UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INTITUTO DE PSICOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM PSICOLOGIA SOCIAL E INSTITUCIONAL

CARLOS ANTONIO CARDOSO FILHO

MÁQUINAS, MÔNADAS, DAEMONS:

UMA BREVE HISTÓRIA E FILOSOFIA DA MÁQUINA UNIVERSAL DE TURING

PORTO ALEGRE, 2016

CARLOS ANTONIO CARDOSO FILHO

MÁQUINAS, MÔNADAS, DAEMONS:
UMA BREVE HISTÓRIA E FILOSOFIA DA MÁQUINA UNIVERSAL DE TURING

Tese apresentada ao Programa de Pós-Graduação em Psicologia Social do Instituto de Psicologia da Universidade Federal do Rio Grande do Sul, como requisito para obtenção do título de Doutor em Psicologia Social e Institucional.
Orientadora: Profa. Dra. Tania Mara Galli Fonseca

PORTO ALEGRE, 2016

CIP - Catalogação na Publicação

Cardoso Filho, Carlos Antonio
Máquinas, Mônadas, Daemons: uma breve história e filosofia da máquina universal de Turing / Carlos Antonio Cardoso Filho. -- 2016.
270 f.

Orientadora: Tania Mara Galli Fonseca.

Tese (Doutorado) -- Universidade Federal do Rio Grande do Sul, Instituto de Psicologia, Programa de Pós-Graduação em Psicologia Social e Institucional, Porto Alegre, BR-RS, 2016.

1. Turing. 2. máquina universal. 3. daemon. 4. mônada. 5. subjetividade. I. Fonseca, Tania Mara Galli, orient. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da UFRGS com os dados fornecidos pelo(a) autor(a).

Agradecimentos

À Linda, por ter me acompanhando até aqui e, apesar de todas as distâncias, ter sempre se mantido nessa proximidade, tão rara e preciosa, de quem sabe que encontrou a pessoa certa. Sem teu apoio, e do Nick, jamais teria chegado aqui. E desculpe por te obrigar a ler a tese inteira!

À Tania, por sua orientação e, principalmente, por sua amizade, por ter apostado e confiado na proposta e no trabalho que levou a esta tese, que muitas vezes pareceu fora do meu alcance. Agradeço À oportunidade de ter participando do grupo de pesquisa nesses quatro anos e pelas maravilhosas aulas no Hospital São Pedro, assim como todas as parcerias e conversas.

À Taninha, pela grande amizade, conversas, desabafos, chás, cafés, almoços, passeios e tantas outras coisas que nem sei como agradecer.

À Katy, por toda a hospitalidade e pela amizade que se formou, assim como as parcerias e desesperos acadêmicos.

À Cleci, pelas interlocuções, colaborações, projetos e amizade.

Ao Carlos, por suas críticas que são sempre bem-vindas e por me indicar um milhão de referências legais que nunca conseguirei ler.

Ao Luis Artur, pelas parcerias e discussões filosóficas sobre sólidos e líquidos.

À Alana, pelas conversas que sempre me deixavam com ideias e me faziam ler coisas que definitivamente não podia ler naquele momento.

Ao Luciano, por nossas aventuras enciclopédicas e instantes blanchotianos.

À Barbara, por todas as nerdisses e batatas, assim como pelo infinito conhecimento de literatura japonesa.

À Joy, por tantas conversas, ideias e divagações que não cabem nesse mundo.

À Marina, não mais pela sopa de ervilha, mas por ser a única pessoa que se empolga por Linux e criptografia junto comigo, apesar de eu explicar tudo pela metade.

Ao Rafael, que esteve junto a tudo o que originou este trabalho, parceiro anônimo e correspondente da primavera árabe, ávido por torresmos.

A máquina é o meio pelo qual o homem se opõe à morte do universo; a faz mais lenta, como a vida e a degradação de energia, e se converte em estabilizadora do mundo.

- Gilbert Simondon

Ruído ininterrupto de máquinas.

- Gilles Deleuze e Félix Guattari

What really exists *is not things made but things in the making.*

-William James

We only know the concrete by small points; we manufacture a lacunary world. Yet our worries and the deontology to be sought pass through these absent connections. The blackest of our ignorances, set off by the brilliance of our knowledge, doesn't reside in the precisely delimited black boxes but runs through the networks that connect them, whose parts we haven't explored.

- Michel Serres

I don't want to give a definition of thinking, but if I had to I should probably be unable to say anything more about it than that it was a sort of buzzing that went on inside my head.

- Alan Turing

Resumo

A máquina universal, ideia proposta por Alan Turing em 1936, seria uma máquina capaz de computar e executar qualquer máquina computável, vindo a ser tomada como um dos modelos abstratos do computador. Colocamos como problema a questão de saber o que pode uma máquina universal. Propomos dois modos de abordar a constituição de um pensamento sobre máquinas universais. O primeiro modo será desenvolvido através da construção de *redes*, no sentido que lhes é dado pela Teoria Ator-Rede. Com as redes teceremos uma narrativa histórica da máquina universal, com o intuito de mostrar que não há *uma* máquina universal, mas uma multiplicidade de máquinas, cada uma ligada e produzida por determinados coletivos que agregam as mais diversas forças, alianças e controvérsias envolvendo atores humanos e não-humanos. Esta história será realizada, primeiro, no âmbito de lógica e da matemática, e posteriormente em sua relação com linguagens de programação, engenharia de *software* e *software* livre. O segundo modo de constituição de um pensamento acerca de máquinas universais será dado por *dobras*, entendidas como procedimentos pelos quais o tempo dobrado, aproximando problemas, temporalidades e conceitos. Nas dobras a máquina universal será problematizada primeiro em sua relação com o pensamento na obra de Turing, segundo com o problema da interação e seus paralelos com as mônadas fechadas e abertas e, em terceiro, na relação estabelecida com a subjetividade através da figura do *daemon* e da concepção de uma cognição estendida.

Abstract

The idea of "universal machine" was proposed by Alan Turing in 1936. The universal machine would be a machine able of computing and executing any computable machine, being taken as one abstract model of the computer. As a problem, we put the question about what can a universal machine do. We propose two ways to approach the creation of a thought about universal machines. The first mode will be developed through the construction of networks, in the sense given to them by the Actor-Network Theory. With the networks we will weave a historical narrative of the universal machine, in order to show that there is not a universal machine, but a multitude of machines, each linked and produced by certain groups that congregate the most diverse forces, alliances and disputes involving human and nohuman actors. This history will be made, first, in the context of logic and mathematics, and later in its relation to programming languages, software engineering and open source software. The second mode of thinking about universal machines will be made through folds, understood them as procedures by which time is folded, approaching problems, concepts and temporality. In the folds the universal machine will be problematized first in its relationship with the thought in the work of Turing, second, with the problem of interaction and its parallels with the open and closed monads and, thirdly, the relationship established with the subjectivity through the daemon figure and the concept of extended cognition.

Sumário

Índice de figuras

1 Introdução

> *Há tão somente máquinas em toda parte, e sem qualquer metáfora: máquinas de máquinas, com seus acoplamentos, suas conexões.*
>
> - Gilles Deleuze e Félix Guattari

1.1 Máquinas, guerras e liberdade

Estabelecer o momento preciso em que algo tem início nem sempre é uma tarefa fácil, ou possível. Mesmo quando encontramos tais momentos, não definem um verdadeiro começo, são antes pontos de convergência do que origens absolutas. Se começos não se referem a uma criação a partir do nada, podem ser utilizados, entretanto, como marcadores, termos úteis para demarcar um ponto de virada, aquele instante em que tudo muda, onde o tempo parece se curvar e nos colocar em outra ordem de acontecimentos. O começo é mais um atalho e um desvio do que uma explicação.

Neste sentido, podemos dizer que este trabalho começou muito antes de sua escrita, antes mesmo de sua concepção. Mais precisamente, em 3 de dezembro de 2010, com um *tweet*:

"A primeira infoguerra séria está em curso. O campo de batalha é o WikiLeaks. Vocês são as tropas. #WikiLeaks" (BARLOW, 2010)[1] [i]

Como a *hashtag* #Wikileaks sugere, o *tweet* se deu no contexto da publicação de milhares de cabos diplomáticos norte-americanos pelo Wikileaks, com início em 28 de novembro de 2010. No dia da publicação, e nos dias seguintes, o site do Wikileaks foi atacado por uma série de DDoS[2], retirando-o do ar. Em paralelo, a Amazon, cujos servidores eram utilizados pelo site, cancelou sua hospedagem, aparentemente a pedido do governo norte-americano, ao mesmo tempo em que Paypal, Visa e Mastercard pararam de receber pagamentos e doações direcionados ao Wikileaks. Os ataques provinham tanto de *hackers* e *hacktivistas* que defendiam os interesses do governo norte-americano, quanto do próprio

1 Todas as citações em outras línguas foram traduzidas pelo autor. As versão originais encontram-se em notas ao final do texto.

2 *Distributed Denial of Service Attack*, um ataque que utiliza uma rede de computadores para realizar um número excessivo de requisições e acesso a um servidor, causando sua sobrecarga e o impedindo de ficar no ar.

governo, de seus aliados internacionais e do setor privado. O grupo *Anonymous* logo deflagrou a *Operation Payback*, cujo objetivo era realizar uma série de ataques DDoS contra os sites e organizações que se posicionavam contra o Wikileaks, como Paypal, Mastercard e determinadas instituições bancárias.

Durante este período, encontrava-me no final do mestrado em psicologia, escrevendo minha dissertação. Havia tomado conhecimento do *Anonymous* nas semanas que precederam a publicação dos cabos diplomáticos, mais precisamente durante os ataques do grupo aos sites da MPAA (*Motion and Picture Association of America*) e outras associações de artistas, em retaliação ao seu apoio a ataques feitos contra o site do Pirate Bay. O *Anonymous* originou-se no 4chan, um dos maiores fóruns de imagens da internet, no qual a postagem anônima é permitida e incentivada, o que faz com todos os usuários tenham o nome "Anonymous". Dentro do fórum, por volta de 2003, começaram a ser organizados "*raids*", invasões em massa de outros sites ou jogos, como Habbo Hotel e Second Life, com o intuito de perturbar e criar caos (COLEMAN, 2014).

Em 2008, o *Anonymous* assume um caráter político, quando diversos protestos contra a Igreja da Cientologia são organizados, em razão de suas tentativas de censura na internet. Além de utilizar ataques DDoS, trotes telefônicos e outras formas de ataque, protestos nas ruas também foram organizados. A máscara de Guy Fawkes, utilizada no quadrinho e no filme *V de Vingança,* foi adotada um dos símbolos do grupo, tanto por seu sentido na ficção, quanto por efetivamente permitir que os membros se tornassem anônimos ao possuírem a mesma face (COLEMAN, 2014).

Seguiram-se outras "operações", chegando ao caso do Wikileaks. O que me atraiu no grupo foi tanto o seu modo de organização único, quanto seu caráter idiossincrático, o que os aproximava do conceito de "multidão" proposto por Negri e Hardt (2004). A princípio, o grupo não possuía nenhum tipo de liderança ou modo de organização pré-estabelecido. Qualquer um podia, portanto, propor uma operação em nome do *Anonymous*. Caso conseguisse um número suficiente de participantes para colocá-la em prática, viria a se tornar, de fato, uma operação do *Anonymous*. O coletivo era constituído de forma completamente auto-emergente e descentralizada, muito similar a um enxame. Posteriormente surgiram subgrupos, porta-vozes, lideranças e agentes infiltrados do FBI, mas neste momento a organização ainda era em grande parte viral, e levou um tempo até que os veículos de mídia compreendessem como um grupo poderia funcionar sem lideranças.

Além de seu caráter emergente e descentralizado, o que transformava o grupo em uma multidão era a sua composição heterogênea. Participavam de uma operação tanto pessoas

que estavam completamente implicadas com a causa, quanto quem estava ali só pelo "lulz", termo que significava um certo humor negro, em querer se divertir criando caos. A multidão não forma um bloco monolítico onde todos devem agir e pensar de modo similar. Ao contrário, o que define a multidão é a diferença e a singularidade, a capacidade de agir junto sem abrir mão de todas as diferenças. O fascinante do *Anonymous* era justamente toda essa diversidade.

Este período coincidiu, também, com minha descoberta do movimento do *software* livre, quando finalmente propus-me a entender e me aprofundar nas questões políticas e sociais ligadas ao *software*. Tomei conhecimento dos partidos piratas, da noção de *copyleft*, das questões relativas à propriedade intelectual e escassez artificial, à política do compartilhamento, e do modo de produção *open source*. E, claro, comecei a usar Linux. Mas entrei em contato, também, a partir de Negri e Hardt, com a literatura dos movimentos anti/alter-globalização. Para quem nasceu na década de 80 e cresceu durante os anos 90, não era difícil ter a impressão de que a história havia terminado. Minha primeira experiência verdadeiramente política foram os protestos contra o aumento da tarifa de ônibus de Florianópolis em 2004, dos quais participei também em 2007 e 2010. Enquanto o primeiro constituiu uma experiência de organização autônoma, sem lideranças, os protestos subsequentes se tornaram cada vez mais centralizados e com uma repressão maior e mais efetiva da polícia.

No final de novembro de 2010, quando o *Anonymous* iniciou sua operação de apoio e defesa ao Wikileaks, participei avidamente da ações. Inicialmente, "participar" consistia em baixar o LOIC[3], programa utilizado para realizar os ataques aos sites. Seu funcionamento era simples, bastava fornecer um endereço e o programa tentaria acessá-lo centenas ou milhares de vezes em um curto período de tempo, repetindo o processo indefinidamente, não muito diferente de um navegador que recarrega a mesma página incessantemente, O seu diferencial encontrava-se no fato de que poderia ser controlado remotamente, fazendo parte de algo similar a uma *botnet*. Em um DDoS, como o nome implica, o ataque é distribuído, são utilizados centenas ou milhares de computadores ao mesmo tempo. Esta rede é quase sempre uma *botnet* composta por computadores infectados por algum tipo de vírus, o que os transforma em computadores zumbis ou robôs, *bots*. No caso do LOIC, a rede formada por nós era uma *botnet* voluntária controlada pelos usuários de um canal de IRC[4].

Os *chats* no IRC constituíam a segunda forma de participação, pois era no canal

3 LOIC é um acrônimo para "Low Orbit Ion Cannon", referência a uma arma do jogo Command & Conquer.
4 Internet Relay Chat, um protocolo de bate-papo utilizado por vários programas, organizado em canais que funcionam como salas de bate-papo.

oficial da operação que milhares de usuários reuniam-se e decidiam por votação qual seria o próximo alvo. Neste caso o "oficial" significava que era o canal com maior adesão. Mas foi também no IRC que eu e mais um amigo entramos em contato com os *anons*[5] brasileiros e, juntos, passamos uma madrugada escrevendo colaborativamente com completos desconhecidos uma nota à imprensa brasileira explicando as motivações e modo de funcionamento do grupo. A nota foi enviada a diversos sites de notícias, e tudo indica que nunca foi lida.

Foi em meio a tudo isso que, no dia 3 de dezembro de 2010, John Perry Barlow nos convocou à participar da primeira infoguerra. A importância do *tweet* residia não apenas em ter expressado com exatidão os ânimos e sentimentos daquele momento, mas também de quem o escreveu. Quatorze anos antes, Barlow escrevera um dos textos mais lidos e distribuídos na nascente *web*, a "Declaração de Independência do Ciberespaço", que começava da seguinte maneira:

> Governos do Mundo Industrial, gigantes aborrecidos de carne e aço, eu venho do Ciberespaço, o novo lar da Mente. Em nome do futuro, peço a vocês do passado que nos deixem em paz. Vocês não são bem vindos entre nós. Vocês não possuem nenhuma soberania onde nos reunimos.
> Não temos governo eleito, nem estamos propensos a ter um, então falo para você sem nenhuma autoridade maior do que aquela pela qual a própria liberdade fala. Declaro que o espaço social global que estamos contribuindo é naturalmente independente das tiranias que vocês buscam impor sobre nós. Vocês não possuem nenhum direito moral de nos governar, nem possuem métodos de coação que nos deem verdadeiros motivos para temê-los. (BARLOW, 1996).[ii]

Este texto, que lia pela primeira vez, sintetizava aquele momento, expressava o sentimento de liberdade de quem agia fora de qualquer governo, fora das coerções das lideranças, centralizações, partidos e polícia. Lá, em meio à milhares de *anons*, experimentava essa sensação de estar em uma terra sem fronteira, o que descrevia a internet em seus primeiros anos. Barlow foi um dos fundadores da Electronic Frontier Foundation (EFF) em 1990, quando a internet ainda era uma "fronteira eletrônica", um território inexplorado que escapava ao domínio da lei e onde a liberdade de expressão era seu valor absoluto, o mundo habitado por *hackers* com seu imperativo de que "a informação quer ser livre". A EFF foi fundada como um meio de proteger a liberdade de expressão na internet, e a Declaração de Independência do Ciberespaço foi uma resposta às tentativas de regular a internet.

Para o *Anonymous*, o *tweet* de Barlow foi como uma corroboração de suas ações vinda de um dos seus maiores heróis. Todavia, teve início uma discussão acerca da validade dos métodos utilizados pelos *anons*, que se propunham a defender a liberdade de expressão mas, que ao derrubarem sites, tolhiam a liberdade de expressão daqueles contra os quais

5 "Anon" é um diminituvo de "anonymous".

protestavam. O próprio Barlow criticou os métodos utilizados pelo *Anonymous*, afirmando que não era isso que quis dizer com infoguerra. O que estava em questão era a própria natureza de um protesto *online*. Quando inúmeras pessoas bloqueiam o acesso a um site não era o equivalente virtual de um grupo que fecha ruas em sua marcha, ou que decide sentar em frente a um estabelecimento comercial, bloqueando seu acesso, como forma de protesto?

Estas controvérsias, e muitas outras que a acompanharam, buscavam dar conta do que veio a ser chamado de *hacktivismo*, o ativismo que acontece no ciberespaço. Não era uma experiência nova, protestos *online* existiam desde os anos 90 e tornavam-se mais frequentes com a popularização da *web* (JORDAN, 2004). Mas foi neste momento que o termo, e a ideia, se popularizou e saiu de seus confins virtuais. Especialmente porque *qualquer um* poderia fazer parte do *Anonymous*, podendo, inclusive, levá-lo a direções fascistas dependendo de como se dessem as articulações (o que veio a acontecer com determinadas vertentes do *Anonymous* brasileiro). Em uma das críticas feitas ao *Anonymous,* seu membros eram denunciados como não sendo verdadeiros *hackers*, mas *script kiddies*, que ao invés de desenvolverem seus próprios métodos de ataque, vencendo o sistema por sua astúcia, habilidade ou conhecimento, apenas utilizavam ferramentas criadas por outros. Mas era justamente essa facilidade e simplicidade que permitia a qualquer um sem o conhecimento técnico suficiente, o que me incluía, a participar do coletivo.

O sentimento de liberdade que desfrutei nessa época deu-se quase como o oposto da repressão crescente que encontrava nos protestos na rua. Ao mesmo tempo, por mais que houvesse uma certa ressonância com a experiência fronteiriça dos anos 90, muita coisa havia mudado nos vinte anos que se passaram. A internet e a *web* deixaram de ser um território fora de qualquer jurisdição nacional e, como inúmeros casos mostraram, incluindo Napster e Pirate Bay, nenhum site estava realmente livre da ação da lei e de suas especificidades nacionais (GOLDSMITH; WU, 2006). O próprio Google teve de censurar os resultados de sua busca para entrar na China. Por mais que o anonimato fosse o valor central do *Anonymous*, contraditoriamente, ao utilizar o LOIC esse anonimato era quebrado, já que o ataque realizado partia do endereço IP[6] das máquinas de seus usuários, sendo muito difícil mascará-lo. Por mais que fosse virtual, os protestos não eram sem riscos, e meu IP assim como o de milhares de outras pessoas encontravam-se registrados nos servidores atacados, e muito provavelmente nos bancos de dados do FBI e NSA. O número IP, o endereço de uma máquina na internet, é em certa medida sua identidade e o signo de seu pertencimento a uma determinada

6 O enderço IP (Internet Protocol) é um número atribuído a cada máquina que acessa a internet, é seu endereço virtual.

coordenada geográfica.

Em 2010, a internet não existia mais em um espaço separado, exclusivamente projetado nas telas de computadores onde expressaríamos um *self* completamente apartado de nossa vida em carne e osso (TURKLE, 1995). Nossa subjetividade não precisava mais se dividir entre duas vidas, *online* e *offline*, mas constituía-se na circulação e na composição de um e outro, tornando-os aspectos inseparáveis de nossas vidas. Lembro de um dia em que sai de casa enquanto deixei o LOIC rodando em uma máquina virtual em meu notebook. Durante todo o tempo em que estive fora, estava também participando dos protestos. Onde começava e terminava minha subjetividade? Não sentia que aquela máquina sob o controle de um coletivo quase que descontrolado, não fazia parte de mim. De algum modo, eu também agia, mesmo que a distância. Pensava nas máquinas desejantes de Deleuze e Guattari (2010), do seu sujeito descentrado que circulava entre máquinas, humanas e não-humanas, conectadas em seus fluxos heterogêneos.

O caráter coextensivo e entrelaçado do *online* e do *offline*, do virtual e da rua, tornou-se ainda mais evidente quando, também em dezembro de 2010 – enquanto os embates envolvendo o Wikileaks mantinham-se fortes –, teve início a Primavera Árabe, com os protestos na Tunísia, propagando-se em janeiro para o Egito e depois por diversos países do Oriente Médio e norte da África. Um traço marcante destes movimentos foi que, especialmente no caso da Tunísia e do Egito, as redes sociais, como Facebook e Twitter, desempenharam um papel fundamental para a organização dos protestos nas ruas, principalmente entre os mais jovens, enquanto sites como o Youtube foram utilizados para distribuir vídeos tanto das manifestações quanto para denunciar a violência policial e estatal. Para o ocidente, o Twitter era o principal, e mais rápido meio de obter informações.

Nos dezoito dias que se passaram entre o começo dos protestos no Egito e a queda do então presidente e ditador Hosni Mubarak, acompanhei avidamente todos os desdobramentos e acontecimentos da revolução egípcia. No Twitter, seguia dezenas de ativistas e jornalistas egípcios. Mas os momentos mais tensos, fortes e marcantes vi através das lentes da rede Al Jazeera, com sua cobertura ininterrupta dos protestos, que assistia via *streaming* por seu site, já que era um canal indisponível nas redes televisas ocidentais. De certo modo, para mim, a revolução foi televisionada. Lembro claramente das cenas em que manifestantes repelem um pelotão de policiais e uma das pontes do Cairo; da batalha campal diante do Museu do Cairo, que durou dias, no qual manifestantes pró-Mubarak tentavam avançar e direção à praça Tahrir, centro dos protestos, enquanto manifestantes contra o regime defendiam acirradamente suas barricadas, em meio a pedras e coquetéis molotovs que voavam de ambos os lados; ou de

quando o escritório da Al Jazeera foi invadido e fechado pelas forças do governo, enquanto transmitia ao vivo. Lembro, ainda, dos pronunciamentos de Mubarak, de seu tom paternalista dizendo que os manifestantes haviam sido ouvidos e que poderiam voltar para casa. Lembro, por fim, do anúncio pelo vice-presidente de que Mubarak havia caído.

Ainda que o *Anonymous* estivesse envolvido na primavera árabe em maior ou menor medida, especialmente através de ataques e *hacks* de sites, desta vez fui apenas um espectador. Mas não me sentia menos envolvido e magnetizado do que em relação ao Wikileaks. Mais precisamente, tudo parecia convergir, todas as linhas e fluxos conjugavam-se em um grande acontecimento global. Algo definitivamente havia mudado na forma de fazer política. E, no dia 27 de janeiro, o terceiro dia de protestos, algo sem precedentes aconteceu: o Egito desapareceu da internet. O pais inteiro ficou *offlin*e, desconectou-se do resto do planeta. Neste dia, ficou claro como a internet, a rede global descentralizada, dependia das fronteiras nacionais, como era frágil e poderia desaparecer apenas com uma ordem vinda de cima. Contudo, a desconexão do Egito representou também o oposto: o medo que o poder tinha da internet. Mais do que apontar a sua fraqueza, a saída do Egito da internet mostrou a sua força.

Muito foi, e ainda é, discutido acerca do verdadeiro papel e influência da internet nos protestos e revolução. Mas era inegável que, doravante, a internet, em suas mais variadas formas, seria uma força e um meio que jamais deixaria de estar presente em qualquer luta política. Foi ali, em meio a infoguerras, DDoS, *hacks*, e revoluções que este trabalho teve início. O que percebia neste momento era um fluxo que atravessava pessoas, máquinas, grupos e ideias. Quando Deleuze e Guattari (2010) concebem as máquinas desejantes, o termo máquina é utilizado no sentido mais amplo possível: uma máquina é qualquer coisa que se conecte a outra. Deste modo produziam-se as conexões mais heterogêneas, onde uma ideia conectava-se a um par de mãos, que por sua vez conetavam-se a um teclado, conectado a um computador, a uma rede elétrica e informacional, compondo-se com protocolos, tráfegos, agenciando mais ideias, olhos, corpos, dispositivos. A liberdade que existia neste momento não se encontrava apenas em um tipo de máquina, na internet, no ciberespaço, na mente ou nas ruas, mas existia justamente em seu entremeio, nas conexões heterogêneas, no fluxo.

