... até que isso não ocorra mais. Em algum ponto as cadeias de nomenclatura das funções passam o trabalho para um Outro que está além do registro simbólico do programa: por exemplo, para o CPU, pedindo-lhe que realmente acrescente dois números, ou então para o SO, pedindo-lhe que forneça mais memória. Mas e se o Outro recusar ou se rebelar? O programa tem agora um problema, um problema que raramente pode ser solucionado de forma autônoma.

Combinador de ponto fixo e duplo entendimento

Outra técnica para superar o desafio imposto pela recursividade é o uso do "combinador de ponto fixo", que permite criar uma função recursiva sem esconder sua inexistência por trás de uma nomenclatura. Recordemos o exemplo da experiência de trabalho que propusemos acima. Um das maneiras de resolver tal problema é fingir que já se tem um emprego: fazer

algo que não seria um emprego (provavelmente não-remunerado), mas que envolva habilidades relevantes à área. Tal ação, por si só, produziria a experiência necessária e tornaria possível o futuro emprego.

Está aí um exemplo típico do impasse presente na neurose, o da criação de significantes *ex nihilo*. Considere uma situação que muitos tomariam como familiar. Alguém quer ser, digamos, um grande filósofo, e vê um filósofo como aquele que escreve grandes trabalhos de filosofia. No entanto, esse alguém diz para si mesmo, já é preciso ser um grande filósofo para escrever um grande trabalho de filosofia. Não é apenas fisicamente impossível fazê-lo: *não lhe é permitido* tentar escrever um grande trabalho de filosofia porque ele ainda não é reconhecido como filósofo. O reconhecimento é, nesse sentido, importante para a realização de determinadas capacidades: é fácil entender a noção de que é preciso já ser poeta para escrever poesia, ou de que é preciso já malhar para se inscrever em uma academia. O significante, sendo um *alguém*, não pode ser criado *ex nihilo*: apesar de isso ser perfeitamente plausível, é incompatível com o ego – com o Outro. Se você tentar ser alguém que ainda não é, é muito provável que fracassará.

Esse tipo de ansiedade[5] é um problema que temos que superar repetidamente ao longo de nossas vidas. Para um neurótico obsessivo, porém, superar é impossível. Muitas vezes ele sabe o que quer ser, mas não tem idéia de como superar o estatuto de não-ser. Uma das coisas que se deve reconhecer aqui é a importância da *ação* na definição de um *alguém* particular. O "discurso de Roma" de Lacan descreve esse processo geral como duplo movimento:

5 O que na verdade é bastante legítimo – não é como se todos pudessem simplesmente sair escrevendo poemas por aí sem repercussões.

A função simbólica se apresenta como um duplo movimento no sujeito: o homem faz de sua própria ação um objeto, mas apenas para devolver-lhe, em tempo hábil, seu lugar fundacional. Nesse equívoco, operando a cada instante, está todo o progresso de uma função em que ação e conhecimento se alternam... na fase um, um homem que trabalha no nível de produção em nossa sociedade se considera pertencente às fileiras do proletariado; na fase dois, em nome desse pertencimento, se une a uma greve geral.

Este processo é exatamente a realização do sujeito, de acordo com Lacan – ele sempre já passou por tal movimento.

Voltemos ao nosso problema de recursividade. Temos uma função chamada "fib", que requer que "fib" preexista a si. Como podemos, dada tal função, criar "fib" *ex nihilo*, isto é, sem exigir que ela exista de antemão? O modo como o cálculo lambda resolve esse problema sem produzir novos nomes é utilizando o combinador de ponto fixo. Sua tipologia – meta-nível ou descrição lógica do que deve acontecer – é "(x -> x) -> x". Lê-se: "dada uma máquina que faz 'x' de 'x', faça um 'x' de tal máquina em si mesmo". Eis uma máquina muito estranha: as máquinas que ela propõe querem que haja um "x", mas não há "x" por perto – e ainda assim, de alguma forma, extrai-se um "x" da própria necessidade de que haja "x".

O "x" aqui só pode ser uma *função* (ou outra encarnação da computação recursiva). Em termos lacanianos, e até hegelianos, a máquina "x -> x" seria parte da ação própria a "x", um passo infinitesimal que não muda "x", mas permite que "x" exista como uma soma de tais passos. Ou melhor, em termos nietzschianos: "x" é antes de tudo uma ação, porém a gramática maquínica exige que essa ação seja expressa como um objeto: é

apenas a partir do "->" que se segue o seu significado. O combinador de ponto fixo encarna uma ação em uma função. Não é expressável em cálculos lambdas simples, e requer tanto ajuda do sistema (que o provê localmente como objeto místico) quanto boa-fé do programador (que se contenta com tentar encarnar uma ação que tornaria o mundo inconsistente).