Esta percepção só veio a se fortalecer quando ainda em 2011 sugiram os movimentos dos Indignados espanhóis e o Occupy Wall Street, levando a ocupações que se espalharam por todo o mundo. Eram movimentos descentralizados, que apareceram na confluência das mais variadas e discrepantes demandas, apontando para a insatisfação generalizada contra o capitalismo e, como veio a se tornar um dos slogans do Occupy, da oposição entre o 1% e o 99%, entre o império e a multidão. Novamente, a internet cumpriu um papel importante na

articulação do Occupy, onde o apoio do *Anonymous* colaborou para o crescimento inicial do movimento (COLEMAN, 2014). Mas, ao mesmo tempo, a rua esteve presente mais do que nunca, pois todos esses movimentos tinham como premissa ocupar o espaço, tomar para si praças e constituir dentro de seus limites um outro mundo (CASTELLS, 2013). As jornadas de junho de 2013 no Brasil acompanharam esse fluxo de protestos descentralizados, sem demandas precisas, emergentes e, especialmente, ambíguos. O que começou como protestos contra o aumento da tarifa em São Paulo, logo se espalhou pelo país como uma série de manifestações contra os mais variados problemas: a Copa, a corrupção, a reforma política. Ao mesmo tempo, em determinados protestos assumiam um caráter nacionalista e até mesmo reacionário. Eram vendidos "kits de manifestação", nos quais uma máscara de Guy Fawkes, símbolo do *Anonymous* e do herói anarquista de *V de Vingança*, era acompanhada por uma bandeira do Brasil.

Perante a amplitude e extensão desses acontecimentos globais, meu pré-projeto do doutorado colocava objetivos bem mais modestos: estudar a constituição da subjetividade através da escrita coletiva de verbetes e da criação do *software open source* da Wikipédia. O caráter específico destes objetivos mantinha, contudo, minha motivação inicial de compreender como era agenciada a composição de humanos e não-humanos, como essas máquinas heterogêneas podiam se organizar de forma livre, aberta, escapando às exigências do capital e levando à construção de uma outra comunidade e objetividade. Como de praxe em todo doutorado, abandonei o objetivo de meu pré-projeto já nos primeiros meses, o que foi motivado por tomar conhecimento da ideia de "máquina universal".

Em sua formulação mais simples, a máquina universal é uma máquina capaz de executar ou imitar qualquer máquina computável. Conceito que foi inicialmente desenvolvido por Turing (1936) em 1936, para abordar o problema da decidibilidade na lógica e na matemática e que, após a invenção do computador digital, veio a se tornar seu modelo abstrato e teórico. Os primeiros computadores foram construídos independentemente das ideias de Turing, mas já incorporavam a ideia de "propósito geral", isto é, ao invés de serem máquinas especializadas, que realizavam apenas determinados tipos de cálculo, poderiam ser reprogramadas para realizar qualquer tipo de cálculo ou operação que sua memória e arquitetura permitissem. Tanto no campo da lógica e da matemática, quanto no da engenharia, a ideia de máquinas universais, ou de propósito geral, apontavam para a criação de um novo tipo de máquina com um potencial aparentemente ilimitado.

A mudança de planos se deu porque foi na máquina universal que encontrei um dos elementos essenciais que permitia a conexão entre os fluxos mais heterogêneos que

atravessavam manifestações e movimentos daqueles anos. O mesmo modo de organização não poderia ter se dado de forma tão disseminada apenas com telefones, papel e caneta ou televisão. Uma das principais mudanças encontrava-se no acesso, dispersão e, às vezes, na ubiquidade das máquinas universais: computadores, *smartphones*, *notebooks*, *lan houses*. A universalidade da máquina, a possibilidade de executar qualquer programa, constituía a liberdade que servia de ponto de conexão para todas as outras. Liberdade que se encontrava no cerne de diversos movimentos e grupos que os precederam: *software* livre, *open source, cypherpunks* criptoanarquistas e os mais variados tipos de *hackers*.

O problema político colocado pela máquina universal se tornou mais evidente quando, em 2012, deparei-me com um texto de Cory Doctorow (2011) em que anunciava uma iminente guerra pelo controle da computação de propósito geral. A liberdade propiciada pela máquina universal em suas muitas incarnações é o que permite que algo como a pirataria e cópia ilegal de músicas, filmes e jogos exista. Na medida em que um computador pode executar quaisquer programas dentro dos limites de seu *hardware* e sistema operacional, isso permite, também, que os seus programas sejam modificados por outros programas. Um jogo pode ser receber um *crack*, permitindo que rode sem necessitar de uma chave de validação, assim como uma música pode ser facilmente duplicada, mesmo que, em determinados casos, isso seja ilegal. Existe então todo um interesse por parte de estúdios, governos, empresas, fabricantes e escolas, dentre outros, em controlar e restringir a liberdade proporcionada pela universalidade e generalidade do computador.

A questão do controle agrava-se ao considerar a extensão em que os computadores permeiam nossa vida. Telefones celulares, máquinas de lavar, geladeiras, carros, aviões, marca-passos e implantes cocleares são alguns dos dispositivos que já se encontram computadorizados, que quando não são computadores plenos, possuem um em seu cerne, controlando todas as suas funções. Em nosso dia a dia, literalmente, entramos em computadores, assim como estes progressivamente entram em nós.

Desse modo, controlar um computador da maneira como fabricantes, governos, estúdios e instituições querem fazê-lo implica em necessariamente limitá-lo, em delimitar uma linha aparentemente muito clara do que pode funcionar ou não: você não pode ouvir esta música, ver aquele filme, jogar certo jogo ou falar com uma determinada pessoa. Ou melhor: "você pode executar **todos** os aplicativos, com exceção desse único que nos incomoda." O que está em jogo é muito mais do que a questão do controle, é a própria natureza do computador, daquilo que o define e o tornou, talvez, uma das tecnologia mais importantes do último século: ser uma máquina universal.

Um computador não tem como saber o que é ou não é permitido, sabe apenas seguir instruções. O que se quer é mudar essa natureza, inserir no conjunto de regras lógicas que utiliza para realizar seu processamento, regras morais e jurídicas de todo tipo, ao nível do próprio *hardware*. Pois quando se busca limitar a utilização de certo recurso ao nível do *software*, essa limitação é quebrada em questão de semanas, dias ou até mesmo horas. Isso é possível justamente pela natureza universal do computador, por constituir-se em uma superfície feita apenas de números, onde tanto o que comanda quanto o que é comandado compõem-se do mesmo material, 0 e 1, sendo, por conta disso, sempre potencialmente intercambiáveis.

A guerra se daria entre os que querem restringir a máquina universal, criando ambientes fechados, "jardins murados", onde apenas determinados programas previamente autorizados poderiam serem executados, e entre aqueles que querem exercer seu direito de utilizar e modificar suas máquinas como lhes aprouver. A defesa das restrições apoia-se do argumento da segurança. Ao permitir que apenas programas autorizados fossem instalados, a propagação de vírus e outros tipos de *malware* poderia ser evitada. O usuário não correria mais o risco de ter seu computador infectado. A universalidade do computador é o que lhe dá liberdade mas é o que também permite a existência de algo como o vírus de computador, cujo funcionamento depende da capacidade de modificar os programas existentes fazendo com que seu código seja executado, o que não é muito diferente de um *crack*.

O problema de colocar o controle de nossas máquinas na mão de terceiros implica em confiar em toda uma série de intermediários aos quais não temos acesso direto, especialmente quando o código do programa não nos é acessível. Deste modo, ficamos duplamente vulneráveis: tanto à vontade dos fabricantes de *software* e *hardware*, que podem decidir arbitrariamente o que podemos executar ou não, quanto a possíveis ataques e invasões dos quais não podemos nos defender já que não temos controle sobre nossas máquinas e seus mecanismos de defesa.

A liberdade da máquina universal é, no mínimo, ambígua, tal como a caixa da Pandora. Foi o que Edward Snowden revelou em 2013, quando expôs ao mundo a dimensão da vigilância realizada pela NSA e outras agência do governo norte-americano. Em grande parte, a extensão que essa vigilância tomou foi possível apenas porque nos encontramos cercados e imersos em um mundo de máquinas universais. Se podemos configurar e modificar nossos computadores de modo a torná-los mais seguros e quase imunes à vigilância, o inverso também se aplica. Grande parte de nossas comunicações e máquinas não são seguras, fato do qual as agências de vigilância tiram o maior proveito possível.

E aqui estamos, no começo de 2016, cinco anos depois de tudo o que motivou a escrita deste trabalho. De certo modo, parece que nada aconteceu. No mundo árabe, especialmente no Egito, a situação encontra-se pior ou mais caótica do que antes da revolução, com a Síria em meio a uma guerra civil brutal, o surgimento do Estado Islâmico e as intervenções do mundo ocidental. Qual foi o verdadeiro impacto do Occupy, ou do *Anonymous*? No Brasil, quais as consequências das jornadas de 2013, quando nos encontramos em meio a um golpe e um recrudescimento conservador? Enquanto isso, continuamos a ser vigiados, continuamos a utilizar redes sociais sem a menor preocupação com privacidade e segurança.

São questões difíceis e, talvez, a forma como estejam formuladas seja falaciosa, dado que não podem ser respondidas simplesmente em termos positivos ou negativos. Mas a sua existência atesta o fato de que durante esse tempo algo mudou. Definitivamente, neste momento, não há mais o mesmo sentido de liberdade que houve naquela época. O problema que se coloca não é o de que algo deva acontecer ou não, mas o de entender a natureza do que houve naquele momento. Em outras palavras, o surto de protestos conectados, de manifestações mediadas digitalmente, de organizações descentradas, de multidões e de novas formas de contestar o capitalismo, constituíram, em parte, epítome dos movimentos de alter-globalização, mas foram também a convergência de inúmeras tendências políticas, sociais, técnicas e econômicas que eclodiram em seu encontro. O poder da internet, das redes sociais e das máquinas universais mostrou toda a sua força, e toda a sua fragilidade.

Como Hardt (2014) afirmou sobre as manifestações no Brasil, o problema não é apenas como criar uma multidão, mas principalmente de como mantê-la, de como estabelecer um poder constituinte, uma democracia ou um espaço de liberdade que se mantenha nutrindo suas diferenças. Para Galloway e Thacker (2007), enquanto o poder era centralizado, a descentralização constituía um modo efetivo de resistência, mas agora que o poder também se descentralizou, como podemos resistir? A caixa de Pandora foi aberta, temos de explorá-la.

Neste trabalho situamo-nos a um passo, ou a vários, antes de todas essas questões. Nosso problema não se refere à internet, ao compartilhamento, aos modos de resistência, novas formas de produção econômica, colaboração, Wikipédia, internet, redes sociais ou *Anonymous*. Abordaremos, ao invés disso, exclusivamente alguns dos problemas colocados pela máquina universal. Pois como já foi dito, o elo comum entre todas essas novas formas de organização é a máquina universal em suas mais variadas formas e desdobramentos. De acordo com Feyerabend (2006), podemos considerar o "universal" em dois sentidos. O primeiro, de origem platônica, valoriza o universal em detrimento do particular. É universal

aquilo que supera e vai além de casos específicos, é o genérico, total e abstrato. Mas o universal pode ser considerado, também, como mediação, como algo pervasivo, que permeia o particular, que se espalha e cria um espaço comum de trocas. É neste sentido que devemos compreender a máquina universal: como um meio, uma mediação.

Para Latour (2012), o que distingue um intermediário de um mediador é que enquanto o primeiro constitui-se como um canal transparente de transmissão, previsível e neutro, o mediador, por sua vez, é opaco, transformando e modificando aquilo que transmite. A mediação acrescenta, remove, modifica, traduz. Nesse sentido, quando a máquina universal funciona como mediadora e não como intermediária, deixa de ser um meio passivo de transmissão e age junto com aqueles que media. São essas algumas dessas transformações produzidas pela introdução da máquina universal que exploraremos neste trabalho, o que nos leva às especifidades metodológicas do estudo da máquina universal.

1.2 Redes e dobras: uma topologia do pensamento

O video "Evolution of the Desk", de Thomsen e Georgiev (2016), com aproximadamente um minuto de duração, começa em 1986, com a câmera focada em uma mesa, sobre a qual se encontra um computador Macintosh Classic no centro, cercado por uma profusão de objetos: fax, telefone, relógio, enciclopédia, catálogo de compras, diário, agenda telefônica, calculadora, lista telefônica, globo terrestre, relógio, mural de recados, carteira, óculos, chaves do carro, dicionário, câmera fotográfica, jornal, e muitos outros. Na medida em que o video avança, e os anos passam, o computador é substituído por um PowerBook, e progressivamente cada um dos objetos transforma-se em um aplicativo, representado por um ícone na tela do computador: Wikipédia (enciclopédia), Amazon (catálogo de vendas), Google Maps (globo terrestre) e assim por diante. No final, em 2016, encontramos sobre a mesa apenas um MacBook Pro repleto de ícones e um iPhone que substituiu a câmera e o telefone (que também se transformou no Skype). O único objeto que se manteve inalterado foi o par de óculos. Até as chaves do carro foram substituídas pelos aplicativos Über e Lyft. As imagens 1-3 mostram as três etapas do processo. Mais do que apresentar a evolução das mesas de escritório, este curto vídeo nos dá uma dimensão do que é uma máquina universal. Em três décadas colocamos uma mesa repleta de objetos literalmente na palma de nossas mãos, em nossos *smartphones*, *notebooks* e *tablets*, acessíveis com apenas um clique.

A substituição de objetos por aplicativos não constitui uma simples transposição de

um meio a outro. É, ao invés disso, uma tradução, que modifica e desloca os elementos que coloca em relação (CALLON, 1986). Ter um carro não é mesmo que utilizar o Über, do mesmo modo que a fotografia criada por uma câmera digital, assim como qualquer arquivo digital, pode ser compartilhada, copiada, modificada e infectada de um modo que não seria possível à sua contraparte física. Por outro lado, quando a bateria no celular acaba não podemos acessar mais a Wikipédia, ou nenhum dos outros aplicativos, enquanto uma enciclopédia se encontra acessível ininterruptamente. A tradução de nosso mundo para a linguagem das máquinas universais não criou simplesmente um meio mais conveniente ou prático, mas modificou profundamente a nossa relação com o próprio mundo, o que nos leva à questão e problema desta pesquisa: o que pode uma máquina universal?

Pode-se dizer que vivemos em um *mundo de máquinas universais*, que grande parte de nossas ações diárias encontram-se de algum modo ligadas a essas máquinas, como trabalho, educação, lazer ou organização política. O modo pelo qual concebemos e utilizamos as máquinas universais, em cada uma dessas esferas, tem uma implicação direta nos modos de agir considerados. Quando um aluno utiliza uma plataforma de ensino à distância, como tal ação é concebida? Se o esperado é que o aluno utilize única e simplesmente a plataforma em

Figura 1.1: Cena de "Evolution of the Desk", ano 1986

Figura 1.2: Cena de "Evolution of the Desk", ano 2005

Figura 1.3: Cena de "Evolution of the Desk", ano 2016

questão, atendo-se exclusivamente às tarefas que lhe são dadas, o que acontece quando abre uma outra janela do navegador e começa a conversar com seus amigos em alguma rede social? E se os alunos utilizam a plataforma para troca de imagens e vídeos que não possuem nenhuma relação com as tarefas propostas? Ou, em um caso extremo, quando o aluno é capaz de reprogramar a própria plataforma, "*hackeando-a"* de alguma forma, ou simplesmente escrevendo um pequeno programa para automatizar exercícios e tarefas tediosas?

Como cada um desses casos será considerado já depende de uma ética e de um pensamento acerca das máquinas universais. Se a plataforma é considerada como mera ferramenta, cuja máquina universal em que é implementada deve ser um simples instrumento transparente a tal utilização bem definida e limitada, então qualquer ação que difira do esperado é concebida como indesejada. A máquina universal transforma-se tanto na origem do desvio quanto na possibilidade do controle. Inversamente, se a máquina universal é considerada enquanto um elemento potencializador, tais ações que ocorrem fora ou dentro da plataforma são de algum modo esperadas e até mesmo incentivadas, podendo criar outras oportunidades de aprendizagem. Estes modos de conceber a máquina universal constituem duas éticas, dois modos de agir e pensar. Serão os únicos? Esta pesquisa, busca evidenciar as condições de possibilidade pelas quais é determinado o que uma máquina universal pode fazer, o que implica na multiplicação das máquinas universais e dos modos pelos quais podemos considerá-las em nossas ações e decisões. Em outras palavras, buscamos problematizar e questionar o modo pelo qual nos relacionamos com estas máquinas, abrindo a possibilidade de *pensar de outra maneira*.

Em certo sentido, a máquina universal é a máquina mais forte, a mais versátil, na medida em que pode ser qualquer outra máquina, desdobrando suas potências. Entretanto, é também a máquina mais fraca, pois não faz nada sozinha. Funciona apenas enquanto se encontra acoplada a outra máquina. Se pode operar apenas na medida em que alista outras máquinas, como podemos fazer sua história? Teremos de levar em conta *todos* os programas que pode executar? Apenas alguns? Quais então? Mas se a máquina universal "desaparece", tornando-se transparente durante a execução de um programa, ficando em segundo plano, estaremos fazendo a história de um software específico e não da máquina universal. Como evitar, ao mesmo tempo, a inesgotabilidade de programas e a transparência da máquina universal?

Devemos retomar a distinção entre intermediários e mediadores. Se a máquina universal funciona como intermediária, desaparece então na execução de um programa, não lhe adicionada nada, é apenas um meio para um fim. Se, ao contrário, opera como mediadora,

intervém na ação do software, muitas vezes de forma imprevisível. O software, portanto, deve levar em conta a máquina em que será executado, onde a aliança é estabelecida, e o agregado de ações que aí se produz efetua-se apenas por uma conjunção específica de forças, ou fraquezas, dos atores envolvidos.

Se colocamos como problema a possibilidade de desenvolver um pensamento sobre e *com* máquinas universais, deveremos tomar como ponto de partida os momentos em que a máquina transforma-se em mediadora, quando é problematizada e produz controvérsias, quando alia-se e realiza conjunções. A existência da máquina universal, como foi visto, coloca em questão a evidência do funcionamento de uma rede sociotécnica que permeia nossas vidas, de sujeitos que vivem no mundo ocidental industrializado, em quase todas as suas dimensões. É por meio destas máquinas que nos socializamos e subjetivamos, que produzimos conhecimento e construímos objetividades, que resistências são organizadas e vencidas, seja através de sua utilização direta ou pelas redes extremamente complexas em que participam e mediam relações. É neste contexto que colocamos o problema de como constituir um pensamento e uma ética das máquinas universais. Como podemos pensar tais máquinas naquilo que possuem de próprio e que é irredutível a outros tipos de máquina, que lhe abre possibilidades e perigos específicos, que não encontramos em outros modos de prática e pensamento?

Um pensamento que não seja pura recognição, ou repetição de um já dado, pode se estabelecer apenas na medida em que exerce uma violência contra si mesmo, em que desconstrói seus pressupostos para que seja capaz de efetivamente criar (DELEUZE, 2006). Assim, propomos dois modos de abordar a constituição de um pensamento sobe máquinas universais. O primeiro modo, será desenvolvido a partir da construção de *redes*, no sentido que lhes é dado pela Teoria Ator-Rede (LATOUR, 2012). Através das redes constituiremos uma narrativa histórica da máquina universal, com o intuito de mostrar que não há *uma* máquina universal, mas uma multiplicidade de máquinas, cada uma ligada e produzida por determinados coletivos que agregam as mais diversas forças, alianças e controvérsias envolvendo atores humanos e não-humanos. Em uma rede, nenhum ator é uma substância ou um ser definido tautologicamente, sua existência constitui-se apenas à medida em que age e que é retomado por outros atores.

O segundo modo de constituição de um pensamento acerca de máquinas universais será dado por *dobras*, no sentido em que são entendidas por Serres (1999). A dobra é um procedimento pelo qual o tempo é considerado não mais em seu aspecto linear e unidirecional, sempre em direção a um progresso e seguindo um sentido ordenado. Ao invés

disso, ao dobrar o tempo, as mais diversas temporalidades podem ser aproximadas. A rede não assume um tempo linear e progressivo, o desconstrói ao multiplicá-lo em uma tessitura complexa que, contudo, ainda segue os atores de acordo com certos caminhos que traçaram. Com a dobragem, estes caminhos não são necessariamente tomados, é possível traçar as mais variadas conexões que os próprios atores não podiam estabelecer. Deste modo, com as dobras propomos um procedimento efetivamente filosófico e ensaístico no qual tomamos a máquina universal não mais pelas redes que a constituíram, mas pelas forças que aí encontramos e que levamos adiante, explorando as possibilidades abertas por tal história.

1.2.1 Redes

A breve história da máquina universal que nos propomos a realizar neste trabalho será realizada a partir de dois domínios de prática. No primeiro, partimos da formulação da ideia por Turing em 1936 e de como a máquina universal foi subsequentemente desenvolvida no campo da lógica, da matemática e na nascente ciência da computação. No segundo, a máquina universal será tratada a partir de sua relação com linguagens de programação, com o desenvolvimento de *hardware* e com a formação da figura e da profissão do programador.

Tomamos como inspiração metodológica a Teoria Ator-Rede (CALLON, 1986; LATOUR, 2012; LAW, 2004). Tal abordagem assume que mesmo os objetos científicos e matemáticos mais exatos ou verdadeiros são construções que não podem ser dissociadas do meio social e político em que foram criadas e posteriormente desenvolvidas. Ao mesmo tempo, considera que o que constitui o "social" não é composto apenas por relações humanas, por "puro discurso" ou meras "subjetividades". Afirma que o social é uma mistura heterogênea de sujeitos, objetos, relações humanas, forças naturais, vírus, substâncias químicas, divindades e uma infinidade de outras forças que compõem um mundo. Com isto, o objetivo não é nem tornar a natureza e os fatos científicos em uma simples "construção social", cuja verdade dependeria apenas da vontade, da retórica ou da sociedade, não possuindo uma "substância", e, inversamente, nem considerar o social enquanto mera camada de significação ou relações humanas colocadas sobre uma realidade natural e objetiva já estabelecida. Abandonamos as dicotomias entre natural e humano, objetivo e subjetivo, real e imaginário, verdadeiro e relativo, a favor de redes que "são ao mesmo tempo reais como a natureza, narradas como o discurso, coletivas como a sociedade" (LATOUR, 1994, p. 12).

O que garante a verdade de uma teoria ou proposição não é sua relação com a natureza, ou seja, a sua capacidade de representar mais ou menos fielmente uma realidade

pré-existente. Enquanto uma ciência ainda não está constituída, tanto a natureza quanto a realidade são entidades indeterminadas. À medida em que várias teorias conflitantes coexistem, cada uma tenta afirmar sua visão e definição de natureza. Quanto mais aliados uma teoria agrega, mais chance tem de ser considerada verdadeira. Os aliados podem ser compostos tanto por humanos, como outros cientistas, agências de financiamento, técnicos, quanto por agentes não-humanos como instrumentos, laboratórios, computadores, resultados de experimentos, provas matemáticas. Com isso não se quer dizer que a ciência, ou a realidade, sejam mera questão de retórica ou discurso. Sim, há um elemento retórico que não pode ser desconsiderado, entretanto o que se busca não é tanto questionar a objetividade dos resultados científicos (relegando-os para a esfera subjetiva), mas, ao contrário, expandir essa objetividade. Não são apenas os resultados e tampouco as teorias comprovadas que compõem a objetividade, mas, também, tudo o que foi necessário para obter essas respostas e teorias, como os instrumentos e modalidades discursivas que permitem considerá-las enquanto soluções. São essas etapas intermediárias, muitas vezes relegadas ao campo do "não-científico", que devem ser reintegradas à própria objetividade. A história de uma ciência, teoria ou técnica deve então seguir o princípio de simetria (LATOUR, 2000, 2012; LAW, 2004), que atesta que tanto o que é considerado verdadeiro, quanto o que é tomado como falsidade e erro, devem possuir o mesmo valor. O que não implica em dizer que verdade e falsidade sejam a mesma coisa, muito pelo contrário, afirma que ambas devem ser tomadas e desdobradas em toda a diversidade de práticas, discursos e ações que as constituíram. Ao mostrar tal batalha e considerar cada uma das teorias como sendo iguais às outras no processo em que travam sua disputa e uma ou mais delas saem vitoriosas, o objetivo não é dizer que o que acontece é um jogo retórico. A teoria que venceu é, *de fato*, verdadeira, tão verdadeira e objetiva quanto uma teoria científica pode ser em meio a todas as suas incertezas.

Este trabalho busca aplicar tal procedimento à máquina universal. Se hoje podemos abrir qualquer livro de ciência da computação e nele obter uma definição matemática exata do que é a máquina universal, e que concorda com todos os outros livros do mesmo tema, temos de nos perguntar como isso foi possível. Onde se encontram todos os conflitos, controvérsias, dúvidas e incertezas que foram necessárias para tornar possível uma definição tão exata? Podemos supor então que existiram, e ainda existem inúmeras máquinas universais. Algumas foram eliminadas, outras esquecidas, muitas se transformaram, e algumas poucas podem ter permanecido sem grandes modificações. Como isso aconteceu? A quais práticas, atores, situações e efeitos tais destinos foram atribuídos?

Escolhemos Turing e o artigo que escreveu em 1936 (TURING, 2004 [1936], 1936)

como ponto de partida. Poderíamos recuar muito antes disso, indo até Leibniz e o seu sonho de uma linguagem universal capaz de expressar e calcular todo pensamento humano (DAVIS, 2000) ou mostrar como a máquina universal tem sua condição de possibilidade nas práticas burocráticas de gerenciamento do século XIX (AGAR, 2003), especialmente nos cartões de catalogação de bibliotecas e escritórios (KRAJEWSKI, 2011). É uma tentação de toda pesquisa histórica começar pela origem absoluta de seu objeto, aquele ponto ideal em que tudo começou verdadeiramente, cujo desenvolvimento futuro seria apenas um desdobramento secundário deste lampejo original. Contudo, a realidade é sempre bem mais complicada, e nunca podemos atribuir um início claro e exato às coisas, a não ser que sejam reduzidas e simplificadas demais. As máquinas universais não surgem com Turing, contudo, é a partir dele que podemos encontrar muitos dos termos que definirão boa parte das controvérsias futuras e os problemas presentes.

Desta maneira não vamos tentar definir os termos "máquina" ou "universal" em um sentido absoluto. Serão os próprios atores de nossa história que, na primeira parte, terão de nos dar tais definições. Nosso objetivo é construir uma rede que evidencie os processos pelos quais tais definições se tornaram-se possíveis, indo além de corroborar atestar a verdade ou exatidão desta ou daquela concepção. Temos de seguir os próprios atores (LATOUR, 2012), ao invés de fazer com que nos sigam e se conformem a categorias pré-definidas. Nesse sentido, um ator pode ser tanto o próprio Turing, quanto as máquinas e os computadores humanos por ele definidos, assim como todos os outros que darão continuidade às suas pesquisas, e toda a pletora de objetos e forças que serão introduzidas: mentes, tubos de memória de mercúrio, circuitos, linguagens de programação, usuários, sistemas operacionais, protocolos, engenheiros, matemáticos, inteligências.

Há uma segunda simetria que deve ser considerada, não mais entre saberes verdadeiros e falsos, mas entre humanos e não-humanos, incluindo-os na construção das relações entre natural e social. A assimetria ontológica, que se situa muito próxima da epistemológica, concebe que exista uma diferença fundamental e intransponível entre humanos e não-humanos. Apenas os humanos seriam dotados de agência ativa, seja por sua consciência, vontade, inteligência, biologia, cultura ou alguma liberdade ontológica ainda mais fundamental. Discurso antropocêntrico, que coloca o homem de um lado e natureza do outro.

Tal postura exige, portanto, uma problematização da linguagem a ser utilizada em tal abordagem. As assimetrias ou assumem a linguagem dos atores, desenvolvendo-se nos termos que estes colocam, ou abandonam completamente a linguagem utilizada, pondo em seu lugar

um outro vocabulário pré-estabelecido e desenvolvido de modo completamente descolado das práticas em questão. Precisamos então de uma linguagem que leve em conta a linguagem desenvolvida pelos atores, sem, contudo, tomá-la em sua evidência, nem criticá-la ou denunciá-la em nomes de valores e categorias que lhe sejam estranhos. Contudo, isso não implica no desenvolvimento de uma metalinguagem por parte do pesquisador.

> Não devemos presumir que atores possuam uma linguagem enquanto os analistas dispõem de uma *meta*linguagem na qual a primeira está "inserida". Conforme já dissemos, concede-se aos analistas unicamente uma *infra*linguagem cujo papel consiste apenas em ajudá-los a ficar atentos à metalinguagem plenamente desenvolvida dos atores, um relato racional daquilo que estão falando (LATOUR, 2012, p. 79).

Os atores não se limitam a criar uma linguagem, desenvolvem também uma metalinguagem, pela qual explicam a si mesmos e justificam-se perante os outros. Não são entidades passivas, completamente imersas em um "senso comum" ou "cotidiano" irrefletido, aguardando a reflexão plenamente consciente e desenvolvida de um pesquisador. Muito pelo contrário, refletem acerca de si mesmos e de todo o mundo, criando metafísicas e cosmologias complexas. Se os atores que analisamos fazem parte do mundo da lógica e da matemática, que é o caso deste trabalho, podemos até mesmo dizer que nos encontramos perante os mestres das linguagens e metalinguagens, uma vez que as criam sem parar, já que cada sistema formal constitui uma linguagem ou metalinguagem. Ao utilizar uma metalinguagem recaímos exatamente nos mesmos problemas das assimetrias que descrevemos: ou reduzimos todos os discursos a uma grade de definições universais e generalizantes, completamente descolada de contextos específicos ou, inversamente, assumimos a metalinguagem, os termos e os pressupostos dos próprios atores.

Por isso, quando utilizamos o termo "ator" nunca sabemos exatamente ao que estamos nos referindo. Tal termo não é uma definição clara a exata que permita identificar uma entidade que já exista. É um termo vazio, que inclui absolutamente qualquer coisa capaz de agir. Por sua vez, "ação" também é um termo indeterminado. Esta aparente falta de clareza e exatidão, ao invés de ser um detrimento à produção do saber, constitui-se como um modo privilegiado de seguir os atores. Um ator é humano ou não-humano? É real ou imaginário? É verdadeiro ou falso? Não sabemos. São os próprios atores, em seus jogos de equivalências e negociações, que definirão tais limites (ou os deixarão em aberto).

Contudo, uma pesquisa não pode ficar apenas na indeterminação, afinal de contas devemos estabelecer como os atores definiram-se mutualmente através de suas práticas e performances. A indeterminação é um ponto de partida. Devemos começar por alimentar as controvérsias e abrir as caixas-pretas. A infralinguagem, por ser subdeterminada, dá espaço

para que os atores coloquem seus termos, dúvidas e certezas, sem submetê-los a uma organização prévia. Mas, ao seguir os movimentos dos atores, as incertezas vão dando lugar a certezas, vemos que certas coisas vão se solidificando, tornando-se duráveis e transformando-se em intermediários, ao mesmo tempo em que novos mediadores e controvérsias vão surgindo. Toda estabilidade advém dos atores, não do pesquisador ou de sua linguagem, que deve estar preparado para expressar o que não é claro e bem definido.

O pesquisador é um interrogador *entre* interrogadores, que apesar de utilizar instrumentos diferentes para realizar seu jogo (como a infralinguagem), coloca-se e circula no mesmo nível dos atores que analisa. Em outras palavras, o relato que será escrito como resultado desta pesquisa tentará tecer uma *rede* (LATOUR, 2012). Ao invés de colocar-se como um saber que vem do céu, que pode falar dos atores a partir de uma posição privilegiada, a construção de uma rede, se for bem feita, *desloca* os atores e suas relações, e, na mesma medida, se deixa deslocar por eles. Um relato só se torna uma rede ao agir de algum modo, ao desviar e ser desviado pelo mundo. Ao utilizar uma infralinguagem propomos não uma representação fiel da realidade, mas a construção e reconstrução de uma outra realidade. Uma história da máquina universal só é possível, ou interessante, quando trabalhamos com mediadores. A mediação, entretanto, não existe sozinha, e como o nome já implica, é uma relação. É preciso então entender como tal mediação se dá, pelos processos de *associação* e *substituição,* o que Latour (1992) chama de *programas de ação.*

É aí que podemos encontrar a possibilidade de diversas histórias de máquinas universais. A máquina de Turing, que nos parecia uma intermediária fechada e pronta, é associada a novos atores e substituída por outras matérias, tornando-se uma mediadora neste processo e abrindo o caminho para novos desdobramentos e controvérsias, ou sendo novamente estabilizada e fechada em uma caixa-preta. Esse jogo de deslocamentos, e equivalência, é o que Callon (1986) chama de tradução. "Traduzir é deslocar. […] Mas traduzir também é expressar na linguagem de um [ator] o que outros dizem e querem, porque agem, de que maneira o fazem e como se associam uns com os outros: é estabelecer a si mesmo como porta-voz." (p. 316)[iii] Pelo procedimento de substituição, o computador funciona como o porta-voz da máquina universal, expressa em termos materiais um princípio lógico. A tradução não se dá apenas ao nível da linguagem, envolve matérias e forças (ou fraquezas). É o que permite também que coisas sem ligação, como o computador (ou a máquina universal) e o cérebro possam estabelecer um contato e negociar suas posições. A história da máquina universal é o processo pelo qual ela foi traduzida, deslocada, associada e substituída.

Ao descrever os participantes de um processo de tradução, situamo-nos ao nível de uma infralinguagem, pois em nenhum momento tenta-se estabelecer uma relação causal entre os atores. São os próprios atores que o fazem. Ao colocá-los na grade de associações e substituições, podemos visualizar não apenas a causas atribuídas, mas o *modo pelo qual foram atribuídas.* Na medida em que realizam associações e substituições, os atores removem estes gestos de sua linguagem quando têm sucesso, permanecendo apenas o resultado ou a conclusão do processo. A infralinguagem não diz o que causou o quê, ao invés disso constitui uma rede que marca os processos pelos quais traduções e apagamentos foram realizados, *junto* com o que foi afirmado pelos atores em sua linguagem própria.

A primeira rede, Capítulo 2, trata da definição da máquina universal, feita por Turing em 1936, na qual a propõe como instrumento para abordar o problema da decidibilidade na matemática. A partir disso a ideia de máquina de Turing é retomada pela lógica e desenvolvida como classe de autômatos infinitos. Diferentemente do que uma versão corrente da história da computação narra, Turing não foi um dos inventores do computador. Os primeiros computadores foram criados de modo completamente independente da noção de máquina universal. Levou mais de duas décadas para que a máquina universal viesse a ser o modelo abstrato do computador, o que aconteceu como um dos modos da ciência da computação estabelecer-se como ciência, colocando retroativamente Turing como um de seus fundadores e criadores. É este movimento de construção da máquina universal como ideal da computação que será elaborado. Partimos de sua contextualização na história da matemática e lógica, apresentando de modo detalhado como Turing a constrói em seu artigo de 1936, e como a noção de "máquina" será introduzida no discursos da matemática e lógica. Segue uma discussão sobre como a ideia de máquina universal foi recebida e posteriormente modificada e incorporada pelas teorias de autômatos infinitos e linguagens formais, transformando-se no modelo abstrato do computador eletrônico digital.

Após o surgimento dos computadores e de sua formulação como máquinas universais, a segunda rede, Capítulo 3, trabalha o modo como a noção de máquina universal foi transportada para o domínio das linguagens de programação, criando linguagens de "alto nível". Os primeiros computadores possuíam, cada um, seu modo próprio de programar e, posteriormente, sua própria linguagem de programação. Ainda que fossem, abstratamente, máquinas universais, seu código não poderia ser reutilizado em outras máquinas sem uma grande quantidade de trabalho, e quando tal conversão era possível. Ao transportar a noção de universalidade às linguagens de programação, a máquina universal deixa de corresponder a máquinas específicas e cria um domínio de código que permite, dentro de certos limites, a

livre passagem de programas de uma máquina a outra. O foco desta rede encontra-se nas controvérsias que envolveram a constituição de tais linguagens, colocando de um lado aqueles que defendiam linguagens específicas na medida em que permitiam um melhor desempenho da máquina, e do outro lado os proponentes destas linguagens gerais e abstratas, ao custo da eficiência e desempenho. Por fim, ao abordar o surgimento do movimento do *software* livre veremos como a noção de universalidade não será mais restrita à execução de um programa, incluindo também a possibilidade de compartilhá-lo além de acessar e modificar seu código. A formação deste novo coletivo envolverá ao mesmo tempo atores jurídicos, pela criação de licenças de *copyleft*, novas formas de organização para a criação de *softwares* coletivos, e o desenvolvimento um sistema operacional livre.

1.2.2 Dobras

A rede define um sistema intrincado de transportes e traduções, onde atores e ações são deslocados, substituídos, aproximados e afastados. Tais relações constituem programas de ações, derivas e uma historicidade que lhes é própria. O princípio de simetria lhes dá uma grande amplitude, pois permite conectar tanto os saberes apagados quanto atores não-humanos. Fundamentalmente, na rede, tal como no rizoma, encontramos uma multiplicidade, mas não uma totalidade. Multiplicamos as dimensões, mas não encontramos um nível superior ou inferior que determine ou reduza os traçados que lhe compõem. Tudo em uma rede encontra-se no "mesmo nível".

Os mesmos princípios se aplicam à dobra. O que a diferencia da rede não concerne tanto à estrutura lisa do espaço que ocupam, quanto ao modo e natureza das relações estabelecidas. Podemos dizer que a rede adota um modelo conexionista, enquanto a dobra é *topológica*.

> Se você apanha um lenço e o estende para passá-lo, você pode definir sobre ele distâncias e proximidades fixas. Em torno de um pequeno círculo que você desenha próximo a um lugar, você pode marcar pontos próximos e medir, pelo contrário, distâncias longínquas. Tome em seguida o mesmo lenço e amasse-o, pondo-o em seu bolso: dois pontos bem distantes se vêem repentinamente lado a lado, até mesmo superpostos; e se, além disso, você o rasgar em certos lugares, dois pontos próximos podem se afastar bastante. Denomina-se topologia a essa ciência das proximidades e dos rasgos, e geometria métrica à ciência das distâncias bem definidas e estáveis (SERRES, 1999, p. 82).

A verdadeira oposição encontra-se não entre redes e dobras, mas entre topologia e geometria.

A geometria euclidiana constitui um espaço métrico, cujas medidas são absolutas e que mantém uma relação estável entre todos os seus pontos, o que permite que cada posição seja medida, possuindo uma localização precisa em um plano cartesiano. As formas e figuras

presentes em um espaço métrico podem ser medidas e localizadas em relação a este sistema de coordenadas total. A topologia, por sua vez, toma o espaço plano da geometria euclidiana e o dobra, do mesmo modo que o lenço é amassado. O problema colocado não é mais o de localizar ou medir figuras a partir de medidas que lhe são externas, mas de como as figuras constituem o próprio espaço ao dobrá-lo. Se na geometria euclidiana uma esfera é traçada *dentro* de um espaço tridimensional, na topologia ela *é o próprio espaço*. Não há nenhum espaço absoluto fora da figura, nenhum sistema de coordenadas que lhe contenha. É neste sentido que, como aponta Smith (2012), Deleuze e Guattari concebem a ausência de totalidade no rizoma, seu *n-1*. Assim como na geometria, uma figura topológica pode ter *n* dimensões, contudo, falta-lhe um espaço externo, um sistema de medidas absolutas, uma totalidade. Por isso o rizoma, por ser topológico e excluir a totalidade, é não-métrico. O princípio de conexão do rizoma, que permite que cada ponto seja conectado a qualquer outro, constitui o procedimento fundamental da dobra topológica.

O problema da topologia não é o de medir dimensões ou descrever propriedades de figuras no espaço, mas o que acontece ao próprio espaço quando é dobrado, torcido e modificado. Em outras palavras, o que muda e o que permanece? Imagine uma corda, cujas pontas se encontram e formam um círculo. Nesta corda são coladas etiquetas numeradas, de modo decrescente. Começando do primeiro número, e seguindo a circunferência do círculo formado pela corda, é possível chegar ao último número e a partir dai retornar ao primeiro. O que acontece quando a corda é dobrada e embaralhada (tomando o cuidado de não rompê-la nem de conectar suas partes)? Vemos que mesmo na forma mais emaranhada e caótica possível, encontramos dois invariantes: a nova figura ainda permanece ilimitada, isto é, tal como um círculo não possui um começou ou fim; do mesmo modo, a ordem dos números é mantida. Se uma formiga fosse colocada no primeiro número, ela seguiria sem problemas até o último. O que a nova figura perdeu foi a forma antiga, a propriedade de ser redonda, assim como as medidas. Entre um número e outro a distância anterior foi alterada e deformada (BARR, 1964).

Dobrar o pensamento, dobrar a máquina universal, constitui uma experimentação filosófica em que deformamos, aproximamos e distanciamos conceitos e práticas no intuito de saber o que permanece e o que se transforma. Em especial, buscamos *singularidades*, os pontos de virada do sistema, aquele momento crítico em que tudo muda, como a catástrofe que desencadeia uma avalanche. A dobra é um tipo de singularidade. (ARNOLD, 1992). Na física, e também na matemática, a singularidade é como um pequeno germe, cuja presença leva o sistema a um novo estágio de equilíbrio. Quando a água é resfriada, em determinado

momento forma-se o primeiro cristal de gelo, que passa a organizar o restante das moléculas de água em formas similares à sua (SIMONDON, 2009). Nosso procedimento consistente, então, em dobrar, em semear singularidades no intuito de ver quais reações são produzidas, quais efeitos são propagados e como os invariante são agenciados.

É um procedimento similar ao que Deleuze (1992) propõe em relação à leitura da filosofia, no qual deve-se tomar as palavras de um filósofo, um intercessor, e com elas criar um filho monstruoso, algo que poderia ser dito mas não o foi. Em certa medida, é fazer como ele. "Fazer como ele não é, necessariamente, ser seu discípulo. Fazer como ele é prolongar sua tarefa, é criar conceitos que têm relação com o que eles criou e colocar problemas em relação e em evolução com o que eles criou" (DELEUZE; PARNET, 1988, p. 49). Na dobragem filosófica o problema constitui um invariante, onde conceitos, intercessores e singularidades são torcidos. O problema é um virtual, que a cada dobra é atualizado de modo distinto. A fidelidade do filósofo ao intercessor que se busca dobrar não é tanto ao nível das palavras, da linguagem ou de seus conceitos, mas das forças que o animam.

Tudo isto que dissemos em relação às dobras, aplica-se igualmente às redes. Seria mais correto dizer que as redes constituem um tipo de dobra, são um caso mais específico. Em ambos os casos criamos aproximações que não estavam lá originalmente, seguimos problemas e buscamos ser fiéis aos atores, dobrando a relação temporal em maior ou menor medida. Nas redes, contudo, há uma preocupação maior em manter a linguagem dos atores e seus problemas nos seus próprios termos, e os conceitos funcionariam como uma infralinguagem, como um mecanismo de mediação que ligaria a rede, conservando ainda uma certa ordenação temporal. Com as dobras, o foco encontra-se mais nos problemas e conceitos do que nos atores. Este toram-se intercessores. No primeiro caso, conceitos ligam atores, problemas e práticas, no segundo, intercessores mediam problemas e conceitos. É uma transformação sútil, mas que produz efeitos muito diversos.

Deste modo, propomos a realização de três dobras, onde recolocamos problemas ligados à máquina universal aproximando-os de novos grupos de conceitos. A primeira dobra, Capítulo 4, aborda textos menores de Turing, que propõem diversos outros tipos de máquina, em especial a máquina desorganizada, mas que permaneceram em grande parte desconhecidos pois não constituíram uma rede. Um elemento recorrente nestes textos é a insistência de Turing em não definir o que é a inteligência e o pensamento, ao mesmo tempo em que propõe os mais variados modelos de máquinas pensantes e testes para decidir se uma máquina é inteligente ou não. Operamos uma dobra ao ligar tais conceitos e posições à noção de imitação de Tarde e "devir todo mundo", proposto por Deleuze e Guattari, colocando a

máquina como pensante apenas na medida em que o pensamento não é definido, abrindo a possibilidade de devir um pensamento não-humano para além da máquina universal.

No Capítulo 5, a segunda dobra, é retomada a potência da transmissibilidade presente nas linguagens de programação gerais, e as recolocamos no contexto de uma materialidade. Partindo da monadologia de Leibniz, e de suas correspondências com o modelo clássico da máquina de Turing, discutimos como a teoria da computação ainda conserva uma visão idealizada e determinística do computador e de seus processos, completamente desconectado de suas condições materiais e relações com o mundo. Com Tarde e Whitehead, colocamos a abertura das mônadas como modelo de pensamento de um código que existe diretamente no mundo, que circula e faz circular, situando a universalidade diretamente entre sujeitos e objetos.

Finalizando, a terceira dobra, Capítulo 6, terá como foco os modos de subjetivação e sua relação com a máquina universal. Abordaremos a noção de *daemon*, tanto em seu sentido computacional, como programa que é executado sem a intervenção do usuário, quanto em seu sentido grego original, para pensar como a máquina universal se compõe com uma cognição estendida ou distribuída, e os problemas políticos e éticos que dai advém, assim como a sua relação com a prática do corpo sem órgãos trabalhada por Deleuze e Guattari.

Parte 1: Redes

2 Máquina Universal

Sem economizar os mais sedutores comentários, Canterel atraiu nossa atenção para os diversos órgãos do aparelho.

- Raymond Roussel

2.1 "Turing não inventou o computador"

O ano de 2012 marcou o centenário de nascimento de Alan Mathison Turing, acontecimento celebrado com homenagens e eventos pelo mundo inteiro, tendo como foco as contribuições de Turing para o desenvolvimento da computação, inteligência artificial e criptografia. Nestas comemorações Turing recorrentemente aparecia como um dos pioneiros da computação, um dos inventores do computador, senão *O Inventor*, pois fora o criador, em 1936, do modelo abstrato do computador, a *máquina universal*.

Como conta uma versão tradicional da história da computação (DAVIS, 2000), o nascimento do computador eletrônico de propósito geral se deu em 1946 com a finalização da construção do ENIAC (*Electronic Numerical Integrator and Computer*), sob coordenação de John Mauchly e J. Presper Eckert, tendo como finalidade a realização de cálculos para o desenvolvimento de armas, como a bomba de hidrogênio. Considerado como o primeiro computador eletrônico construído, o ENIAC foi, contudo, precedido por um certo número de projetos e ideias. Podemos voltar um século antes, à década de 1840, quando Charles Babbage teria começado a projetar sua "máquina analítica", dispositivo extremamente complexo que utilizaria engrenagens e cartões perfurados para realizar qualquer tipo de cálculo matemático, tendo a possibilidade, como Ada Lovelace apontou, de manipular símbolos e até escrever músicas. A máquina analítica, todavia, nunca foi construída.

No que concerne às ideias, podemos voltar ainda mais no tempo, até o século XVII, quando Leibniz concebeu um cálculo universal que, utilizando uma linguagem lógica estrita e um dicionário ou enciclopédia, poderia traduzir qualquer tipo de em argumento em uma fórmula que produziria conclusões exatas sem ambiguidade. Em outras palavras, seria possível desenvolver uma máquina de calcular não apenas números, mas também ideias, permitindo um desenvolvimento completamente lógico e ordenado do pensamento. Apesar de

nunca ter levado a frente seu projeto, este foi retomado e realizado, em parte, pela lógica moderna, especialmente por Boole e Frege. (DAVIS, 2000)

Nesta leitura da história da computação teríamos um desenvolvimento linear e progressivo que iria de Leibniz a Turing, incluindo as contribuições da lógica e da matemática (passando por Boole, Cantor, Frege, Gödel e Hilbert, mas omitindo Babbage). Enquanto as concepções anteriores de uma linguagem ou cálculo universal obtiveram um sucesso apenas parcial, foi com Turing que o "Sonho de Leibniz" pode ser finalmente realizado, pois teria criado o primeiro modelo de um computador universal, de uma máquina capaz de executar, ou calcular, qualquer máquina computável. Ainda que não fosse possível criar um sistema tão absoluto e perfeito quanto o desejava Leibniz, era possível ao menos desenvolver uma máquina que seria universal dentro das limitações da computabilidade e da lógica. Deste modo, a formulação da ideia da máquina universal, que encontrava-se apenas de forma implícita na máquina analítica de Babbage, permitiu que o computador eletrônico moderno fosse construído, nos levando à revolução computacional ou informática. A proeminência de Turing em relação a Babbage nesta história deu-se pelo fato do primeiro ter desenvolvido explicitamente a ideia de universalidade assim como seus limites e consequências, enquanto nos projetos do segundo ela estaria implícita.

Apesar de sua aparente evidência, esta história pode ser problematizada de diversas formas. Primeiro, é uma história de origens, que busca começos absolutos que definiriam, já em em seu nascimento, o caminho futuro a ser seguido ou, no mínimo, o *melhor* caminho, aquele *deveria* ser realizado. O que nos leva ao segundo problema, diretamente ligado ao primeiro: é uma história negativa, que considera os acontecimentos na medida em que correspondem em maior ou menor medida a um ideal ou critério pré-estabelecido. No que se refere à história do conhecimento, este ideal é a razão ou, mais precisamente, a verdade. Todo saber passado é julgado nos termos do conhecimento contemporâneo, daquilo que é tomado como verdade neste momento, seja como "A Verdade", ou como um conhecimento mais "avançado" ou "desenvolvido". Os saberes considerados falsos são esquecidos ou mantidos a título de exemplo, como caminho a ser evitado ou episódio do triunfo do pensamento científico e racional. Encontra-se implícita nesta perspectiva a noção de que a prática é precedida pela teoria enquanto pensamento abstrato, uma vez que o computador eletrônico encontraria sua possibilidade de existência real apenas com a formulação teórica que lhe precedera.