Tal é o mistério da encarnação do cálculo recursivo (esse é o termo mais geral e honesto, do qual "realização do sujeito" é outra instância secularizada). Considere como o mistério funciona. Nunca há um milagre real e perceptível: sempre que executamos o programa criado assim, ele segue etapas técnicas muito específicas e óbvias. A solidez de uma execução particular não é nada mais que fato positivo. Entretanto, quando temos o programa à nossa disposição, já existe uma realização virtual (no sentido whiteheadeano/deleuzeano) que é sólida o suficiente. Essa existência virtual é o milagre e o mistério aqui: de vez em quando, é parcialmente encarnada pela execução do programa, mas ela não é encarnada o bastante para ser verificada, nem é autossuficiente o bastante para ser considerada regular.

Isto pode parecer um mero exemplo do problema da indução humeano, mas note que não nos referimos a leis da natureza e sua observância, nem a nossa crença na validade do programa em particular, tampouco a sua construção por vir. O programa tem uma relação complicada com o código-fonte, que é seu fundo teórico. Mas a ordem entre a realidade e a teoria aqui está invertida – a teoria realmente vem em primeiro lugar, como alguns filósofos gostariam que viesse – e mesmo assim não é possível resolver o problema da necessidade de se acreditar nela ou não. O operador de ponto fixo, ferramenta

necessária e útil, é uma demonstração gritante de que não basta coerência lógica.

Assim, há uma promessa específica discernível no uso do combinador de ponto fixo, que é uma promessa de consistência. Essa promessa é dada pelo programador, que, por sua vez, confia em outra promessas – na consistência da matemática, de seu conhecimento técnico e de sua intuição (na medida em que comprovações raramente são escritas). Essas promessas não são dadas explicitamente, pois quem quer que precise dá-las apenas recorre a instâncias superiores; e o hipotético Uno que teria poder real para colocar as promessas à prova só pode ser o próprio Outro a quem a promessa é conferida – um Outro que não existe de fato. A falta do Outro, a própria falta que forma o sujeito, não está situada entre humanos: ela penetra tudo, da computação à matemática, como buracos nas peças da torre de Hanói.

Uma das manifestações mais importantes dessa falta é a dissimetria entre a teoria e a realidade da computação. Como demonstramos, mesmo a informática, campo inteiramente constituído de objetos feitos pelo homem, é incapaz de escapar dela. A visão psicanalítica dessa diferença entre teoria e realidade é que ela sempre existe nas pessoas[6] e é fonte de sofrimento. Entretanto, é fácil observar que ela não se limita às pessoas, e que os objetos técnicos no mínimo exibem também tal dissimetria – eles dependem de sua própria inexistência e

6 Não acreditar nessa existência virtual é acreditar que cada vez que executamos o programa ele é executado fielmente – o que é uma crendice ainda maior. Há também um precedente religioso, a saber, a teoria sunita da causalidade dos Asharitas, que acreditavam que não é o movimento da caneta que causa o aparecimento da escrita no papel, mas Deus que, independentemente, causa o movimento da caneta e o aparecimento da escrita no papel.

são constantemente perturbados por sua própria presença. Perceber a dissimetria não é uma atividade apenas teórica ou apenas prática. O correlacionismo presente nos objetos técnicos não é apenas um correlacionismo humano, e a crítica habitual a este, assim considerada, pode ser exatamente o erro antropocêntrico que tentamos evitar.

Com isso concluímos uma primeira tentativa de psicanalisar o processo de computação. Aprendemos a procurar a falta da máquina nas discrepâncias entre ela e suas formas ideais; descobrimos que a computação depende do Outro (e que este Outro não é confiável); demonstramos a importância da dimensão simbólica na computação; e encontramos, nas maneiras de implementar a recursividade, analogias com o método psicanalítico. Assim tentamos expandir o domínio da problemática psicanalítica para além do corpo e da alma – para a teoria e a prática dos objetos técnicos. Sempre foi o objetivo de Lacan apresentar o método psicanalítico como função formal – esperamos que este trabalho não seja uma fantasia humanizadora sobre o computador, mas sim um olhar desumanizador sobre a psicanálise, que ajude a compreender melhor sua própria maquinaria, assim como ensine a ver o ser real e não matemático da maquinaria digital.