Esta perspectiva acerca de invenção do computador, e do papel que Turing desempenhou, atualmente é questionada por diversos historiadores da computação (AGAR,

2003; BULLYNCK; DAYLIGHT; DE MOL, 2015; DAYLIGHT, 2012; HAIGH, 2014; MAHONEY, 2011; PRIESTLEY, 2011), e como Haigh (2014) diz já no título de seu texto: "Na verdade, Turing não inventou o computador". Turing desenvolve a ideia de máquina universal em 1936, dez anos antes da construção do ENIAC. Em nenhum momento deste período a ideia de máquina universal foi utilizada como base ou modelo para a construção de computadores eletrônicos. O próprio termo "computador", tal como aparece no artigo de Turing, possui um sentido no máximo ambíguo, referindo-se mais ao sentido original da palavra, *computadores humanos*, pessoas cuja tarefa era realizar cálculos, do que necessariamente a uma máquina, à qual reservava os nomes "máquina de computar" ou "máquina de calcular".

Ao invés da máquina universal sair de seu domínio lógico-matemático para servir de modelo a mecanismos concretos, o inverso aconteceu. Em diversos textos matemáticos e lógicos escritos a partir do final da década de 40, as máquinas de Turing eram definidas como sendo um certo tipo de computador, entendendo esta palavra em seu sentido atual. Foi construção dos primeiros computadores eletrônicos que deu um modelo explicativo ou demonstrativo para as máquinas de Turing e não o inverso. O artigo de 1936 teve pouca repercussão, sendo pouco lido ou compreendido, tanto na comunidade de lógicos e matemáticos quanto de engenheiros (DAYLIGHT, 2012). A ideia da máquina de Turing ou máquina universal, obteve um alcance e circulação muito limitados inicialmente. Como Priestley (2011) aponta, a ideia do computador como máquina universal começou a ser discutida entre um público mais amplo apenas na década de 50, após a publicação do artigo "*Computing Machinery and Intelligence*" (TURING, 2004 [1950]), texto que obteve grande circulação e tornou-se conhecido por ser a fonte do "Teste de Turing", além de propor a ideia do computador digital como uma máquina universal.

Do mesmo modo que Turing não foi um dos inventores do computador, também não esteve envolvido no desenvolvimento da ciência da computação.

> Turing não foi, em nenhum sentido literal, um dos construtores da nova disciplina. Não se envolveu com a ACM[7] ou qualquer outro grupo profissional, não fundou ou editou nenhuma revista, e não orientou as dissertações de grandes grupos de futuros cientistas da computação. Nunca construiu um laboratório, implementou um programa de graduação, nem ganhou grandes investimentos para desenvolver pesquisas na área. Seu nome não aparece como organizador dos primeiros simpósios para pesquisadores da computação, e ao tempo de sua morte seus interesses já haviam se distanciado das preocupações centrais da disciplina nascente.
> Quando uma casa é construída as fundações vêm primeiro. As fundações de uma nova disciplina são construídas tardiamente no processo. O artigo de 1936 de Turing foi escavado, por outros, da tradição da lógica e da matemática, sob a qual encontrava-se originalmente, e movido ao novo campo em desenvolvimento.

7Association for Computing Machinery, criada em 1947.

(HAIGH, 2014, p. 35)[iv]

O papel central que Turing ocupa em nossas narrativas acerca da invenção do computador, e de sua influência na constituição do campo da informática e das ciências da computação é um acontecimento retroativo. Por mais que o computador nos pareça como o modelo da exatidão mecânica, sendo em seu nível mais fundamental uma entidade matemática, tal aparência não se traduziu no estabelecimento direto e não problemático de uma disciplina científica. O que chamamos de "Ciências da Computação" é a confluência de pelo menos três vertentes de práticas e teorias: matemática/lógica, engenharia e ciências (MAHONEY, 2011). Desde o começo a formação do campo se deu nos conflitos e embates entre essas três linhas, cujos focos e interesses dividiam-se entre abordagens puramente teóricas e dedutivas, no caso da matemática, passando por problemas práticos implicados na construção de computadores, no caso de engenharia, chegando a sua aplicação em domínios e pesquisas científicas. O próprio nome da disciplina constituiu um motivo de disputa, assim como seu objeto ou status científico. Ciências estudam fenômenos ou leis naturais, então como o computador, artefato humano, poderia ser objeto de ciência? Do mesmo modo, noções abstratas como computabilidade, complexidade e informação não dariam conta de toda a miríade de campos de saber englobados pelas práticas em seus mais variados domínios de aplicação, que ampliavam-se progressivamente com o desenvolvimento de novas tecnologias, demandas e exigências econômicas, científicas e sociais, sendo que as disputas acerca da natureza de uma ciência da computação permanecem abertas ainda hoje (TEDRE, 2015).

Foi em meio a este movimento conflituoso de constituição do campo das ciências da computação que a figura de Turing como um dos pioneiros da computação começou a ser construída na década de 60, com a máquina de Turing tornando-se o modelo abstrato do computador. A criação de figuras exemplares é um recurso utilizado em diversos campos de saber, tanto como tentativa de unificação, quanto de criação de um modelo a ser seguido por seu praticantes, especialmente os recém-chegados. Foi o que aconteceu com Gauss no século XIX, que acabou tornando-se o modelo do matemático brilhante e polivalente (BULLYNCK; DAYLIGHT; DE MOL, 2015).

Estes movimentos nos dão uma dimensão de como Turing veio a se tornar uma figura tão central no campo da computação. Movimento que, em um primeiro momento, nos parece como sendo puramente político, dado que cria uma ficção com o objetivo de arregimentar e interessar o maior número de forças em favor da criação de um campo de teoria e prática específico às redes da computação. Por outro lado, podemos assumir que tal gesto nada mais é do que o reconhecimento da genialidade de Turing, a restituição tardia ao

local que deveria ter ocupado desde o começo.

A primeira explicação, política, nos coloca no domínio que Latour (2001) denomina "construtivismo social", o qual considera a atividade científica exclusivamente como uma prática social desprovida de qualquer conteúdo que lhe seja próprio. Ou, quando assume a existência de um núcleo que seja "propriamente científico", não o toca, criando um conjunto de explicações assimétricas, onde a linguagem utilizada para descrever os fatos científicos não é a mesma para suas implicações sociais e políticas. A ciência é uma construção social do começo ao fim, ou então deve ser considerada apenas em relação a seus aspectos sociais. Em ambos os casos a objetividade da ciência, a "natureza", seu conteúdo, nunca é abordado diretamente nem considerado em sua positividade própria. A segunda explicação, recai na tradicional história da ciência de que falamos acima, a qual assume um progresso da razão apoiado nas ações de grandes figuras, indivíduos geniais, que conteriam em si mesmos todos os desdobramentos possíveis das ciências que criam. A rede pela qual tais ciências são arduamente construídas é apagada, restando apenas sujeitos que encarnam em si todos os ideias da razão.

Agar (2003) afirma que "a melhor questão histórica a ser colocada não é "Computadores programáveis [*stored-program*] são máquinas universais de Turing", mas, "Por que computadores eletrônicos programáveis [*stored-program*] foram tomados como máquinas universais, de uso geral?" (p. 7)[v]. Como vimos, a explicação política, ou subjetivista, não dá conta de responder a tal questão. Se a máquina universal pode ser considerada como o modelo dos computadores eletrônicos e, consequentemente, objeto das ciências da computação, com Turing ocupando um local central, isto não se deu por uma escolha arbitrária nem por uma necessidade da razão. Se nosso objetivo é partir de uma perspectiva simétrica, que adota a mesma linguagem e o mesmo sistema de explicação para fatos naturais e sociais, devemos olhar para a *própria máquina*.

O que propomos neste capítulo é que a posição que Turing veio a ocupar se deu por conta do novo ator que inseriu no campo discursivo: a máquina. Como veremos, ao trabalhar com a ideia de uma máquina, ao invés de se apoiar exclusivamente em sistemas formais e fórmulas matemáticas, Turing abriu um campo ambíguo, que permitiu, por um lado, o transporte da máquina para o domínio lógico, assegurando sua tradução e convertibilidade à linguagem da matemática e da lógica, e por outro lado, que a tornou diretamente associável aos computadores eletrônicos que acabavam de ser construídos. Atualmente podemos criticar a ambiguidade dos termos utilizados por Turing e, como o faz Cleland (2007), mostrar como a máquina de Turing não é, de fato, uma máquina. Mas se isto nos é problemático agora, não o

foi em grande parte da história da computação. Mais do que ser problematizado, o *status* ambíguo da máquina de Turing constituiu sua grande força.

É uma história parcial da rede formada pela máquina universal de Turing que buscamos narrar neste capítulo. Não será nem uma história do computador eletrônico, de sua criação ou desenvolvimento, mas de como as ideias de máquina de Turing e máquina universal foram pensadas e desenvolvidas nos domínios da matemática, lógica, teoria de autômatos e linguagens formais. Mais precisamente, exploramos como a de máquina de Turing surgiu em meio à problemática da decidibilidade da matemática, junto a noções às quais foi considerada como equivalente, como o cálculo-λ (cálculo-lambda) e as funções recursivas gerais, e como destas se diferenciou. Posteriormente, foi tomada pela teoria de autômatos e linguagens formais, sendo retrabalhada para servir como um modelo mais fiel aos computadores eletrônicos, assim como a e de linguagens formais e gramáticas gerais.

2.2 Crises da matemática

2.2.1 Formalismo, lógica e intuicionismo

O século XIX constituiu um período de vasto desenvolvimento e revolução sem precedentes na matemática, mas foi, também, atravessado por profundas crises, advindas das mais diversas áreas – como o surgimento das geometrias não euclidianas ou da teoria dos conjuntos e seus paradoxos –, que colocaram os próprios fundamentos da matemática em questão. No caso das geometrias não-euclidianas, a noção de referência foi posta em dúvida, assim como toda a concepção de verdade que lhe era implícita.

Por mais de dois mil anos, a geometria euclidiana constituiu tanto um fundamento, quanto uma evidência para o pensamento matemático e filosófico. O procedimento axiomático utilizado por Euclides na exposição de sua geometria tornou-se o próprio modelo do pensamento dedutivo e matemático, no qual deve-se partir dos elementos mais simples e universais, derivando proposições verdadeiras e generalizáveis. A derivações e provas seguiriam uma racionalidade dedutiva estrita, lógica e necessária. Contudo, o quinto postulado de Euclides, que hoje conhecemos como o postulado das retas paralelas, sempre se apresentou como problemático para os matemáticos. Tal postulado afirmava que, dadas duas retas, se uma terceira reta cruzá-las, formando um ângulo de 90 graus com cada uma, as duas retas iniciais serão paralelas e nunca se cruzarão. Mesmo não contendo nenhuma inexatidão, contradição ou falsidade, a maneira como foi escrito o diferenciava perante os quatro

postulados que o precediam, era por demais complexo. Postulados e axiomas devem conter o mínimo de pressuposições possíveis, expressando apenas o necessário. Por mais de um milênio tentou-se reduzir o quinto postulado a outros postulados, ou reformulá-lo de um modo mais elementar (DELONG, 2004; KLINE, 1982).

As geometrias não-euclidianas surgem quando o significado dos termos "retas paralelas" é modificado. O que acontece se for concebida uma geometria que assume retas paralelas que possuem ângulos maiores ou menores que noventa graus? Foi essa a questão colocada, durante o século XIX, por Gauss, Lobachevski e Bolyai, considerados os criadores das geometrias não-euclidianas, desenvolvidas depois por Riemann, entre outros. Temos, então, geometrias cujas retas paralelas nunca se encontram, distanciando-se infinitamente, ou que encontram-se em um ponto determinado. O grande problema das geometrias não-euclidianas foi sua ausência de correspondência ao que é dado na experiência. Se as verdades da geometria euclidiana não são uma parte constitutiva da própria realidade, comporiam ao menos a estrutura de nossa subjetividade, os juízos sintéticos *a priori* que, de acordo com Kant, seriam a condição de possibilidade de nossa experiência. Afinal de contas, em nossa experiência, retas paralelas mantêm uma distância constante entre si, nem se aproximando, nem se distanciando, como uma estrada de duas pistas construída sem irregularidades. Quando não eram consideradas aberrantes, as geometrias não-euclidianas eram simplesmente irrelevantes, uma vez que não possuíam nenhuma aplicação científica ou prática. Não passavam de um mero exercício abstrato ou excentricidade da matemática pura (KLINE, 1982).

No entanto, o que permitia a essas novas geometrias modificar o quinto postulado de Euclides? Isso não as levaria necessariamente a algum tipo de contradição? Era o que muitos acreditavam. Da mesma maneira que o quinto postulado não podia ser reduzido a outros, observou-se que era *independente*, isto é, que poderia ser substituído por outros postulados similares sem acarretar na implosão do sistema. Portanto, o quinto postulado não era uma verdade auto-evidente, não retirava sua certeza da capacidade de representar a natureza ou a realidade de forma exata, mas a partir de uma escolha arbitrária. O que as geometrias não-euclidianas tonaram claro foi que os axiomas e postulados, tomados como os fundamentos da matemática, não são necessariamente verdades auto-evidentes, e sequer precisam ter alguma correspondência com a realidade. O fundamento da matemática não se encontrava mais na sua relação diretamente isomórfica com a natureza (DELONG, 2004; HOFSTADTER, 1999; KLINE, 1982).

Pode-se dizer, portanto, que todas essas geometrias, euclidiana e não-euclidianas são

verdadeiras, mas verdadeiras acerca do quê? É neste período que a matemática divide-se em três grandes correntes, cada uma propondo fundamentos para a matemática que dariam conta dos problemas da referência e verdade, dentre outros. A primeira corrente é a lógica. Em um primeiro momento, parece óbvia a ligação entre matemática e lógica, o que, contudo, nem sempre aconteceu. Durante boa parte da história da matemática ocidental, matemática e lógica eram disciplinas distintas, a primeira preocupada com números e suas operações, a segunda com as leis da argumentação verdadeira e silogismos. A lógica, tal como a concebemos atualmente, é em grande parte resultado de uma série de desenvolvimentos iniciados no século XIX, especialmente a partir da obra de Frege, e diretamente ligados às transformações sofridas na matemática. Se a realidade objetiva não pode mais servir de critério para o sistemas formais, como podem ser validados então?

A lógica tomou para si esse papel de fundamentação, já que a matemática não concernia apenas a números e sim a símbolos e regras para sua formação, esses procedimentos poderiam ser utilizados para abordar qualquer linguagem ou, pelo menos, qualquer linguagem artificial. Ao invés de apoiar-se a uma correspondência direta com a realidade, a matemática deveria ser colocada em bases lógicas que seriam a manifestação de leis universais e eternas, o que constituiria um certo platonismo. Deste modo, para Frege (2002) o pensamento não se refere a nenhum processo subjetivo, restringe-se exclusivamente em expressar verdades lógicas independentes de qualquer sujeito. A perspectiva lógica/platonista assume uma verdade externa a qualquer linguagem ou sujeito, ainda que seja abstrata e ideal (ROTMAN, 1993, 2000).

O intuicionismo, proposto por Brouwer, constitui a segunda corrente que busca dar novos fundamentos à matemática, e define-se em termos opostos à tradição lógica. Primeiro, rejeita a própria lógica e o princípio do terceiro excluído. Assume provas e demonstrações que sejam exclusivamente construtivas, descartando provas por contradição. O princípio do terceiro excluído garantia que caso uma afirmação não fosse verdadeira, seria automaticamente falsa, e vice-versa, não havendo uma terceira opção. Uma prova por contradição afirmaria a ideia contrária a aquilo que buscaria demonstrar, eventualmente chegando a uma contradição, uma falsidade. Logo, de acordo com o princípio do terceiro excluído, se a proposição contrária é falsa, a proposição que se deseja afirmar é verdadeira. Para o intuicionismo isto não se aplica, pois ao demonstrar que um número não é 2, não posso afirmar automaticamente que é 3, já que pode se encontrar em qualquer um dos valores infinitos no intervalo entre um número e outro (no caso dos números reais). A única forma de fazê-lo é através de uma prova construtiva, que mostre como o número é efetivamente

produzido. Em relação à referência, esta é colocada completamente do lado subjetivo. A verdade matemática depende da concordância entre intuições subjetivas e individuais, que compartilham a mesma estrutura subjetiva *a priori*, seguindo um modelo kantiano. Deste modo, a prática matemática pode ser concebida como se dando completamente desprovida de linguagem ou de qualquer forma de intersubjetividade, acontecendo unicamente na interioridade intuitiva dos indivíduos que a exercem e que concordam por compartilharem a mesma estrutura subjetiva *a priori* (ROTMAN, 1993, 2000).

Por sua vez, o formalismo é a terceira grande corrente do pensamento matemático, e na qual Turing se insere ao conceber a máquina universal. Enquanto a lógica assume que o referente da matemática são ideias universais, e o intuicionismo restringe-se à concordância de significados subjetivos, o formalismo abandona a ideia de referencia. Mais precisamente, assume que a matemática é uma linguagem composta por signos sem sentido, sistemas de regras que manipulam significantes sem significado. Hilbert, a figura central desta perspectiva, afirmava que a matemática não passava da manipulação de símbolos sem significado inscritos em papel. As verdades lógicas e o psicologismo intuicionista são substituídos pela pura materialidade do significante, por sua *forma*. A verdade da matemática depende, portanto, exclusivamente dos aspectos formais dos sistemas em questão. É de acordo com sua forma que as contradições, tautologias e teoremas devem ser considerados. A aritmética, por exemplo, não concerne a somas, subtrações ou multiplicações de coisas no mundo. Somar dois números implica unicamente em seguir regras mecanicamente até obter-se o resultado esperado, isto é, uma determinada marcação em uma folha de papel. Os símbolos formais podem vir a adquirir significado, mas esta é uma atividade posterior e externa à prática matemática nesta perspectiva. A atribuição de significado não influenciaria de modo algum a verdade ou validade de teoremas e sistemas formais, sendo mesmo uma fonte de erros e usos indevidos.

Como Rotman (2000) coloca, cada uma destas perspectivas corresponde, *grosso modo*, a três eixos do discurso e da linguagem: a referência (lógica/platonismo), o aspecto psicológico (intuicionismo) e o elemento formal (formalismo). Mais precisamente, a perspectiva platonista pode ser dividida entre as asserções matemáticas, consideradas como fatos (dotadas de uma referente externo, ainda que ideal), e os meios para obter tais verdades, que seria o domínio próprio da lógica. Neste sentido, lógica e formalismo apresentariam métodos muito próximos ou até mesmo intercambiáveis, pois buscariam construir uma linguagem formal desprovida de quaisquer pressuposições ou resultados que não pudessem ser derivados de sua lógica estrita. A linguagem da lógica seria um modo de tornar o

pensamento (no sentido de Frege) o mais claro possível, e completamente desprovido de ambiguidades. Ainda que discordem acerca do *quê* é afirmado, lógica e formalismo tornam-se aliados no modo *como* a verdade deve ser buscada, com ambos distanciando-se da corrente intuicionista.

2.2.2 Sistemas formais, lógicas

Em ambos os casos, lógica e formalismo tem como principal instrumento o *método formal axiomático*. A axiomática é um método milenar que, desde Euclides, foi recorrentemente considerado como sinônimo da prática ou do pensamento matemático, senão da própria razão. Mas, como foi visto, a geometria euclidiana enquanto sistema axiomático baseava-se em diversas pressuposições implícitas, que apoiavam-se na evidência da experiência de um sujeito que desenha figuras com régua e compasso, além de assumir axiomas e proposições que não haviam sido definidos previamente. O desenvolvimento dos sistemas formais axiomáticos se deu em direção a remover tudo aquilo que não pudesse ser definido e assumido explicitamente.

Um sistema formal, em sua concepção moderna, é composto por um alfabeto, conjunto de símbolos a serem utilizados (como “1”, “0”, “+”, “-”, “=“, etc.), regras para a combinação de símbolos, e axiomas, que são as proposições iniciais. Em um sistema formal moderno não há mais diferença entre postulados e axiomas, pois nenhum se apoia mais em auto-evidências ou noções comuns. Nesse sentido, a aritmética é um sistema formal para a combinação de símbolos numéricos, que parte de axiomas como “zero é um número”, “zero não é sucessor de nenhum número”, etc, e de regras de formação, como a subtração, adição, mas também regras sintáticas através das quais a proposição “ 5+ 2 3 =” não tem sentido, em contraste a “2+3=5”. O número não designa mais uma quantidade, torna-se uma relação de pertença a um conjunto. O conjunto 2, e aquele conjunto de maçãs são isomorfos, por isso podemos falar de duas maçãs. Mas quando trabalhamos, por exemplo, com a lógica booleana, os conjuntos 0 e 1 podem designar relações de verdade e falsidade e não quantidades de coisas. Tudo depende do significado posterior que á dado aos símbolos utilizados. A princípio, o símbolo “2” não se refere a nenhuma quantidade, é apenas um traço, uma inscrição sem sentido. O trabalho da matemática na perspectiva formalista consiste unicamente em derivar teoremas a partir de axiomas ou, no sentido inverso, provar teoremas, remetendo um teorema dado aos axiomas que lhe originaram, caso seja realmente um teorema, isto é, uma proposição

verdadeira (HOFSTADTER, 1999).

Pode-se conceber um sistema formal como um jogo. Fazendo uma analogia com o xadrez, o alfabeto é constituído pelas peças, aos axiomas definem suas posições iniciais e as regras de formação são as próprias regras do jogo que especificam os movimentos possíveis das peças. Em um sentido formal, as peças do xadrez não possuem nenhum sentido e não representam nada, podendo ser substituídas por qualquer outro tipo de significante material, desde que as regras, axiomas e funções do sistema sejam preservados. Derivar teoremas consistiria em mover as peças criando determinadas configurações no tabuleiro, enquanto as provas partiriam de uma posição atual e tentariam retraçá-la até a posição inicial do jogo. Deste modo um mesmo teorema pode possuir muitas provas. Mas, diferentemente do xadrez, onde há um final e condições de vitória ou empate, um sistema formal não chega necessariamente a um fim, podendo conter combinações infinitas de teoremas e provas (TRUDEAU, 1993).

Podemos dizer que a lógica moderna é composta por diversos sistemas formais, como a lógica proposicional, o cálculo de predicados ou lógica de primeira ordem e a teoria de conjuntos ou lógica da segunda ordem. Cada um destes sistemas utiliza convenções e regras específicas para formalizar determinados domínios do pensamento ou da ação humana, tendo como foco principal a criação de linguagens generalizáveis cuja verdade de suas afirmações derive tento de sua forma quanto da necessidade de leis lógicas universais.

A lógica proposicional constitui o primeiro tipo de sistema formal lógico. Como o nome implica, é uma linguagem que se aplica a proposições, a afirmações cuja correspondência a um referente terá um valor de verdadeiro ou falso, assim como as relações estabelecidas entre referentes. “Sócrates está vivo” é uma proposição, cujo valor será verdadeiro caso Sócrates esteja vivo e falso se estiver morto. Do mesmo modo, a proposição “Platão está morto”, terá seus valores de verdade e falsidade decididos de acordo com a condição mortal de Platão. Duas ou mais proposições podem ser conjugadas utilizando conectores lógicos, como “E” e “OU”. Assim, “Sócrates está vivo E Platão está morto” será verdadeiro apenas se o primeiro estiver vivo e o segundo morto. Contudo, a lógica, ou cálculo proposicional, não se limita à aplicação de valores de verdade a frases de uso corrente ou enunciadas em uma linguagem natural, mas as converte em uma notação que torna evidente suas relações lógicas e formais.

As proposições “Sócrates está vivo” e “Platão está morto”, tornam-se as varáveis *p* e *q*, e os conectores lógicos “E” e “OU”, transformam-se em “**^**” e “*v*”. Deste modo, “Sócrates está vivo E Platão está morto” converte-se em:

$p \wedge q$

Como procedimento para decidir a verdade de tais proposições, a lógica proposicional utiliza *tabelas de verdade* que contém todas as combinações possíveis dos valores de verdade das variáveis em questão. Abaixo temos a tabela de verdade das duas proposições em questão.

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

O termos "E", "OU" são funções de verdade. O "E" é chamado de *conexão*, e produz um valor verdadeiro apenas se todos os seus termos forem verdadeiros, enquanto o "OU" é uma *disjunção*, cuja verdade depende de que seus valores não sejam falsos. Poderíamos construir a tabela de verdade para "Sócrates está vivo OU Platão está morto" da seguinte maneira:

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

A diferença entre os dois termos é a mesma que podemos encontrar em dois anúncios de emprego onde um pede que o candidato seja fluente em "inglês E francês" ($p \wedge q$), e o outro que seja fluente em "inglês OU francês" ($p \vee q$). As tabelas de verdade não precisam se limitar a dois termos, podendo conter quaisquer variações possíveis entre proposições e funções de verdade. Portanto, estas relações e tabelas de verdade se aplicam a qualquer proposição cuja forma possa ser colocada em um formato lógico, não se limitando a um determinado tipo de conteúdo. Podemos, inclusive, implementá-las utilizando circuitos elétricos, como uma lampada que liga apenas se dois interruptores forem ligados ($p \wedge q$), ou que liga caso pelo menos um dos dois interruptores for ativado ($p \vee q$). Milhões destes circuitos lógicos compõem um computador no nível mais fundamental de seu *hardware*.

Ainda que a lógica proposicional seja aplicável a um domínio extenso de proposições, não pode dar conta dos silogismos mais simples, como "Sócrates é um homem.

Todo homem é mortal. Sócrates é mortal". A lógica proposicional pode ser aplicada e decidir a verdade de proposições específicas, utilizando tabelas de verdade, mas é desprovida de expressões como "todo", "qualquer", "pelo menos um", que lhe permitiram ser generalizada para além de casos específicos.

O cálculo de predicados, ou lógica de primeira ordem, amplia a lógica proposicional ao introduzir *predicados*, ou *funções proposicionais*, e *quantificadores*. Antes de mais nada, um predicado restringe a validade das afirmações a um domínio específico, como os números naturais, ou o conjunto formado por pessoas canhotas. O predicado tem a forma de uma função que aceita um ou mais argumentos e cujos valores produzidos limitam-se a verdadeiro ou falso. No domínio dos números naturais, o predicado *Primo*(x) será verdadeiro caso a variável *x* seja substituída por 7, *Primo*(7) ou falso quando for *Primo*(9). Quando um predicado possui um argumento, isto é, quando sua variável assume um determinado valor, o predicado torna-se uma proposição. (PETZOLD, 2008)

A introdução de predicados, por si só, não altera a situação da lógica proposicional, pois ainda há a necessidade de substituir cada variável, dado que o predicado *Primo*(x) não nos diz nada sem um argumento. Por isso a lógica de primeira ordem introduz os quantificadores "∀" e "∃", que aplicados ao predicado sobre números primeiros produzem as seguintes expressões:

$$(\exists x)Primo(x)$$

$$(\forall x)Primo(x)$$

A primeira expressão utiliza o "quantificador existencial", ∃, que pode ser lido como "existe pelo menos um", o que nos permite afirmar então que "existe pelo menos um número natural *x* tal que *x* é primo", o que é verdadeiro. O "quantificador universal", ∀, é lido "para todos", cuja utilização na segunda expressão afirma que "para todos os números naturais *x*, *x* é primo", ou ainda, "todos os números naturais são primos", o que é falso.

Fica claro que o poder generalizador e dedutivo da lógica de primeira ordem permite sua aplicação a classes e domínios extensos de objetos, sem a necessidade de operar sobre casos específicos. Com os quantificadores não é mais estritamente necessário o apelo a um conteúdo. É no interior deste contexto que a lógica constitui sistemas formais com o poder suficiente para dar conta de todo tipo de prova e dedução. Foi com base nesta lógica que Russell e Whitehead em seu monumental trabalho *Principia Mathematica*, publicado em 3

volumes a partir de 1910, buscaram colocar toda a matemática em bases lógicas, derivando todas as suas verdades a partir de uma axiomática.

2.2.3 Consistência, completude e decidibilidade

Na perspectiva formal, a verdade matemática consiste exclusivamente em um processo de derivação, sem precisar referir-se a uma realidade externa, o que levou Hilbert a afirmar que basta conceber um objeto matemático para que ele exista, ainda que de um modo exclusivamente formal (KLINE, 1982). O problema que se colocava era o de ter certeza de que os sistemas formais não apresentariam nenhuma contradição, sendo capazes de declarar todas as verdades possíveis dentro de seu escopo. Com esse propósito, Hilbert definiu um programa com quatro características que todo sistema formal deveria possuir:

1. Independência
2. Consistência
3. Completude
4. Decidibilidade

A independência afirma que um sistema formal não deve conter axiomas supérfluos, assim como um axioma não pode ser derivado de outro (já que um axioma é sempre o ponto de partida, não pode ser uma combinação). Um sistema formal consistente é aquele do qual não podem ser derivadas contradições a partir de seus axiomas. Com certeza, é a característica mais importante para um sistema formal. A contradição viola o princípio do terceiro excluído e permite que algo seja ao mesmo tempo verdadeiro e falso. Quando se chega a uma contradição, a conclusão lógica é que tudo é possível, logo, nada pode ser afirmado e o sistema formal implode. O Programa de Hilbert era marcado por um profundo otimismo e uma crença de que todos os problemas matemáticos poderiam ser conhecidos e resolvidos (HOFSTADTER, 1999; PETZOLD, 2008).

A completude refere-se à capacidade de derivar *todos* os teoremas – proposições verdadeiras – de um sistema formal a partir de seus axiomas. Um sistema incompleto seria como uma linguagem que apesar de reconhecer certas verdades, não pode afirmá-las, pelo menos não sem cair em contradição. A afirmação de uma verdade depende da possibilidade de prová-la, dado que um sistema incompleto pode expressar um teorema, mas sem prová-lo.

Estreitamente ligado à completude, encontra-se o problema da decidibilidade, isto é, de saber se existe um procedimento geral capaz de decidir se um problema matemático, com seu resultado formulado em termos de verdadeiro ou falso, pode ser respondido, isto é, se pode-se decidir entre sua verdade ou falsidade. Tal procedimento não dá a resposta, apenas nos diz se é possível responder a um problema em termos de verdadeiro ou falso. O problema de decidibilidade foi colocado por Hilbert primeiro em 1900, e retomado em 1928, vindo a ser a ser conhecido como "*Entscheidungsproblem*", literalmente traduzido como "problema de decisão". Com seu programa, Hilbert buscava uma matemática fundada em axiomas sólidos, livre de contradições, capaz de provar todas as suas verdades e decidir se qualquer problema tem uma solução. Até 1930 tal sonho ainda parecia possível (HOFSTADTER, 1999; PETZOLD, 2008).

Em ambos os projetos, de Russell e Hilbert, almejava-se a construção de um aparato lógico-matemático infalível, capaz de justificar-se completamente e resolver todos os problemas apresentados. Entretanto, a história da lógica e da matemática é muito mais complexa do que um simples progresso em direção a uma formalização e logicização totais, uma vez que todos os desenvolvimentos que foram aqui descritos nunca deixaram de ser acompanhados pelos paradoxos mais insidiosos e irredutíveis. O mais significativo é o Paradoxo de Russell, que pode ser descrito da seguinte maneira: os conjuntos "normais" são aqueles que não contêm a si mesmos. O conjunto dos homens não é ele mesmo um homem. Os conjuntos "anormais" contêm a si mesmos, como o conjunto de todos os pensamentos matemáticos, que é ele próprio um pensamento matemático. Agora, imagine um conjunto N, que é o conjunto de todos os conjuntos normais, os conjuntos que não contêm a si mesmos. É N um conjunto normal? Se for normal, quer dizer que não contém a si mesmo. Contudo, já que N é o conjunto dos conjuntos normais, sendo ele próprio normal, está contido em si mesmo também, o que o transforma em um conjunto anormal. N é normal e anormal ao mesmo tempo, eis o paradoxo. Em outras palavras, 1 é diferente de 1. (HOFSTADTER, 1999)

Russell tentou contornar esse grave problema criando uma teoria dos tipos, na qual um conjunto jamais referenciaria a si mesmo, podendo ser referenciado apenas por um conjunto de nível superior. No caso da matemática, a teoria dos tipos teve um custo alto, pois foi necessário abrir mão de toda noção de auto-referência e recursividade. Simultaneamente, permitiu traçar uma divisão clara entre matemática e metamatemática. Esta última, como o nome implica, é o discurso sobre a matemática. Escrever 1+1=2 é uma operação matemática, já "1+1=2" é um enunciado sobre uma operação (e ""1+1=2"" é o enunciado do enunciado...). Com essas precauções dois níveis de discurso jamais deveriam se misturar, assegurando ao

mesmo tempo a clareza e não contradição (HOFSTADTER, 1999).

Em 1931, Kurt Gödel publica seu famoso artigo "Sobre as Proposições Indecidíveis dos *Principia Mathematica* e Sistemas Correlatos", no qual prova que todo sistema formal consistente é necessariamente incompleto, colocando um limite claro à capacidade da lógica e da matemática de solucionar todos os seus problemas. O que Gödel fez foi reintroduzir a auto-referência no cerne da matemática, justamente naquele sistema onde ninguém poderia pensar que isso seria possível: a teoria dos números. Através de um procedimento complexo, Gödel fez com que números ocupassem o lugar de palavras, e colocou esses números junto a outros números, fazendo efetivamente com que números falassem de números, quebrando a distinção entre matemática e metamatemática. E o que os números disseram? "Esse teorema não pode ser provado." Se for tomado como falso, tal teorema é provável, portanto, por ser provável, é verdadeiro, resultando em uma contradição. Se tal teorema for verdadeiro, então não é capaz de ser provado. Isso implica em que existam proposições verdadeiras em um sistema formal que, no entanto, não podem ser provadas, isto é, que não podem ser derivadas a partir de axiomas e regras de formação. Tal sistema formal será necessariamente incompleto, já que pode *expressar* uma verdade, mas não pode *prová-la*. O que primeiramente se deu como um duro golpe ao sonho lógico de tudo conhecer, logo abriu caminho a um prolífico campo de estudo de funções recursivas, que é onde encontraremos Turing e também sua máquina universal. Cinco anos após o artigo de Gödel, em 1936, Turing, Church, Kleene e Post darão, cada um seu modo, provas que demonstram a impossibilidade de um método geral de decisão, e em que cujo cerne poderemos encontrar procedimentos recursivos similares ao de Gödel (HOFSTADTER, 1999).

2.3 *Entscheidungsproblem*: problema de decisão

Os problemas de decisão podem se definidos do seguinte modo:

> Considere uma dada classe infinita e contável de questões matemáticas ou lógicas, em que cada uma pode ter uma resposta em termos de "sim" ou "não".
> Há um método ou procedimento pelo qual podemos responder a qualquer uma das questões da classe em um número finito de passos?
> Mais precisamente, nos questionamos se, para a classe de questões dada, é possível descrever um procedimento, ou um conjunto de regras ou instruções listadas de uma vez por todas para serem utilizadas como se segue. Se (*após* o procedimento ter sido descrito) selecionarmos *qualquer* questão da classe, o procedimento nos dirá como realizar passos sucessivos, e após um número finito de passos teremos uma resposta para a questão selecionada. Ao realizar os passos, temos apenas de seguir as instruções mecanicamente, como robôs; nenhum *insight*, engenhosidade ou invenção são exigidos de nós. Após qualquer passo, se ainda não obtivermos a resposta, as

> instruções junto com a situação atual nos dirá o que fazer em seguida. As instruções nos permitirão reconhecer quando os passos chegam ao fim, e também ler da situação resultante a resposta à questão, "sim" ou "não" (KLEENE, 1967, p. 223).[vi]

O procedimento de decisão deve ser finito, pois ao contrário jamais seria possível saber quando termina ou não. Ao mesmo tempo, a classe de questões a que deve responder é infinita. Neste sentido, o procedimento de decisão não seria diferente de um sistema formal ou linguagem, que partindo de axiomas e regras finitas podem gerar combinações infinitas de resultados. Contudo, o problema de saber se há um procedimento de decisão não depende apenas da capacidade de produzir respostas infinitas. Cada questão da classe infinita deve ser efetivamente respondida.

Pode parecer uma tarefa impossível, pois como um procedimento finito poderia dar conta de toda a extensão de um domínio infinito de questões? Contudo, isto é possível para diversas classes de problemas, incluindo a lógica proposicional. As tabelas de verdade constituem um processo de decisão, pois ao prescrever como os termos de um proposição devem ser organizados em uma tabela, e como seus valores de verdade são calculados, define em passos finitos como obter o resultado final em termos de verdadeiro ou falso. E este procedimento único pode ser utilizado para decidir o valor de *qualquer* proposição da lógica proposicional. Ao mesmo tempo, há um número infinito de proposições possíveis, com infinitos termos, sendo que a adição de uma nova variável aumenta exponencialmente o tamanho da tabela e o tempo levado para determiná-la, o que torna a tarefa impossível na prática. Entretanto, isto foge ao escopo do problema, que em sua versão mais geral pergunta acerca da possibilidade lógica e não prática.

Haveria um procedimento de decisão para a lógica de primeira ordem? Esta era uma questão que ganhava cada vez mais importância no começo do século XX, dado o papel central que ocupava na lógica, encontrando-se na base dos sistemas formais que buscavam fundamentar toda a matemática. A expressão "problema de decisão", *Entscheidungsproblem*, referia-se primeiramente à questão da decidibilidade da lógica de primeira que ordem, formulada em 1928 por Hilbert e Ackermann nos seguintes termos: "o problema de decisão será resolvido quando conhecermos um procedimento com um número finito de operações que determina se qualquer expressão é valida ou satisfatória... O problema de decisão pode ser considerado o principal problema da matemática lógica" (HILBERT; ACKERMANN apud PETZOLD, 2008, p. 260).[vii] Uma expressão é "válida" ou "universalmente válida" caso todos os seus valores sejam verdadeiros, o que a torna uma tautologia, sendo "satisfatória" quando pelo menos um de seus valores finais é verdadeiro. O problema de decisão, tal como foi

formulado por Hilbert e Ackermann, não pretende definir um método para descobrir todas as respostas das expressões da lógica de primeira ordem. Quer, ao invés disso, um procedimento que nos permita saber se tais questões *possuem uma resposta*, isto é, se possuem algum valor de verdade ou não, independentemente de como tal valor seja obtido. O procedimento deve decidir se a expressão em questão possui uma resposta, em termos de verdadeiro ou falso, sendo uma expressão decidível, sendo indecidível caso contrário.

Se em 1928 uma resposta positiva ao *Entscheidungsproblem* ainda era considerada possível, em 1931 tal perspectiva tornava-se incerta, pois Gödel provou que existem proposições indecidíveis, que não podem ser provadas, em sistemas equivalentes ao do *Principia Mathematica*. Em 1936 tal incerteza transformou-se na impossibilidade de uma resposta, quando quatro artigos, escritos por Church, Kleene, Turing e Post foram publicados e que demonstravam, de modos distintos, que não á um procedimento geral de decisão para a lógica de primeira ordem.

Church (1936, 1965a), trabalhou com a noção de cálculo-λ (cálculo-lambda), enquanto Kleene (1936) abordou o problema através de funções recursivas gerais. Alonzo Church, professor em Princeton, e Stephen Kleene, seu aluno na época, desenvolveram no início dos anos 30 o que veio ser conhecido como "cálculo-λ", um sistema formal axiomático que trabalhava com a noção de função recursiva. O cálculo-λ, como todo sistema formal axiomático, especifica um modo de produzir sentenças bem definidas e formadas, que possuam algum sentido dentro do sistema. Estas sentenças bem formadas são chamadas de "fórmulas-λ" e seguem uma notação no estilo "*λfx.f(x)*"[8]. Um dos problemas que o cálculo-λ pretendia resolver era o fato de que uma função matemática poderia ser expressa como "*f(x)*", "*f(x)* = $x^2 + x + 2$", ou simplesmente "$x^2 + x + 2$". Tanto "*f(x)*", quanto "$x^2 + x + 2$" poderiam referir-se a um número ou a uma função, criando uma ambiguidade. No cálculo-λ, a função seria expressa como "*λx*[$x^2 + x + 2$]", onde "*λx*" designaria a própria função. Deste modo, a própria função seria incluída explicitamente em sua definição, desfazendo a ambiguidade e tornando-a, ao mesmo tempo, recursiva, uma vez que a função torna-se uma variável que pode ser manipulada. Isto fica ainda mais claro com a fórmula-λ "*λfx.f(x)*", onde a variável "x" e a função "f" estão expressas na própria definição da função, não permitindo a separação entre função e fórmula numérica. (KLEENE, 1981)

Deste modo, poderíamos expressar a sequência de números inteiros positivos 1, 2, 3, … com a seguinte sequência de fórmulas-λ:

8 Esta fórmula pode ser expressa também como λfx[f(x)] ou λf{λx[f(x)]}.

λfx.f(x), λfx.f(f(x)), λfx(f(f(fx))), ...

Se existir uma função *F(n)*, onde *n* = 1, 2, 3, …, e como resultado produzir a sequência de fórmulas-λ acima, podemos dizer que a sequência é definida-λ por F. Em outras palavras, a definibilidade-λ constitui o processo passo a passo pelo qual fórmulas-λ são produzidas, dado que o cálculo-λ, diferente do uso normal de funções, conserva e diferencia os elementos que o compõem, tornando explícito o caminho que foi tomado (KLEENE, 1981). Estas funções fazem parte da classe de funções recursivas gerais, que em 1936, Church (1936, 1965a), a partir de seu trabalho conjunto com Kleene, definiu sua equivalência com a definibilidade-λ e com a noção de "calculabilidade efetiva":

> Teorema XVI. Cada função recursiva de números inteiros positivos é definível-λ.
> Teorema XVII. Cada função definível-λ de números inteiros positivos é recursiva.
> […]
> Agora definiremos a noção, já discutida, de um função *efetivamente calculável* de números inteiros positivos ao identificá-la com a noção de função recursiva de números inteiros positivos (ou de função definível-λ de números inteiros positivos). Esta definição é pensada como justificada pelas considerações que seguem, na medida em que justificativas positivas podem ser obtidas para uma definição formal corresponder à uma noção intuitiva.
> Já foi mostrado que, para cada função de números inteiros positivos que é efetivamente calculável no sentido recém definido, existe um algoritmo para o cálculo de seus valores (CHURCH, 1965a, p. 99–100)

No trecho acima encontramos quatro tipos de atores: funções recursivas, funções definíveis-λ, calculabilidade efetiva e algoritmos. Primeiro é afirmada a equivalência entre os dois tipos de função, que podem ser identificadas à noção de calculabilidade efetiva. Quer dizer que todo o domínio de funções definíveis-λ ou recursivas é efetivamente calculável, o que implica na existência de um algoritmo, de um procedimento de decisão em passos finitos para o cálculo de seus valores. Esta afirmação veio a ser conhecida como "Tese de Church", sendo nomeada como tal pela primeira vez por Kleene (1965 [1943]), em 1943: "Tese I. Cada função efetivamente calculável (predicado efetivamente calculável) é recursiva geral" (KLEENE, 1965 [1943], p. 274).[viii]

A Tese de Church é uma tese e não um teorema pois não foi provada. Church, ao igualar o domínio das funções efetivamente calculáveis às funções definíveis-λ, tentou dar uma definição formal da calculabilidade efetiva, considerada como uma noção intuitiva, pois parte de um apelo a uma experiência ou compreensão comum acerca do que pode ser calculado. Deste modo, uma função é efetivamente calculável se qualquer pessoa puder executá-la, e se puder fazê-lo, será definível-λ. Em leituras futuras, o que a Tese de Church (ou Church-Turing) afirma é o limite do que pode ser computado ou calculado tanto por um

ser humano ou máquina, apesar de existirem diversas controvérsias acerca da extensão da ou objeto da Tese (CLELAND, 2004, 2007; COPELAND, 2007; DAVIS, 1982a; GOLDIN; WEGNER, 2008; HODGES, 2007). Inversamente, uma função que não seja definível-λ não possui um procedimento de decisão dentro do cálculo-λ, o que nos leva à refutação do Entscheindungsproblem.

Se a Tese de Church não pode ser provada, o mesmo não se aplica ao Entscheindungsproblem, que Church (1936, 1965a) demonstra ser insolúvel. Basta uma única prova negativa, que apresente um caso indecidível, para que a impossibilidade do problema de decisão seja demonstrada, o que Church faz ao mostrar como um determinado tipo de conversão não pode ser realizado no cálculo-λ, o que o coloca fora do domínio da definibilidade-λ. A força tanto da Tete de Church, quanto de sua prova negativa ao Entscheindungsproblem, situa-se no fao de que não se limita ao cálculo-λ, pois este é equivalente a qualquer sistema que incorpore porções de aritmética e que sejam ω-consistentes e, como prova posteriormente (CHURCH, 1965b), também vale para a lógica de primeiro grau.

Na definição da Tese de Church dada por Kleene (1965 [1943]), não há nenhuma referência à noção de definibilidade-λ, apenas à noção de funções recursivas gerais. No artigo de 1936, onde Kleene também demonstra que o *Entcheidungsproblem* é insolúvel, o faz como funções recursivas, sem utilizar o cálculo-λ. Na década de 30 o cálculo-λ não havia despertado o interesse da comunidade matemática, sendo que a noção de função recursiva geral, desenvolvida nos trabalhos de Gödel e Herbrand, era mais familiar e próxima ao modelo tradicional da matemática, razão pela qual Kleene a utilizou em seu artigo, assim como em sua obra posterior. Mas também em 1936, Church e Kleene publicaram artigos em que demonstravam a equivalência entre o cálculo-λ e a funções recursivas gerais. (KLEENE, 1981)

Temos em 1936 duas provas negativas ao Entscheidungsproblem, ambas utilizando sistemas formais axiomáticos distintos, mas equivalentes, e que permitiam afirmar, a título de tese, a extensão dos problemas matemáticos que poderiam ser resolvidos de modo algorítmico. Unindo-se a estes esforços temos artigos que Turing (2004 [1936], 1936) e Post (1936, 1965a) publicaram no mesmo ano e que demonstram, também, que não há um procedimento de decisão geral para a lógica de primeira ordem. Mas, diferentemente do trabalho de Church e Kleene, Turing e Post não utilizam o método formal axiomático, ainda que estejam completamente inseridos na problemática formal. Turing, que no momento da escrita não conhecia os trabalhos dos outros autores, parte da noção de "máquina", que

desenvolve ao abstrair o processo pelo qual uma pessoa, um computador humano, calcula a expressão decimal de um número, desenvolvendo a noção de computabilidade (que, como veremos, será associada à definibilidade-λ e às funções recursivas gerais). Post, que era familiar com os trabalhos de Church e Kleene, parte também da abstração de uma pessoa que realiza cálculos, construindo um modelo muito similar ao de Turing, mas sem incluir a noção de máquina. Realizaremos uma análise aprofundada do artigo de Turing, conectando-o ao trabalhos de Church, Kleene de Post, na perspectiva de uma rede formada pelos mais diversos atores, matemáticos ou não, onde a máquina, tomada como ator, ocupará uma posição móvel, seguindo um destino distinto de suas contrapartes lógico-matemáticas, por mais que sejam recorrentemente reconectadas e identificadas.

2.4 *On computable numbers*

2.4.1 Definição de máquina

O artigo "*On computable numbers, with an application to the* Entscheidungsproblem" foi publicado em novembro de 1936 no *Proceedings of the London Mathematical Society* (TURING, 1936). O artigo é composto por 11 seções e um apêndice. Nas primeiras três seções encontramos a apresentação, definição e exemplos de "máquinas de computar", onde são descritos vários tipos de máquinas, seu funcionamento e sua definição e demonstração em termos de tabelas de instruções. As seções 4 e 5 complementam as anteriores ao apresentarem a ideia de tabelas abreviadas, sequências enumeráveis e descrições padrão. A máquina universal é apresentada e construída em detalhes nas seções 6 e 7, o que culmina na seção 8 com a demonstração de máquinas que não podem ser computadas por uma máquina universal, o que já constitui o cerne do argumento contra o *Entscheidungsproblem*. As seções 9 e 10 são dedicadas aos limites, extensão e exemplos de classes números computáveis, além da justificação de sua definição através do uso de máquinas de computar. Por fim, na seção 11, as máquinas e suas tabelas de instruções são convertidas à notação da lógica de primeira ordem e que apoiando-se nos resultados na seção 8, são utilizadas para provar que não há um procedimento de decisão geral para a lógica de primeira ordem. O apêndice discute a equivalência entre as noções de computabilidade e calculabilidade efetiva, e foi adicionado quando Turing tomou conhecimento dos escritos de Church e Kleene, logo após ter terminado de escrever o artigo. Das 36 páginas que compõem a versão original do artigo, 20 páginas são dedicadas diretamente à construção e descrição das máquinas de

computar, com a abordagem mais aprofundada dos números computáveis ocupando 10 páginas, com as seis restantes dividindo-se entre as questões acerca de da decidibilidade e o apêndice.

Nesta observação estrutural e temática fica evidente que o foco do artigo encontra-se, primeiramente, na construção detalhada de máquinas, seguida da definição da classe de números computáveis, pontuado pelas implicações ao *Entscheidungsproblem*. A centralidade das máquinas deriva não apenas da posição e extensão ocupadas no texto, mas do papel que desempenham na composição dos números computáveis e de suas consequências para a decidibilidade, pois como é afirmado ao final do primeiro parágrafo, "de acordo com minha definição, um número é computável se seu decimal puder ser escrito por uma máquina" (TURING, 2004 [1936], p. 58).[ix] O que coloca o texto de Turing à parte de outros escritos lógico-matemáticos não é definição de uma nova classe de números, mas o *modo* como o faz, utilizando máquinas ao invés de construir sistemas formais, provas e demonstrações.

O que não quer dizer que o texto seja destituído de provas, formalizações ou notações matemáticas. Afinal de contas um dos objetivos de Turing, através da definição dos números computáveis, é provar a impossibilidade de um procedimento geral de decisão. Por que construir máquinas? Se tais máquinas podem ser utilizadas em provas quer dizer que podem, também, ser formalizadas. Por que não partir direto de um sistema formal, sem precisar de todo esse desvio por mecanismos complicados? Talvez porque, mesmo situando-se em uma discussão formalista, Turing afirme algo não-formal. Os números computáveis são definidos não por alguma propriedade intrínseca, mas pelo *procedimento* pelo qual podem ser obtidos, isto é, por meios finitos extrínsecos.

A máquina é o meio finito ao qual Turing refere-se, mas poderia muito bem utilizar um "computador", tomando esta palavra em seu sentido original, isto é, uma pessoa que realiza um cálculo. A justificativa dessa exigência de finitude encontra-se "no fato de que a memória humana é necessariamente limitada"[x] (TURING, 2004 [1936], p. 59). A necessidade de amparar uma definição matemática na limitação da memória humana é estranha à tradição formalista, para a qual a matemática deveria ser completamente desprovida de referência, assim como para uma perspectiva lógica/platonista que se preocupa com verdades universais independentes de qualquer psicologismo ou subjetivismo. Frege (2002), afirma que o pensamento refere-se às leis da verdade e não aos modos de funcionamento ou limitações da cognição humana. Para a lógica o pensamento deve se relacionar apenas com as verdades formais imutáveis, e não com sujeitos em suas ações concretas e limitadas.

Simultaneamente, as máquinas que Turing descreve ocupam o lugar desse computador

humano cuja memória é necessariamente limitada. Pode-se dizer, então, que as máquinas são meras formalizações do comportamento humano, ou até mesmo um comportamento humano idealizado. Como será visto, as máquinas descritas são expressas por "tabelas de instruções" que definem passo a passo o que a máquina deve fazer. Estas tabelas, contudo, não se utilizam da linguagem tradicional da lógica, nem da matemática. Possuem uma notação própria. Dentro do discurso matemático, Turing desenvolverá o que podemos chamar de "discurso algorítmico". É um discurso que se expressa primeiro por tabelas de instruções e posteriormente através de linguagens de programação, cuja formalização se dá pela conversão à notação da lógica. Todavia, a conversão *não é necessária*. Em outras palavras, o que aparece no uso de tabelas de instruções é a possibilidade de descrição de algoritmos sem a necessidade de uma formalização estritamente lógico-matemática, da mesma maneira que, atualmente, alguém pode aprender a programar ou escrever um software sem saber nada de matemática, especialmente em seu aspecto lógico e formal. O que Turing faz ao descrever o comportamento de máquinas, e humanos, através de tabelas de instruções é abrir a possibilidade de conceber todo um domínio de ação que não se reduz a um modelo lógico e formal estrito, mesmo que tenha sua própria formalização. [9]

Ao permitir que homem e máquina sejam igualmente descritos por um discurso algorítmico, Turing não define necessariamente a sua equivalência, isto é, que o homem seja uma máquina ou que a máquina seja um homem. O que se dá é uma ambiguidade ou, mais precisamente, uma zona de indiscernibilidade, onde *não importa* se o agente em questão é homem ou máquina. A maioria das controvérsias futuras sobre a máquina universal ou as máquinas de Turing se dará por conta dessa ambiguidade, onde inúmeros atores tentarão, cada um a seu modo, definir de uma vez por todas qual a relação existente entre homem e máquina, e o que Turing "realmente queria dizer". Mas, por enquanto, têm-se apenas máquinas ambíguas que neste capítulo serão descritas em seu modo de agência própria, enquanto no quarto capítulo desdobraremos algumas das implicações da relação entre humano e máquina no pensamento de Turing.

Uma máquina de computar é descrita do seguinte modo:

> Podemos comparar um homem no processo de computar um número real a uma máquina que é capaz apenas de um número finito de condições $q_1, q_2, \ldots q_r$, que serão chamadas "configurações-*m*". À máquina é suprida com uma "fita" (o análogo do papel), que corre através da máquina, e é dividida em seções (chamadas de "quadrantes") cada uma capaz de conter um "símbolo". A qualquer momento há apenas um quadrante, digamos que o *r*-ésimo, contendo o símbolo $\mathfrak{S}(r)$ que está "na máquina". Podemos chamar este quadrante de "quadrante lido". O "símbolo lido" é

9O artigo de Post (1965a) apresenta uma situação similar, que será discutida no terceiro capítulo.

> o único dos quais a máquina está, pode-se dizer, "diretamente ciente". Contudo, ao alterar sua configuração-*m* a máquina pode efetivamente lembrar de alguns dos símbolos que tenha "visto" (lido) previamente. O comportamento possível da máquina a qualquer momento é determinado pela configuração-*m* q_n e pelo símbolo lido $\mathfrak{S}(r)$. Este par $q_n\mathfrak{S}(r)$ será chamado de "configuração": portanto, a configuração determina o possível comportamento da máquina. Em algumas das configurações em que o quadrante lido está em branco (*i.e.* não contém nenhum símbolo) a máquina escreve um novo símbolo no quadrante lido: em outras configurações apaga o símbolo lido. A máquina pode também mudar o quadrante que está sendo lido, mas apenas ao deslocá-lo uma posição à direita ou à esquerda. Em adição a qualquer uma dessas operações, a configuração-*m* pode ser modificada. Alguns dos símbolos escritos formarão a sequência de figuras que são o decimal de um número real que está sendo computado. Os outros [símbolos] são apenas anotações para "auxiliar a memória". Apenas essas anotações poderão ser apagadas (TURING, 2004 [1936], p. 59–60). [xi]

Uma máquina, basicamente, manipula uma fita, deslocando-a para a direita e para esquerda, lendo, escrevendo e apagando símbolos de acordo com a configuração e com o símbolo que lê no momento, passo a passo. Esta máquina é conhecida também como "máquina de Turing". Contudo, é necessário tomar cuidado com esse termo, pois a máquina aqui apresentada tornou-se uma "máquina de Turing" apenas quando deixou de ser *de Turing*, isto é, quando foi tomada, modalizada e transformada por outros, quando foi recolocada em meio a diversos outros atores que não o próprio Turing e os termos em que a concebeu inicialmente. Por essa razão, nesta seção será utilizado apenas o termo "máquina", deixando para o próximo a discussão das implicações de sua transformação em "máquina de Turing".

A palavra máquina é utilizada em três sentidos. O primeiro, descritivo, foi recém exposto. No segundo sentido, a máquina é tomada como um tipo, uma classe ou categoria. Por sua vez, o terceiro sentido refere-se à máquina enquanto implementação, isto é, como instâncias individuais de uma classe, nomeadas utilizando letras maiúsculas e expressas por tabelas de instruções e outros tipos de notação (configurações completas, descrições padrão, etc).

Turing considera as máquinas, enquanto classes, a partir de três critérios: automação, computabilidade e circularidade. Uma máquina automática, máquina-*a*, é *completamente determinada* por sua configuração, pois cada comportamento procede necessariamente da configuração anterior, indo para a próxima configuração (ou parando, caso não haja nada mais a ser feito). Em uma máquina de escolha, máquina-*c*, apenas parte de seu comportamento é determinado por suas configurações, sendo necessária a intervenção de um operador externo para tomar uma decisão quando a máquina encontra-se perante um estado ambíguo. É importante notar que este é o único momento em que Turing faz referência a um usuário externo que manipula a máquina. O artigo trata apenas de máquinas automáticas, por isso

deve-se assumir que o termo máquina refere-se implicitamente a uma máquina-*a*. Mesmo ao limitar-se às máquinas automáticas, em nenhum outro momento fica claro qual a relação precisa entre uma máquina e o mundo externo. Por exemplo, a uma máquina é suprida uma fita que já contém determinados símbolos. Quem supre a fita? E as tabelas de instruções que contêm todas as configurações que a máquina deve seguir, situam-se aonde? No corpo da máquina, na mente do operador? Essa indeterminação é o que contribui para a ambiguidade criada entre homem e máquina, participando de controvérsias futuras, especialmente no campo da inteligência artificial e das ciências cognitivas.

Uma máquina-*a* será considerada uma máquina de computar se escrever dois tipos de símbolos. O primeiro tipo consiste exclusivamente de 0s e 1s. O segundo tipo pode ser composto por quaisquer outros símbolos que sejam definidos previamente. Como foi dito anteriormente, a escrita de símbolos na fita se dá de forma alternada. Em um quadrante são escritos os símbolos de primeiro tipo e no próximo os símbolos de segundo tipo, e assim sucessivamente. Os quadrantes que contêm os símbolos de primeiro tipo serão chamados de quadrantes-*F*, e os de segundo tipo, quadrantes-*E*, como na sequência abaixo:

F	*E*	*F*	*E*	*F*	*E*
0	x	1	x	1	y

Apenas os símbolos nos quadrantes-*E* podem ser apagados. A fita funciona como uma memória, e a necessidade da intercalação dos símbolos se dá porque entre os quadrantes-*F*, que constituem o resultado a ser obtido, é necessário que exista uma memória auxiliar para realizar cálculos e outras operações necessárias à composição da sequencia. Uma máquina de computar é aquela que apenas escreve 0 e 1 nos quadrantes-*F*. Uma *sequência computada* é composta pelos símbolos de primeiro tipo, enquanto o *número computado* é o número real cujo decimal é expresso pela sequência computada.

O terceiro tipo de máquina é definido por sua circularidade. Uma máquina será considerada circular se escrever apenas um numero finito de símbolos do primeiro tipo (0 e 1). Pode, portanto, continuar a escrever símbolos do segundo tipo (*x, y, a, b, c...*), mas será circular desde que não escreva mais nenhum de primeiro tipo, como uma máquina que continua funcionando após realizar a tarefa que lhe foi pedida, que escreve infinitamente o mesmo símbolo sem mudar de quadrante, ou que simplesmente para. Consequentemente, uma máquina não-circular continua escrevendo símbolos de primeiro tipo, em novos quadrantes-*F*, sem parar. É importante notar que apesar de Turing nunca dizê-lo explicitamente, a fita da

máquina tem uma extensão infinita, podendo operar por um tempo também infinito. Se a definição de computabilidade depende da finitude da memória humana, a utilização de uma máquina deveria conter esse aspecto. Ao implicar que a máquina possui uma fita infinita, isto é, uma memória ilimitada, retira-se, aparentemente, a limitação que dá sentido à computabilidade. No entanto, mesmo com uma fita infinita a máquina 1ida possui configurações *finitas*, isto é, pode ter apenas um número finito de configurações, símbolos e operações. A limitação que Turing atribui à memória humana é a de conter apenas um número limitado de estados e configurações enquanto realiza uma tarefa específica. A finitude refere-se então ao número possível de especificação de operações, devendo ser descrita em um número finito de passos ou instruções, e não ao tempo ou resultado de sua execução, sendo a infinitude da fita necessária para calcular, por exemplo, números irracionais cuja expressão decimal é infinita (PETZOLD, 2008).

As máquinas que Turing implementa são, portanto, máquinas automáticas, não circulares e que computam, isto é, máquinas cujas operações são determinadas por suas configurações, que escrevem apenas símbolos de primeiro tipo, 0 e 1, em quadrantes-*F* e o fazem sem parar. A primeira implementação de máquina, que neste trabalho será referida como máquina 1[10], é expressa pela tabela de instruções abaixo:

Configuração		*Comportamento*	
config-m.	*símbolo*	*operações*	*config-m. final*
𝔟	Nenhum	*P0, R*	𝔠
𝔠	Nenhum	*R*	𝔢
𝔢	Nenhum	*P1, R*	𝔣
𝔣	Nenhum	*R*	𝔟

Nesta tabela de instruções, encontram-se presentes todos os elementos descritos até agora. De um lado as configurações, do outro os comportamentos, determinados pelas configurações. A primeira linha da tabela deve ser lida da seguinte maneira: "se a máquina encontra-se na configuração-*m* 𝔟, e no quadrante lido não há nenhum símbolo, deve escrever o símbolo 0 (*print 0, P0*), deslocar a fita uma posição à direita (*right, R*) e seguir à configuração-*m* 𝔠". Seguindo as instruções a máquina escreverá nos quadrantes-*F* a sequencia 010101010101010... infinitamente, mantendo os quadrantes-*E* em branco. É uma máquina

10 Turing nunca dá um nome a esta e a próxima máquina. Ambas foram nomeadas neste trabalho para facilitar a exposição.

estritamente linear, no sentido de que uma linha da tabela segue à outra necessariamente, repetindo o ciclo. Logo a seguir, Turing apresenta uma máquina mais complexa, aqui referida como máquina 2, que escreve a sequência computável 01011011011110111110..., utilizando pela primeira vez os quadrantes-*E* para escrever o símbolo "*x*".[11]

config-m.	*símbolo*	*operações*	*config-m. final*
𝔟	Nenhum	*Pe, R, Pe, R, P0, R, R, P0, L, L*	𝔬
𝔬	1	*R, Px, L, L, L*	𝔬
	0		𝔮
𝔮	Qualquer (0 ou 1)	*R, R*	𝔮
	Nenhum	*P1, L*	𝔭
𝔭	x	*E, R*	𝔮
	e	*R*	𝔣
	Nenhum	*L, L*	𝔟
𝔣	Qualquer	*R, R*	𝔣
	Nenhum	*P0, L, L*	𝔬

A segunda linha, pode ser lida da seguinte maneira: "se a máquina encontra-se na configuração-*m* 𝔬 e o símbolo lido é 1, deve deslocar a fita uma posição à direita, escrever *x*, deslocar a fita três posições à direita e seguir à configuração-*m* 𝔬; caso o símbolo lido seja 0 deve prosseguir à configuração-*m* 𝔮". Entre cada símbolo 0 a máquina escreve uma sequência de símbolos 1, começando com apenas um e aumentando progressivamente. Em outras palavras, a máquina precisa ter algum mecanismo para "contar" e "saber" quantos números já escreveu, para isso utiliza o símbolo *x* nos quadrantes-*E*, como uma memória temporária que é apagada antes que a próxima sequencia de símbolos 1 seja escrita. A máquina 2 é mais complexa que a primeira, dado que cada configuração-*m* apresenta diversos estados possíveis de acordo com o símbolo presente.

Uma máquina não precisa ser necessariamente constituída por apenas uma tabela de instruções. Existem tabelas auxiliares, que Turing chama de "tabelas abreviadas" ou "tabelas-molde" (skeleton tables), compostas por uma série de processos compartilhados pela maioria das máquinas, que incluem "copiar sequências de símbolos, comparar sequências, apagar todos os símbolos de uma determinada fórmula, etc" (TURING, 2004 [1936], p. 63).[xii] Essa tabela funciona como uma biblioteca, à qual as outras máquinas podem referir-se, não sendo, contudo, necessária, já que pode sempre ser reimplementada por cada máquina, mas cuja

11 Uma versão funcional desta máquina pode ser vista em operação no endereço http://mecanosfera.net/turing/ .

utilização justifica-se por uma questão de brevidade e economia.

Por fim, uma máquina pode ser expressa através de sua "descrição padrão" (D.P), isto é, a série de instruções expressa em uma tabela é colocada como uma sequencia de símbolos alfabéticos. Por exemplo, tomando as duas primeiras instruções da máquina 1:

𝔟	Nenhum	*P0, R*	𝔠
𝔠	Nenhum	*R*	𝔢

As instruções acima podem ser normalizadas, sendo reescritas assumindo um conjunto de símbolos mais padronizado:

q_1	S_0	PS_1, R	q_2
q_2	S_0	PS_0, R	q_3

As duas instruções podem ser expressas como a sequência $q_1S_0S_1Rq_2$; $q_2S_0S_0Rq_3$;[12], com cada instrução separada por ponto e vírgula. A descrição padrão de uma sequência é obtida ao converter seus símbolos pelas letras "A", "C", "D", "L", "R","N". Obtêm-se então a seguinte descrição padrão da máquina 1, cujas duas primeiras sequências entre ponto e virgula correspondem às instruções exibidas previamente:

DADDCRDAA; DAADDRDAAA; DAADDCCRDAAAA; DAAAADRDA;[13]

A conversão em uma descrição padrão adota uma notação aparente mais complicada, que perde em clareza imediata mas que torna possível a uma máquina *ser computada por outra máquina*. Ao falar de máquinas, tal como são empregadas por Turing, é impossível saber se a referência é a um agente humano ou não, e tal incerteza não constitui um problema, pois a capacidade de um código ser mais facilmente compreendido, para um ser humano, não é tão importante quanto a possibilidade de ser computado, exige-se apenas a capacidade de manipulação mecânica de símbolos, seja por um computador humano ou mecânico. Possuindo a descrição padrão é preciso construir uma máquina capaz de computá-la, isto é, uma *máquina universal*.

12 Nessa notação S representa um símbolo, então S_0 expressa um símbolo vazio, enquanto S_1 e S_2 expressam 0 e 1 respectivamente. Se fossem utilizados mais símbolos eles seriam expressos por S_3, S_4 e assim por diante.

13 Uma descrição padrão pode muito bem ser expressa utilizando números, o chamado "número de descrição", mas que Turing apesar de mostrar não utiliza em nenhum momento em seu texto.

2.4.2 A máquina universal

Após a descrição das diversas máquinas propostas por Turing, temos todos os elementos necessários para construir uma *máquina universal*, definida da seguinte maneira:

> É possível inventar uma única máquina que pode ser utilizada para computar qualquer sequência computável. Se esta máquina *U* for suprida com uma fita em cujo começo encontra-se escrita a D.P. [descrição padrão] de alguma máquina de computar *M*, então *U* pode computar a mesma sequência que *M* (TURING, 2004 [1936], p. 68).[xiii]

A máquina universal, máquina *U*, é uma máquina como as que foram descritas até agora, mas que em sua fita contém a descrição padrão de outra máquina, máquina *M*. É uma máquina que contém outra máquina. Ou, mais precisamente, é uma máquina que pode conter *qualquer outra máquina computável*. É necessário construir uma tabela de instruções capaz de computar outras máquinas além da máquina *M*. Turing procede então à construção da tabela que compõe a máquina universal. Por ser muito grande e complexa colocamos no Anexo A a tabela de instruções completa da máquina universal, junto com a tabela abreviada que também utiliza.

O que, precisamente, significa "computar uma máquina"? Primeiro, implica em que a máquina universal escreva a mesma sequência de símbolos que a máquina *M* seria capaz. Se a máquina *M* for, por exemplo, a máquina 1, deve-se obter uma sequência computada composta pelos símbolos 0 e 1 alternados. A máquina universal deve ler o conteúdo da fita e realizar uma série de operações, listadas em sua tabela de instruções, para obter o resultado desejado. Diferentemente das máquinas descritas, à máquina universal é suprida uma fita que não se encontra em branco, ou seja, tanto a máquina a ser computada quanto sua sequência se dão sobre *a mesma fita*, no mesmo nível. Seria possível escrever uma máquina que ao ser computada por uma máquina universal reescreveria a si mesma, uma vez que tanto as instruções quanto a sequência escrita por esses instruções se dão ao mesmo nível. Pode-se obter uma máquina que ao escrever sobre sua descrição padrão seja "corrompida" e pare de funcionar, pois as instruções que contém não seguem mais nenhuma ordem. Ou, ao contrário, é possível que a máquina produza sequências que expressam instruções válidas capazes de produzir novas instruções e sequências computadas. No caso da máquina universal proposta por Turing a sequência computada é escrita após a descrição padrão presente na fita, evitando as complicações que uma máquina completamente auto-referente poderia gerar.

A universalidade da máquina depende de três elementos: 1. a tabela de instruções exterior à fita; 2. uma descrição padrão escrita sobre a fita; 3. uma sequência computável a ser

escrita. Tanto a descrição padrão quanto a sequência computável encontram-se "na memória", enquanto a máquina universal propriamente dita encontra-se na tabela de instruções que irá manipular essa memória. A grande diferença em relação às outras máquinas se dá pelo fato de colocar descrição e sequência computável no mesmo nível, isso permite que a máquina universal compute infinitas máquinas sem precisar modificar a si mesma. Por exemplo, um relógio executa em suas engrenagens as instruções necessárias para calcular e exibir as horas. Os ponteiros sobre os números constituem a sua escrita. Para mudar as funções de um relógio e, por exemplo, fazê-lo desempenhar o papel de uma calculadora, seria necessário alterar a ordem e distribuição de suas engrenagens, ou seja, modificar sua tabela de instruções. A máquina universal seria o equivalente às engrenagens, mas construída de tal forma que poderia escrever qualquer sequência computável, a partir de uma descrição que lhe fosse fornecida (por exemplo, mantendo os mesmos elementos de um relógio, ao lhe dar corda de maneiras específicas), sem precisar modificar a estrutura e modo de funcionamento de suas partes.

No artigo, não são utilizados termos como "*software*", "*hardware*", ou equivalentes, pois não é necessário fazer uma distinção entre uma tabela de instruções encravada no corpo da máquina, seu *hardware*, e a descrição padrão, seu *software*. Em nenhum momento são dadas instruções claras de como construir uma máquina ou qual a natureza exata da composição de suas partes. Turing, afinal de contas, não escreve um texto de engenharia neste momento, quer, ao invés disso, dar uma definição de números computáveis e provar a impossibilidade de um procedimento geral de decisão. Todo o maquinário que constrói é voltado à realização desses dois objetivos, apesar de não se limitarem a eles de modo algum, colocando mais problemas e possibilidades de controvérsias do que as que busca solucionar. Por isso Turing não explora aqui todas as possibilidades abertas por suas máquinas, em especial a máquina universal, mas as constrói para demonstrar a impossibilidade de existência de *outras máquinas*. Ao cercar-se de aliados, suas máquinas, Turing o faz estabelecendo uma oposição frente a máquinas que seriam capazes de computar *qualquer máquina*, inclusive as não computáveis.

A descrição padrão de uma máquina pode ser convertida em um "número de descrição", em que cada letra é substituída por um número. Compõe-se, então, uma lista contendo os números de descrição de todas as máquinas. Cada um desses números pode ser novamente convertido em sua descrição padrão e ser suprido a uma máquina universal. Muitos números não corresponderiam a nenhuma máquina válida, cujas instruções não seriam funcionais ou bem formadas. Descartando esses casos e limitando a máquina universal às

descrições válidas, como determinar se uma descrição corresponde a uma máquina circular ou não circular? “Vamos supor que exista tal processo; quer dizer, que podemos inventar uma máquina *D* que, quando suprida com a D.P [descrição padrão] de qualquer máquina computadora *M*, testará esta D.P e se *M* for circular marcará a D.P com o símbolo 'u' e se for não circular marcará com 's'” (TURING, 2004 [1936], p. 72–73).[xiv]

Unindo a máquina *D* com a máquina *U,* obtêm-se a máquina *K,* que pode ler a descrição padrão de outras máquinas e decidir se são circulares ou não, isto é, se escreve ou não um número finito de 0s e 1s. Eventualmente a máquina *K* chegará à sua própria descrição padrão, isto é, irá computar a si mesma. Esta é outra característica importante de uma máquina universal, a recursividade obtida ao colocar uma máquina a computar a si mesma. Entretanto, no caso da máquina *K* isto tem um efeito desastroso, pois entrará em uma recursão infinita. Tentará avaliar uma máquina que estará avaliando outra máquina avaliando outra máquina... e assim por diante. A máquina *K* nunca chega a uma decisão acerca de sua circularidade pois nunca para de computar a si mesma. Em outras palavras, não existe máquina capaz de decidir se outra máquina é não circular. Essa impossibilidade demonstra que não existe um procedimento geral de decisão, ou seja, uma única máquina capaz de computar e decidir o destino de qualquer outra máquina (PETZOLD, 2008). A máquina universal afirma-se na impossibilidade da existência das máquinas *D* e *K*, e com isto Turing dá a primeira solução negativa ao *Entscheidungsproblem*.

2.5 O acontecimento: máquinas, agentes, atores

O texto de Turing pode ser considerado como a realização de um *programa de ação*, entendido como “a série de objetivos, passos e intenções que um agente pode descrever numa história” (LATOUR, 2001, p. 205). Como primeiro objetivo temos a definição da classe dos números computáveis, com a solução negativa ao *Entscheidungsproblem* vindo em segundo lugar. Para atingir seu objetivo, Turing não pode fazê-lo sozinho, pois não basta que tenha uma opinião ou crença acerca da natureza dos números computáveis para que existam ou sejam verdadeiros. Uma prova ou demonstração matemática, assim como uma afirmação científica, é aceita apenas na medida em que convence outros matemáticos, em si mesma não possui nenhum valor. A realização do programa de ação depende da composição de um *coletivo*. Turing deve ser capaz de mobilizar e interessar os outros pesquisadores, mas, novamente, não pode fazê-lo sozinho. Antes de compor um coletivo com a comunidade matemática e lógica deve arregimentar e interessar aliados que lhe permitam construir uma

definição forte o suficiente para resistir às provas impostas por seus colegas como critério de entrada em sua comunidade.

O primeiro aliado que Turing convoca é o computador humano, dotado de memória limitada e que realiza computações utilizando lápis e papel. O computador é imediatamente substituído por uma máquina automática, máquina-a. Mais precisamente, os dois formam um novo agregado, uma mistura humano-máquina que os torna intercambiáveis durante grande parte do texto. Entretanto, a inserção da máquina se faz necessária pois diferentemente do computador humano, e do próprio Turing, ela possui uma fita/memória infinita, não comete erros e pode operar por toda a eternidade, sem precisar se preocupar com necessidades e anseios humanos. A associação ao computador humano permite que Turing capitalize o modo de ação de uma imensa classe de seres humanos: todos os que calculam com lápis e papel. A substituição do computador por uma máquina e sua correspondente associação amplia a força do sistema a ponto de lhe possibilitar dar conta da extensão infinita dos números computáveis. O coletivo Turing-Computador-Máquina torna-se mais forte.

Um coletivo não é composto apenas por humanos, não é nem um grupo nem uma sociedade em suas concepções tradicionais, constitui-se como um agenciamento de humanos e não-humanos, como fica claro nas associações operadas até agora. Equações, definições, provas, variáveis, máquinas, fitas, símbolos, tabelas, computadores são todos participantes ativos que compõem um coletivo, são o que podemos chamar de *atores* ou *actantes*. Todo ator é definido pela ação, é quem age, desde que entenda-se, por um lado, que a ação não é exclusiva ao ser humano e, por outro, que um ator "não é a fonte de um ato e sim o alvo móvel de amplo conjunto de entidades que enxameiam em sua direção" (LATOUR, 2012, p. 75). O ator é definido pelo modo que age, mas a sua ação já se encontra em um emaranhado de outros atores e ações, constitui-se apenas na medida em que compõe e tece associações. Isto implica em que "jamais fica claro quem ou o quê está atuando quando as pessoas atuam, pois o ator, no palco, nunca está sozinho ao atuar" (LATOUR, 2012, p. 75). Cada ator é uma fonte de incerteza, pois seu aparecimento necessariamente desloca e altera o modo de agir de outros atores, tal como um novo personagem que entra em cena. De acordo com as associações que forma e os híbridos que compõe, as incertezas podem ser potencializadas ou estabilizadas.

Mas como podemos dizer que uma equação age, que signos inscritos em um artigo são capazes de modificar ou deslocar outras entidades? A ação de uma máquina abstrata ou imaginária, como as máquinas de Turing, não seria nada mais que a ação daquele que as imagina. Temos dois problemas aí, um que concerne à natureza e modo de existência de

atores e actantes, e outro relativo à especificidade da matemática perante outras ciências no que tange à criação de atores.

O que comumente chamamos de "descoberta" científica pode ser considerado como um *acontecimento*, ou evento, no qual um novo ator é produzido. Começamos com listas de ações, descrições de comportamentos ou fenômenos: anomalias gravitacionais, substâncias esbranquiçadas acumuladas em um tubo de ensaio, disfunções motoras. São puros desempenhos ou *performances.* Algo age mas ainda não sabemos o quê, neste momento não temos nem sujeito nem objeto, apenas listas de ações, os actantes. Eventualmente um actante pode ganhar uma figuração, vindo a compor uma figura que as estabiliza e funciona como sua origem, isto é, compõe um ator. A diferença entre actante e ator é a mesma que, em uma narrativa fantástica, há entre a espada mágica e o herói. A espada age, é encantada, mas apenas na medida em que é empunhada pelo herói que a narrativa progride, que há uma resolução. O herói tem de apelar ao poder da espada, enquanto esta é integrada a uma figura que lhe permite ser personagem de uma história.

Quando um actante ganha uma figuração, a performance é referida a uma *competência*. Na linguística e na semiótica a competência designa a posse dos meios necessários para executar uma determinada ação. Neste sentido a competência precede a performance, tal como o conhecimento de uma língua é necessário para que seja falada, ou a existência de uma anomalia gravitacional depende da presença de matéria escura, assim como os sintomas de uma doença são atribuídos a um agente patológico. Os números computáveis são, por definição, aqueles números cuja expressão decimal é computada por uma máquina. A precedência da competência só acontece, contudo, quando os actantes *já possuem* uma figuração. Começamos com desempenhos, e é *só depois* que a competência poderá lhes *ser anterior*. É a diferença entre ciência em ação e ciência estabelecida. Na primeira as controvérsias ainda estão abertas, encontramo-nos em meio a tentativas de dar figurações a actantes, de constituir competências a partir de controvérsias. Quando uma ciência já se estabeleceu, e as controvérsias foram resolvidas, é que pode falar de um mundo onde determinados atores são origem de seus atos, como é o caso da natureza.

Uma lista de ações, entretanto, não pode compor um ator, "nenhum evento [acontecimento] pode ser explicado por uma lista de elementos que penetraram na situação *antes* de sua conclusão" (LATOUR, 2001, p. 147). Não podem, também, ser definida por atores já existentes ou estáveis. É preciso que algo seja acrescentado, que no experimento científico seja produzido algo para além de seus componentes iniciais. O novo ator deve constituir um acontecimento. Um experimento científico é um acontecimento, e não uma

descoberta, pois não desvela algo que já estava ali desde o começo, uma natureza ou realidade primeira escondida que se manifestava apenas de modo indireto. É preciso realizar toda uma construção, uma invenção extremamente complexa, para que o novo ator possa ser produzido, isto é, têm-se de arregimentar e alistar uma enorme lista de atores e actantes, que serão todos modificados e transformados ao final do experimento. O acontecimento não é apenas o acréscimo de um novo elemento a uma situação que permanece idêntica. Um novo coletivo é composto e, com isso, as ações passam a ser desempenhadas em um mundo diferente. "Porque a única maneira de definir um ator é por intermédio de sua atuação; assim também, a única maneira de definir uma atuação é indagar em que outros atores foram modificados, transformados, perturbados ou criados pela personagem em apreço" (LATOUR, 2001, p. 143).

Ao aliar-se às máquinas automáticas, Turing não realiza uma simples substituição, colocando um ator no lugar de outro. As ações do computador humano são delegadas às máquinas, mas apenas na medida em que o próprio computador é modificado. Para que possa ser substituído o computador humano deve ocupar uma posição que torna a sua associação com a máquina possível. Prosseguindo em seu artigo, Turing (2004 [1936]), após dar os primeiros elementos da prova da decidibilidade, busca delimitar a extensão dos números computáveis. Para isso deve responder à questão, "Quais são os possíveis processos que podem ser executados na computação de um número?" (p. 74)[xv] As respostas serão de três tipos: 1. um apelo direto à, intuição; 2. uma prova de equivalência; 3. exemplos de grandes classes de números computáveis. Nos dois primeiros tipos, Turing descreve precisamente como se dá o ato da computação, a partir da ação de um computador humano.

> A computação é normalmente feita pela escrita de certos símbolos em papel. Podemos supor que esse papel é dividido em quadrantes como um livro de aritmética infantil. Em aritmética elementar o caráter bidimensional é eventualmente utilizado. Mas tais usos são sempre evitáveis, e penso que pode-se concordar que o caráter bidimensional do papel não é essencial à computação. Assumo que a computação é desenvolvida em papel unidimensional isto é, em uma fita dividida em quadrantes. Devo supor também que o número de símbolos que podem ser escritos é também finito. […]
>
> O comportamento do computador é determinado a qualquer momento pelos símbolos que observa e por seu "estado mental" atual. Podemos supor que há um limite *B* ao número de símbolos ou quadrantes que o computador pode observar em um momento. Se deseja observar mais, deve utilizar observações sucessivas. Vamos supor também que o número de estados mentais que precisam ser considerados é finito. […]
>
> Vamos imaginar as operações realizadas pelo computador como sendo divididas em "simples operações", que de tão elementares é difícil imaginá-las em divisões adicionais. Cada operação consiste na mudança do sistema físico constituído pelo computador e por sua fita.
>
> […]
>
> As operações simples devem incluir, portanto:
>
> (a) Mudanças do símbolo em um dos quadrantes observados.
>
> (b) Mudança de um dos quadrantes observados para outro quadrante entre *L* quadrantes [de distância] de um dos quadrantes previamente observados.

> Pode ser que algumas dessas mudança envolvam necessariamente a mudança de um estado mental. A operação mais simples e geral deve, consequentemente, ser tomada como uma das seguintes:
> (A) Uma possível mudança (a) de símbolo junto com uma possível mudança de estado mental.
> (B) Uma possível mudança (b) de quadrantes observáveis, junto com uma possível mudança de estado mental (TURING, 2004 [1936], p. 75–77).[xvi]

As ações que podem ser realizadas por tal computador são as mesmas das máquinas de computar já descritas. Uma vez que todo o repertório de ações do computador foi definido, pode-se construir uma máquina capaz da realizar o mesmo trabalho. No caso da máquina, as suas configurações-*m* equivalem aos estados mentais do computador. Tem-se aí, elementos chave que serão retomados pela cibernética, inteligência artificial e ciências cognitivas, ao conceberem os processos cognitivos em termos de estados mentais definíveis, discretos e compartimentalizáveis, equivalentes às configurações de uma máquina. Contudo, neste momento o que importa é definir tanto o que um computador quanto uma máquina são capazes de fazer ou não. Tanto é que mais à frente Turing reformula alguns dos aspectos da ação do computador humano, evitando a ideia de "estado mental", em favor de um recurso mais físico.

> [Supomos que] a computação é executada em uma fita; mas evitaremos introduzir o "estado mental" ao considerar uma contraparte mais física e precisa. Ao computador é sempre possível largar seu trabalho, ir embora e esquecer tudo sobre sua tarefa, e retornar depois para lhe dar continuidade. Se fizer isso, é preciso que deixe uma nota com instruções (escritas em alguma forma padrão) explicando como o trabalho deve ser continuado. Essa é nota é a contraparte do "estado mental". Podemos supor que o computador trabalha de uma maneira tão incoerente que que nunca executa mais do que um passo por sessão. A nota com instruções deve permitir que execute um passo e que escreva a próxima nota. Portanto, o estado do progresso da computação em qualquer estágio é completamente determinado pela nota com instruções e os símbolos na fita (TURING, 2004 [1936], p. 79).[xvii]

Se em um primeiro momento era possível assumir que computador humano e máquina eram completamente intercambiáveis, agora se torna evidente que não o são. Ou, mais precisamente, fica claro tudo o que é necessário para que um computador humano comporte-se como uma máquina. É necessária toda uma disciplina, uma organização de sua ação em gestos elementares de um sistema de anotações preciso e complexo (para não dizer ineficiente) que guie seu comportamento, especialmente no caso de um computador humano *incoerente*. Em outras palavras, ao invés de desempenhar o papel de mero substituto da ação humana, como um simples intermediário, a máquina *desloca* a ação humana, opera como um desvio que problematiza e torna mais incerto o resultado a ser obtido, necessitando introduzir mais recursos e passos no rumo do objetivo inicial, e com isso torna o próprio computador *mais incoerente*.

Quem age, Turing ou a máquina? Turing, com certeza, pois descreve, argumenta e agencia a máquina, mas a máquina também age, pois é quem computa um número até o infinito, não Turing. O que se desenrola é um pedido para acompanhar ou seguir as operações da máquina. E se precisamos recorrer à imaginação, é por conta de *nossa* limitação, e não a da máquina. Imaginamos a máquina pois não podemos *fazer o mesmo*. É por nossa fraqueza que precisamos de um aliado mais forte, ainda que este não exista do mesmo modo que nós. A ação é possível apenas neste encontro.

A associação com máquinas e a substituição do computador humano não constituiu um gesto necessário ou indispensável. Turing poderia muito bem ter se limitado a descrever as operações de um humano para definir a noção de computabilidade, tal como foi feito por Post (1936, 1965a). Também em 1936, Post publica um artigo em que descreve um operador [*worker*] que circula entre caixas marcando e apagando símbolos.

> Na presente formulação o espaço de símbolos consiste em uma sequencia infinita bidirecional de espaços ou caixas, i.e., similar à sequência ordinal de números inteiros …, -3, -2, -1, 0, 1, 2, 3, … . O solucionador de problemas ou trabalhador deve trabalhar e mover-se em um espaço de símbolos, sendo capaz de estar em, e operar apenas uma caixa por vez. E independentemente da presença do trabalhador, uma caixa admite apenas duas condições possíveis, i.e., ser vazia ou não marcada, e ter uma única marcação em si, digamos que um traço vertical.
> […]
> Assume-se que o trabalhador é capaz de realizar os seguintes atos primitivos.
> (a) *Marcar uma das caixas em que se encontra (assumindo que esteja vazia),*
> (b) *Apagar a marca na caixa que em que se encontra (assumindo que esteja marcada),*
> (c) *Mover para a caixa em sua direita,*
> (d) *Mover para a caixa em sua esquerda,*
> (e) *Determinar se a caixa em que se encontra está marcada ou não*
> (POST, 1965a, p. 289).[xviii]

As ações realizadas pelo trabalhador de Post são muito similares às realizadas pelo computador humano ou pelas máquinas-a, com a diferença de que a sequência de espaços é bidirecional, enquanto a fita das máquinas é unidirecional, e de que são utilizados apenas dois símbolos, enquanto Turing utiliza vários. Em todo caso, as duas descrições serão posteriormente tomadas como idênticas ou, no mínimo, como equivalentes, inclusive pelos próprios autores. Mas se há a necessidade de afirmar a equivalência entre as duas descrições é, justamente, *por não serem equivalentes*. Toda tradução media elementos heterogêneos. Em outro momento, Post (1965b) propõe um modelo de máquina de Turing que, de acordo com ele, corrige certos erros da formulação inicial, além de incorporar diversas propriedades do espaço simbólico com seu trabalhador. Ao se tornarem equivalentes, computador, máquina e trabalhador podem ser associados e intercambiados, criando novos atores híbridos.

Mas como afirmarmos mais acima há uma diferença entre o modo como certos atores

são constituídos na matemática e na ciência. Enquanto que em um experimento científico podemos observar o processo pelo qual uma série desempenhos adquire uma competência, no caso da matemática isto não acontece.

> Quanto *mais formalizado* é um dado campo, mais obvio será que performances e competências encontra-se juntas sem nada entre elas. Se há uma definição de objeto matemático compartilhada entre qualquer posição filosófica a que você esteja inclinado, é que *aquilo que são* é completamente descrito por *aquilo que fazem.* Objetos matemáticos nascem pragmáticos, pode-se dizer, no sentido que se comportam tal como foram definidos (bem, quase sempre) […]. Os vários pequenos intervalos que tiveram de ser atados juntos e lentamente alisados para criar um objeto mundano […] não são tão exigentes no caso de uma entidade formal dado que seu comportamento, como dizem, é "completamente ditado" por sua definição. É claro, existem muitas brechas (caso contrário não haveria nenhum ganho na demonstração), mas é sempre possível pular sobre cada uma das quebras depois de haver retraçado seu caminho sem precisar buscar nada em outro domínio. (LATOUR, 2015, p. 10)

As associações e substituições colocam em contato atores e actantes que, de outro modo, estriam isolados. Para isso devem realizar saltos, em maior ou menor medida, na tentativa de conciliar e compor suas heterogeneidades, como no caso da tradução, o que constitui um acontecimento, o aparecimento de uma entidade que não pode ser reduzida ao estado de coisas anterior. No caso da matemática, como performances e competências já estão dadas, não é preciso buscar um ator que agencie as listas de ações, uma vez que esta já é dada a partir de uma definição prévia – uma competência. Entre definição e ação há um caminho reversível, que permite a livre passagem de um a outro, ignorando detalhes e especificidades.

Em um primeiro momento, os números computáveis constituem uma lista de ações, são um actante. Recebem uma figuração quando lhes é atribuída uma competência, quando passam a ser os números cuja expressão decimal pode ser computada por uma máquina. A máquina, contudo, não foi "descoberta" no mesmo sentido em que acontece nas outras ciências, onde as listas de ações precederam o gesto que as integrariam à uma figura capaz de servir como sua origem ou princípio organizador. No caso da matemática já temos uma figura, partimos da definição do ator em direção aos actantes, performance e competência se dão simultaneamente. O acontecimento neste caso não segue o modelo da "descoberta", mas o da "construção" ou "demonstração". Contudo, como Latour também afirma, isto não livra o saber matemático ou formal de incertezas. Uma definição determina completamente o comportamento da entidade que descreve, o que não impede que a entidade venha a se mostrar como algo muito maior que a definição. Por exemplo, ainda não existe uma equação que seja capaz de produzir todos os números primos. Existem equações que produzem grande parte dos números, mas não todos. A questão é saber se tal equação completa é possível. Os números primos produzidos pelas equações existentes são completamente determinados por

sua definição, mas existem outros números primos que existem como listas de ações, que são produzidos mas não se integram a nenhum ator dado. Tal como na ciência, a questão permanece em aberto à espera de um acontecimento que possa lhes dar uma figuração.

A relação entre competência e performance que é própria aos saberes formalizados se dá, no caso da matemática, pelo modo como esta se utiliza de experimentos mentais. Rotman (1993) desenvolve um modelo de ação em matemática partindo da ideia de que o fazer matemático se dá exclusivamente por experimentos mentais executados por três níveis de agência: Pessoa, Sujeito e Agente.

> O modelo identifica o raciocínio matemático em sua totalidade – provas, justificações, validação, demonstrações, verificações – com a execução de ações imaginadas que detalham a realização passo-a-passo de certas narrativas instituídas simbolicamente e mentalmente experimentadas. Portanto, diferentemente da ciência física em que experimentos mentais são contrastados com experimentos reais, considerados como um dispositivo racionalizante e persuasivo entre outros, a matemática, como apresentada aqui, será exclusiva e singularmente fundada sobre eles. (ROTMAN, 1993, p. 66).[xix]

Nas ciências naturais, os experimentos mentais, tais como os de Galileu, Einstein e Maxwell, são utilizados quando se quer afirmar uma determinada ideia ou argumento cuja demonstração empírica seria, no mínimo, impraticável, ou até mesmo impossível. Esses experimentos, uma vez que não podem ser comprovados diretamente, que é justamente a função de suas contrapartes "reais", acabam desempenhando um papel retórico.

O corte estabelecido entre experimentos mentais e reais, entretanto, nem sempre se dá de forma tão clara ou exata. Um experimento real, como o realizado em um laboratório, precisa ser de algum modo retransmitido, aparecendo em algum tipo de publicação ou comunicação que permita a outros tomarem conhecimento de seus resultados ou até mesmo reproduzi-lo, caso possuam os recursos necessários. Enquanto o experimento relatado não é refeito, se é que chega a sê-lo, não opera de modo diferente de um experimento mental. Possui os mesmos mecanismos retóricos, uma vez que exige que as situações em que se deu sejam imaginadas e reconstituídas mentalmente pelos leitores. O estabelecimento de um fato científico depende dos procedimentos retóricos utilizados por experimentos mentais. Isso não quer dizer que os dois tipos de experimentos sejam a mesma coisa. A diferença situa-se no objeto das narrativas: experimentos mentais tratam de ficções e situações imaginárias, enquanto experimentos reais referem-se a dados e situações empíricas (ROTMAN, 1993).

No caso das matemáticas, não há distinção entre os dois tipos de experimentos, uma vez que é constituída exclusivamente por experimentos mentais. Um experimento mental é um procedimento complexo, composto por séries de ações distintas, que, contudo, não esgota o fazer matemático. A matemática é, basicamente, uma prática voltada à comunicação, à

construção de um saber, desdobrando-se em pelo menos dois níveis. O nível que pode ser chamado de formal ou rigoroso

> é composto por todos aqueles textos *escritos* (livros, artigos, divulgação de pesquisas, lições em quadros negros) que são apresentadas de acordo com regras muito precisas e sem ambiguidade, - convenções e protocolos controlando os padrões permitidos de uso de signos com respeito ao conteúdo de asserções e suas validações lógicas, o estados de conjecturas e hipóteses, definições, especificação de notações, exibição de projetos, e assim por diante. Na extrema forma de laconismo permitido aos matemáticos, tais textos constituem-se por sequências de definição, teorema, prova e nada mais. (ROTMAN, 1993, p. 69).[xx]

O nível formal constitui o *Código*.[14]

Por sua vez, o nível informal, ou não rigoroso, do *metaCódigo* [*metaCode*] agrega uma heterogeneidade de ações, como

> desenhar figuras ilustrativas e diagramas; dar motivações; suprir ideias cognatas; desdobrar intuições, princípios-guia e histórias pressupostas; sugerir aplicações; fixar as interpretações esperadas de sistemas formais e notacionais; fazer conexões extra-matemáticas – geralmente usando uma linguagem natural assim como uma variedade de movimentos explicativos, figuras de linguagem e dispositivos iconográficos para transmitir toda forma de sentido matemático, indo desde o nível do conteúdo técnico até o "ponto" de uma prova, a "trivialidade" de um resultado ou a "fecundidade" de um contra-exemplo. (ROTMAN, 1993, p. 69–70).[xxi]

A distinção entre Código e metaCódigo parecer ser clara. De um lado, encontram-se todas as atividades que constituem a "verdadeira" matemática, que pode ser reconhecida por seu rigor e exatidão, sendo lógica, racional, objetiva, atemporal, livre de todos os elementos mundanos e acidentais, dedicando-se exclusivamente a um domínio de signos e de pura escrita, o Código. Do outro lado, situam-se todos aqueles elementos "estranhos" à matemática, que apesar de nunca estarem ausentes do fazer matemático, são desnecessários, e ao serem levados muito a sério podem induzir ao erro. O metaCódigo não passaria de um domínio de exposição ou comentário, suplemento ou epifenômeno, acerca de um Código que seria a fonte real do saber matemático.

Pode-se, contudo, traçar a diferença entre os dois níveis considerando-os a partir de diferentes *modos de agência*. Ao lado do metaCódigo encontramos a *Pessoa*, indivíduo corpóreo, dotado de uma história pessoal, sentimentos, emoções, que vive no mundo e é dotado de uma dêixis, a capacidade indicar a si mesmo com pronomes pessoais. A Pessoa *fala,* utiliza figuras de linguagem, gesticula. Constrói um espaço comum para a matemática a partir dessa relação presencial e prosódica com outras pessoas.

O Código constitui-se pela ação de um *Sujeito*, entidade capaz de manipular símbolos. Em ultima instância é o Sujeito quem lê e escreve, e consequentemente é aquele que imagina, que demonstra e prova. A partir da Pessoa, o Sujeito é produzido por um duplo

14 O sentido do "código" que será trabalhado nos capítulo 3 e 5 difere da concepção de Rotman.

processo de idealização e esquecimento. Enquanto entidade ideal, é a-histórico, a-social e a-cultural, pura subjetividade trans-individual e universal. Nos textos matemáticos, a escrita é realizada em tal posição, que exige ao Sujeito esquecer tudo aquilo que define a Pessoa: suas memórias, sua temporalidade, espacialidade e, principalmente, a faculdade de dizer "Eu", de situar-se em um ponto individual do mundo. O Código não tem a capacidade de registrar tais elementos não formais. O Sujeito é um esqueleto da Pessoa, é anônimo e mantém apenas o mínimo de características necessárias para realizar sua tarefa ideal de demonstrar e provar, "uma voz atemporal de ninguém e de lugar nenhum" (ROTMAN, 1993, p. 74).[xxii].

A quem é endereçada a instrução "calcule os números primos maiores que 13"? À Pessoa ou ao Sujeito? Uma vez que é este quem escreve, deve restar à Pessoa a tarefa de calcular, infinitamente, tais números. Contudo, a Pessoa não manipula símbolos escritos, opera no domínio da fala e da informalidade, ou, pelo menos, não ao nível de fórmulas e execuções de instruções, especialmente quando requerem a realização de tarefas impossíveis a um ser finito e limitado. É necessário que haja um terceiro modo de agência, o que Rotman chama *Agente*.

> O Agente, diferentemente do Sujeito, não possui nenhuma habilidade de imaginar e pode responder a signos apenas em sua forma truncada, esquelética, enquanto significantes desprovidos de significado intencional. Em outras palavras, o Agente é concebido enquanto um autômato, um completo substituto [proxy] mecânico e formal do Sujeito. (ROTMAN, 1993, p. 76).[xxiii]

Se o Sujeito é constituído por uma abstração da pessoa, o Agente é produzido por uma segunda abstração, tornando-o completamente incorpóreo e desprovido de qualquer conexão com uma realidade específica. É esse descolamento que o torna apto a realizar tarefas impossíveis ao Sujeito ou a Pessoa, como percorrer uma distância infinita entre números. As ações realizadas pelo Agente não precisam respeitar nenhuma lei da física, nenhum princípio de conservação de energia ou limitação temporal. O que importa não são as restrições impostas, mas única e exclusivamente as instruções que deve seguir de modo completamente automático. Por isso o Agente opera ao nível de um *subCódigo*, em que manipula os símbolos sem lhes dar uma significação ou sentido, executando essas ações no lugar do Sujeito ou da Pessoa.

Em um texto matemático, é o Sujeito que toma o lugar de maior evidência, buscando apagar tanto as referências à Pessoa, isto é, a qualquer contextualização físico-material, quanto a referência ao Agente, este sendo um mero desdobramento do Sujeito, figura imaginária cuja existência não possui densidade suficiente para ser considerada à parte. Ou melhor, esta é a perspectiva do próprio Sujeito e do discurso matemático na medida em que

descreve a si mesmo de uma maneira exclusivamente formal. Rotman (1993), entretanto, concebe este modelo tripartite da ação com o intuito de tornar evidente tudo o que se dá efetivamente na prática matemática e que não pode ser reduzido apenas a um dos termos envolvidos, ou a uma escola matemática específica. O fazer matemático transforma-se, então, em um modo de ação que agencia os elementos mais diversos e heterogêneos.

Seguindo esta divisão, a ação de Turing é distribuída através de pelos menos três registros: temos a pessoa Turing, com uma história pessoal, desejos, paixões e vontades, que compõem com o sujeito Turing, que lê, escreve e interpreta símbolos, que prova e demonstra, e que se cerca de delegados, agentes, como computadores, máquinas-a e máquinas universais, que realizam o trabalho mecânico de cálculo e computação proposta em seu texto. Temos uma divisão, ou até uma assimetria, que coloca Turing enquanto pessoa e sujeito de um lado, e seus agentes, em grande parte não-humanos, de outro. Claro, tais agentes, na proposta de Rotman, ainda fazem parte de Turing, pois são imaginados. A noção de imaginação, contudo, é problemática, pois apoia-se em uma separação entre realidade e ficção, como se os agantes possuíssem menos realidade que sua contraparte física, a pessoa, sendo que o sujeito ficaria em uma situação intermediária.

Contudo, *ainda não foi decidido o que é real,* a delimitação entre fato e ficção, sujeito e objeto encontram-se abertas. Já que nosso ponto de partida é a ação, seria mais interessante dizer que constituem diferentes domínios de prática, cada um com sua própria materialidade e efetividade. O sujeito substitui a pessoa, do mesmo modo que é substituído por agentes, o que não esgota as relações possíveis entre os atores, que também compõe associações, criando todo tipo de híbridos.

Turing propôs em seu programa de ação a definição da classe dos números computáveis. Para atingir seu objetivo teve de recorrer a diversos *desvios*, como a introdução e laboriosa construção de suas máquinas, em especial a máquina universal, além de especificar a sua relação com computadores humanos, chegando a exemplos e demonstrações da extensão dos números computáveis, além de definir algumas de suas propriedades. Em outro desvio, ao terminar de construir a máquina universal, mostrou a existência de máquinas que são incapazes de computar determinados números. Para que este desvio venha a compor um objetivo auxiliar, isto é, um subprograma de ação, nos termos da lógica e do *Entscheidungsproblem*, é necessário que todo o repertório de atores, competências e performances que Turing introduziu em seu artigo sejam traduzidas para a linguagem da lógica de primeira ordem. A seção final do artigo é dedicada à converter, parcialmente, as máquinas-a e a máquina universal na notação da lógica, traduzindo as máquinas

incomputáveis em proposições indecidíveis da lógica de primeira ordem.

A tradução das máquinas para a linguagem da lógica não era algo estritamente necessário, mas ao fazê-lo Turing tornou seu subprograma mais viável. Se por algum motivo a tradução fosse impossível, Turing se encontraria isolado e em desacordo com os termos da comunidade lógica, cabendo-lhe o ônus de, sozinho, convencer seus interlocutores a modificarem todo o edifício da lógica moderna para comportar as máquinas de um autor desconhecido ou, então, de que a lógica deveria coexistir com entidades que não falassem 0 mesmo idioma, duas opções que seriam impossíveis com quase toda a certeza. A possibilidade de formalização, como tradução, permite que Turing se alie com uma tradição bem estabelecida e, do que é próprio da formalização, seja comunicável e transportável para os mais variados domínios.

Isto nos leva à especifidade dos atores. Máquinas, computadores humanos e equações podem ocupar a posição de agentes o que, contudo, não supõe que sejam equivalentes ou redutíveis uns aos outros. Como vimos, um computador só pode equivaler a uma máquina após uma complexa série de transformações que, ao final, o fazem agir de um modo no mínimo peculiar. Se tanto as máquinas-a, quanto a máquina universal são agentes, devemos colocar agora a questão de sua especificidade. Dada a possibilidade de tradução, por quê falar em máquinas e não um sistema formal, como o cálculo-λ ou funções recursivas?

2.6 Computabilidade e suas equivalências

Em 1937 o *Entscheidungsproblem* deixara de ser o problema fundamental da matemática para tornar-se mais um episódio do fracasso das pretensões totais do programa formalista de Hilbert. Como é caso de uma teoria ou hipótese que é refutada, Hilbert viu-se desprovido de aliados, o que pretendia ser um fato transformou-se em ficção. Desfecho nem um pouco evidente quarenta ou trinta anos antes, quando o método formal axiomático dominava o pensamento matemático e prometia, com o devido empenho e trabalho, solucionar todos os grandes problemas da matemática. “Esta convicção na solução de cada problema matemático é um poderoso incentivo ao praticante. Ouvimos em nosso interior o perpétuo chamado: há um problema. Busque a sua solução. Você pode encontrá-la através da razão pura, pois na matemática não há *ignorabimus*[15]” (HILBERT, 2000 [1900], p. 412).[xxiv] Seguindo este espírito, em 1900, no Congresso Internacional de Matemática de Paris, Hilbert

15 O *ignorabimus* é uma expressão que refere-se à impossibilidade de conhecer algo.

selecionou 23 problemas que deveriam guiar a pesquisa matemática no próximo século, cuja solução colocaria a matemática em bases consistentes e auto-justificáveis, sendo que o décimo problema constituía uma primeira versão do *Entscheindungsproblem*. O programa de Hilbert desenvolvia-se em paralelo ao projeto de Russell e Whitehead de colocar toda a matemática em bases lógicas, em seu monumental trabalho *Principia Mathematica,* constituindo uma colaboração mútua e aliança poderosa, apesar de todas as diferenças entre formalismo e lógica.

Como foi visto, os trabalhos de Gödel, Church, Kleene, Turing e Post desenvolveram-se em grande parte dentro da perspectiva lógico-formal. Os métodos utilizados e os problemas abordados nas pesquisas foram todos colocados nos termos formalistas e guiados por seus objetivos. Nenhum dos autores sozinhos teria a força suficiente para opor-se a um aparato tão bem montado e estabelecido. Por mais que Gödel duvidasse da completude de sistemas formais que incorporassem a aritmética, sua dúvida sozinha não conseguiria convencer a maioria do estabelecimento matemático, se é que convenceria alguém, nem teria a força para colocar em xeque a esperança formalista. Assim como um cientista recorre ao laboratório na busca de aliados a sua hipótese, Gödel teve de articular o aparato matemático e lógico com seus dispositivos de notação, sistemas formais e esquemas de dedução para que pudesse introduzir um novo ator, a "numeração de Gödel". Como todo ator, a numeração define-se pelas transformações e deslocamentos que provoca em outros, neste caso, fez com que números naturais, que poderiam ser referidos apenas em um discurso metamatemático, pudessem referir-se a si mesmos, apoiando-se em funções recursivas e sendo levada à prova da incompletude. É toda a potencialidade dos números naturais e das funções recursivas que é modificada pela aliança estabelecida entre Gödel e sua numeração.

Os atores introduzidos por Gödel, por mais engenhosos que o sejam, não produziriam nenhum efeito caso não fossem considerados e levados em conta por seus colegas. A primeira apresentação da prova de Gödel se deu em um congresso em 1930, no qual Hilbert estava presente. Ao que tudo indica, com a exceção de John Von Neumann, nenhum dos presentes compreendeu seu argumento ou as consequências de sua prova. Era preciso que a prova de Gödel se inserisse em um "coletivo de pensamento" (FLECK, 2010), que assumisse uma "representação pública" (LATOUR, 2001), isto é, deveria ser capaz de circular, interessar e pertencer a um determinado coletivo. Retomando os termos de Rotman, não bastava que o texto fosse desenvolvido ao nível do Sujeito ou que introduzisse novos Agentes, era necessário que operasse no nível informal, no domínio da Pessoa. A aliança com Von Neumann e a publicação de seu artigo em 1931 garantiu à prova de Gödel a sua

circulação no meio matemático, especialmente em Princeton, sendo lido por Church e Kleene, e onde daria um curso onde desenvolveria a noção de função recursiva geral, em 1934.

Os trabalhos publicados em 1936 apoiavam-se todos no trabalho realizado por Gödel, pelo modo como modificou os números naturais e as funções recursivas, assim como por ter enfraquecido os pressupostos e objetivos formalistas, dando margem a novos rumos de pesquisa. Contudo, o horizonte aberto por Gödel, por si mesmo, não solucionou o problema da decidibilidade. Como vimos, a elaboração de um ator constitui um acontecimento, pois se nada for transformado, se nenhuma diferença for produzida não há ator. A solução negativa ao *Entscheidungsproblem* não foi um simples desdobramento do que já estava dado, uma dedução lógica e necessária que já estava completamente contida em suas premissas. Foi necessário inventar, *fabricar* uma solução, o que nos levou ao recrutamento de pelo menos quatro grandes coletivos de atores, humanos e não-humanos, que criaram distintos mundos onde a solução para o problema de decisão não era possível.

Funções recursivas gerais, cálculo-λ, máquinas de Turing e trabalhadores de Post são todos acontecimentos, coletivos agenciados recrutando humanos e não-humanos, que transformaram os limites do que humanos e máquinas podem realizar no gesto de computar, o que pode ser decidido na lógica de primeira ordem, e os limites de procedimentos finitos em sistemas formais. Mas tal como foi o caso de Gödel, assim como de qualquer hipótese científica, a força de uma afirmação, seu destino, factualidade e objetividade, será decidido apenas na medida em que é retomada por outras afirmações.

Nosso ponto de partida não pode ser apenas o artigo de Turing, as máquinas que constrói ou o aparato que monta. Tudo isso só faz sentido na medida em que é um texto *apropriado por outros*, do mesmo modo que apropriou-se de textos que lhe precederam. Como na perspectiva da Teoria Ator-Rede não adotamos uma linguagem capaz de estabelecer objetos e referências *a priori*, os atores é que devem realizar esse trabalho. Começamos com um texto, agora temos de colocá-lo em movimento. Tais máquinas podem dizer muito, ou podem dizer nada. Podem ser fatos ou pura ficção. Não podemos decidir nada disso apenas com Turing. A qualidade intrínseca de suas máquinas, o que são *"realmente"*, encontra-se em mãos alheias. É necessário ir até seus leitores, e para isto introduziremos a noção de *modalidade* (LATOUR, 2000, 2012).

Isolada, uma afirmação em um texto científico não faz nada. Primeiro, precisa estar embasada em algo, como em referências a uma literatura que lhe precede ou em dados obtidos por um instrumento em um laboratório ou outro espaço de pesquisa. Quando Church, Kleene, Turing e Post em seus textos de 1936 citam Gödel, o fazem para estabelecer uma filiação ao

autor, arregimentando-o para assegurar uma base sólida como ponto de partida. Esta associação dificulta a ação de quem queria discordar do texto, pois terá de enfrentar também a prova da incompletude Gödel, suas funções recursivas e numeração, que naquele momento já se apresentavam como sólidas e não eram objeto de dúvida. Simultaneamente, o texto de Gödel é fortalecido ao ser incorporado como base indubitável dos textos de 1936. Isto nos leva ao segundo ponto, em que uma sentença apenas se transforma realmente na afirmação de um fato se é retomada por textos futuros, assim como o fez com suas referências. Uma sentença que nunca é lida, não pode ser nem verdadeira, nem falsa, fato ou ficção, pois simplesmente não faz parte do jogo que decide tais valores.

Contudo, o processo de assimilação de uma afirmação não é tão simples ou linear como pode parecer em um primeiro momento. A retomada de uma sentença por outra constitui uma modalidade. Esta é "o que modifica o predicado de um enunciado" (GREIMAS; COURTÉS, 2012, p. 314). Em outras palavras, é uma sentença que retoma outra sentença e a modifica neste processo. Por exemplo, uma sentença afirma que "a função *f* possui a propriedade *y*", enquanto outra sentença diz que "uma vez que a função *f* possui a propriedade *y*, pertence à classe *z*". A segunda sentença retoma a primeira e a fortalece ao assumir tudo aquilo que é dito, modalizando-a na medida em que não é uma simples repetição, ligando-a a um determinado conjunto de implicações. Este procedimento de modalização fica evidente nas provas ou afirmações que tornam dois sistemas formais equivalentes, permitindo que um seja convertido ao outro. Em 1936, Kleene publica outro artigo onde demonstra que o cálculo-λ e as funções recursivas gerais são equivalentes. No apêndice a seu artigo, e em um publicação posterior, Turing (2004 [1936], 1937) prova a equivalência de suas máquinas tanto ao cálculo-λ e às funções recursivas gerais, além de afirmar que sua noção de computabilidade é equivalente à de calculabilidade efetiva de Church. Já em 1937 temos uma equivalência geral entre máquinas de Turing, cálculo-λ e funções recursivas gerais, com Post sendo incluído em 1943. Tornar duas coisas equivalentes já constitui uma modalização, pois ambas as entidades são modificadas ao serem associadas. Do contrário, o gesto de conversão não seria necessário. Turing precisa mostrar como suas máquinas podem ser convertidas ao cálculo-λ e, inversamente, como esta pode funcionar como uma máquina de Turing, fazendo com que sistemas concebidos em contextos de práticas distintas sejam levados a novos âmbitos de seu fazer. Ainda que em um primeiro momento os textos de 1936 tenham sido escritos com um certo garu de independência, em 1937 encontravam-se completamente entrelaçados, constituindo um coletivo ainda mais forte, tal como uma corda ou tecido cuja força deriva do cruzamento de seus filamentos.

A equivalência entre duas coisas não se refere a propriedades comuns ou isomorfismos que possuam; é antes um gesto, uma ação que *compara* e *equaliza* dois atores. A equivalência não é uma propriedade intrínseca, é algo que é *acrescentado* ao que já está dado, gesto que se compõe, desloca ou fortalece um certo arranjo de relações, como uma *tradução* (CALLON, 1986). Entre a máquina de Turing e sua representação através do cálculo-λ há o gesto que converte uma a outra. Esta tradução, contudo, é muito mais do que a transformação de uma notação na outra, pois deve criar um canal de comunicação entre duas práticas, ou melhor, entre duas ontologias distintas. As máquinas e suas tabelas não constituem um sistema formal, encontram-se mais próximas ao que mais tarde será chamado de linguagem de programação, do que a um sistema dedutivo que estabelece e prova teoremas.

As modificações realizadas por uma modalidade não precisam apenas fortalecer uma afirmação, podem enfraquecê-la também. Uma modalidade pode agregar ou isolar um enunciado, tudo depende de como se tá tal relação. No caso dos textos científicos, quanto mais aliados conquista, seja pelo cuidadoso trabalho que realiza com suas referências, modalizando e fortalecendo aquelas que podem lhe auxiliar, enfraquecendo aquelas que representam alguma oposição, seja através de sua citação e utilização em textos futuros. Quanto mais aliados e ligações, mais próximo o texto encontra-se dos fatos e, inversamente, na medida em que se isola torna-se ficcional.

> Chamaremos de *modalidades positivas* as sentenças que afastam o enunciado de suas condições de produção, fortalecendo-o suficientemente para tornar necessárias algumas outras consequências. Chamaremos de *modalidades negativas* as sentenças que, ao contrário, levam um enunciado para a direção de suas condições de produção, e explicam com detalhes porque ele é forte ou fraco, em vez de usá-lo para tornar mais necessárias algumas outras consequências (LATOUR, 2000, p. 42).

Na medida em que agrega modalidades positivas, um enunciado distancia-se de suas condições de produção, indo em direção ao fato, permitindo que seja agregado de forma não problemática em sentenças futuras, o que pode, superficialmente, ser visto como uma "progressão do conhecimento". Ao contrário, quanto mais modalidades negativas são atribuídas a um enunciado, mais ficcional ele se torna, tendo suas bases criticadas e problematizadas, levando tanto ao seu desaparecimento e relegação ao status de erro, quanto à construção de novas sentenças que reorganizem o saber em questão. Entretanto, as duas modalidades se dão, geralmente, em conjunto. Para ser fortalecido um texto precisa também enfraquecer outros enunciados concorrentes e dissociar atores. Similarmente, uma mesma sentença pode ser decomposta em suas partes, onde algumas são aceitas e outras rejeitadas.

Tanto na prova da incompletude quando na solução negativa ao

Entscheidungsproblem, os sistemas formais são dissociados, na medida em que seus próprios termos são modalizados, seja na utilização de sua própria linguagem, como é caso das funções recursivas, seja pela criação de um novo sistema dentro das regras do jogo formal, como o cálculo-λ, ou ainda, desviando-se dos pressupostos formalistas e criando um conjunto de práticas estabelecidas ao redor de máquinas ou computadores humanos. Os termos do programa formalista são modificados por dentro, ou são levados a condições externas, no caso das máquinas, onde os atores formais não resistem às provações que lhe são colocadas por máquinas que computam a si mesmas e computadores humanos incoerentes.

As modalidades não constituem um caminho linear, que simplesmente adiciona ou retira coisas de uma trajetória retilínea. Cada modalidade nos leva a trilhar caminhos diversos. Uma mesma afirmação pode ser considerada verdadeira de modos completamente distintos, dependendo das traduções que sofre e a quais atores ou coletivos é associada. Como veremos mais adiante, as máquinas de Turing acabam constituindo-se como uma certeza, ao mesmo tempo em que são constantemente modificadas. Por mais que sejam modalizadas, criticadas e modificadas, as máquinas de Turing são fortalecidas a tal ponto que, neste contexto matemático e lógico, não encontram nenhum discordante sério. Tornam-se um ponto de passagem obrigatório pelo qual textos e discussões acerca da computabilidade têm de passar, utilizando os seus termos.

> Esta situação rara é aquilo que as pessoas costumam ter em mente quando falam em "fato". Espero que esteja claro que esse acontecimento não o torna qualitativamente diferente da ficção; um fato é algo que é retirado do centro das controvérsias e coletivamente estabilizado quando a atividade dos textos ulteriores não consiste apenas em crítica ou deformação, mas também em ratificação. A força da afirmação original não reside em si mesma, porém deriva de qualquer dos textos que a incorporam (LATOUR, 2000, p. 72).

Tornar-se um fato, contudo, tem um preço. As traduções pelas quais um ator passa, do mesmo modo que o fortalecem, o transformam em *outra entidade*. Quanto mais objetivo um fato torna-se, mais se afasta de suas condições iniciais, muitas vezes sendo colocado em termos absolutamente distintos e estranhos à sua formulação original. É o que acontecerá, por exemplo, quando a máquina de Turing for formalizada de determinadas maneiras. A transformação em fato implica na ampliação do raio de circulação do ator, que pode ser considerado como uma entidade que existe independentemente das condições em que foi produzido, que "existe lá fora", e que por conta dessa estabilização é capturada e associada a práticas cada vez mais distantes e heterogêneas a seu contexto original. A objetivação não se dá por um processo de redução, mas de ampliação, pois quanto mais objetiva é uma entidade, mais dotada de associações e alianças, o que dificulta tremendamente o trabalho de torná-la

uma ficção, já que cada uma dessas relações deve ser rompida. Ao mesmo tempo, este processo coloca em perigo a própria autonomia e identidade do ator, que passa a ser o membro ou protótipo de uma classe, sai de seu status de actante, de lista de ações desconectada para ser figurado por um ator, incorporado a um coletivo e representado por um porta-voz. Ao invés de tornar-se o general, o eremita passa a ser mais um soldado, que fortalece o exército como um todo e que, todavia, é só mais um entre tantos, praticamente indiscernível.

A máquina de Turing foi traduzida, convertida e associada ao cálculo-λ e às funções recursivas gerais. Deixa de ser um acontecimento isolado e ingressa em uma ordem ou classe de entidades. Logo após publicação de seu artigo, Turing torna-se orientando de Church, publicando em 1938 o texto "*Systems of Logic Based on Ordinals*", no qual enuncia diretamente a equivalência de seus estudos anteriores às concepções de Gödel, Church, Kleene e Post.

> Uma função é dita "efetivamente calculável" se seu valores podem ser encontrados através de algum processo puramente mecânico. Embora seja razoavelmente fácil conceber intuitivamente esta ideia, é todavia desejável possuir uma definição com uma expressão mais exata e matemática. Tal definição foi dada primeiro por Gödel em Princeton em 1934 (Gödel [2], 26), seguindo em parte uma sugestão não publicada de Herbrand, sendo desenvolvida posteriormente por Kleene [2]. Estas funções foram descritas como "recursivas gerais" por Gödel. Não precisamos nos ocupar muito aqui com sta definição particular. Outra definição de calculabilidade efetiva foi dada por Church (Church [3], 356-358) que a identifica com a definibilidade-λ. O autor recentemente sugeriu uma definição correspondendo mais aproximadamente à ideia intuitiva (Turing [1], ver também Post [1]). Foi dito acima que "uma função é efetivamente calculável se seus valores podem ser encontrados através de algum processo puramente mecânico". Podemos tomar esta afirmação literalmente, entendendo como puramente mecânico um processo que possa ser executado por uma máquina. É possível dar uma descrição matemática, em uma certa formal normal, da estrutura destas máquinas. O desenvolvimento destas ideias leva à definição do autor de uma função computável, e à identificação da computabilidade com a calculabilidade efetiva. Não é difícil, ainda que um tanto laborioso, provar que estas três definições são equivalentes (Kleene [3], Turing [2]) (TURING, 2004 [1938], p. 150–151).[xxv]

A filiação de Turing ao projeto de Church e o fato de abandonar as tabelas e esquemas de seu artigo de 1936 em favor da notação do cálculo-λ se dá como um movimento em que um pesquisador isolado associa-se a um programa de pesquisa, e a uma linguagem, mais bem estabelecidos. Como aponta Petzold (2008), as tabelas de instruções que Turing elabora em seu artigo de 1936 não aparecerão em mais nenhum de seus textos publicados. É como se Turing abandonasse o caráter informal e intuitivo de suas máquinas, assumindo uma posição mais formalizada.

Deste modo, tudo aquilo que era específico às máquinas de Turing seria absorvido pela equivalência entre as definições de máquina, funções recursivas e cálculo-λ. Esta

passagem recíproca entre os três sistemas consistiria um esquema de escolha livre, dado que seriam igualmente verdadeiros. O pesquisador deveria escolher, então, o sistema mais econômico, que afirmasse mais com menos, capaz de produzir a maior generalidade com o menor número de elementos isto é, o sistema mais forte. Deste modo, por todas as suas conexões e filiações, o cálculo-λ e a teoria das funções recursivas gerais estariam em uma posição muito mais vantajosa que as máquinas de Turing, seja por seu pertencimento institucional a programas de pesquisa, seja por situaram-se na tradição matemática. É este princípio de economia que Fleck (2010) contesta, pois

> todas essas posições formais não levam em consideração, ou o fazem em reduzida medida, o condicionamento cultural e histórico da suposta escolha epistemológica, da suposta convenção. […] A história ensina que pode haver lutas árduas pelas definições de conceitos. Isso mostra como as convenções igualmente possíveis não são enxergadas como equivalentes, independente de quaisquer razões utilitaristas. (p. 49)

Como o próprio Turing coloca ao final de sua afirmação sobre a equivalência entre sua máquina e outras definições, "não é difícil, ainda que um tanto laborioso provar que estas três definições são equivalentes". Na equivalência encontramos algo que não pode ser subtraído, que é *somado* à equivalência e que estão não dá conta: seu caráter laborioso. Há um custo, que não é baixo, para que a equivalência seja realizada. A existência desse custo coloca um obstáculo à livre circulação e escolha entre as três opções. Uma afirmação não pode ser modalizada de qualquer maneira, cada uma existe em um campo de modalidades possíveis ou virtuais. A equivalência é o resultado de um laborioso processo que encadeia modalidades, criando uma aparente continuidade no local da miríade de operações descontínuas e heterogêneas que a produziram.

Ainda que possam ser convertidas umas nas outras, as definições não completamente equivalentes. Isto se dá tanto por *acoplamentos ativos*, como as contingências históricas e culturais que colocam em relação determinados atores, quanto por *acoplamentos passivos*, que concernem à ação dos próprios atores e cuja aparente independência da agência humana lhes dá um caráter "objetivo", "real" ou "efetivo" (FLECK, 2010). É aí que encontraremos a máquina em sua agência própria, em tudo aquilo que não pode ser reduzido ao cálculo-λ e funções recursivas, permitindo que constitua-se como um ponto de passagem obrigatório de estudos de autômatos, linguagens formais, computabilidade e, inclusive, como modelo abstrato dos computadores digitais. Por mais que seja formalizável, a máquina não se desfaz de seus acoplamentos ativos, de todos os sentidos culturais que lhe foram atribuídos que nos fazem reconhecer a máquina como algo distinto de uma equação, existindo em um mundo com potencialidades distintas. E do lado dos acoplamentos passivos a máquina se produz em

tudo aquilo que escapa à formalização, às indeterminações e regularidades que produz ao agir.

Em 1934, quando Gödel ministrou um curso em Princeton, onde introduziu a noção de "funções recursivas gerais", afirmou que as funções recursivas primitivas podem ser calculadas procedimentos finitos e, inversamente, que procedimentos finitos são calculáveis através de funções recursivas parciais. É uma afirmação muito similar à Tese de Church, de que cada função efetivamente calculável é recursiva geral. Todavia, neste momento Gödel não acreditava que as funções recursivas corresponderiam à totalidade de tudo o que pudesse ser efetivamente calculado, pois ainda não existia uma definição satisfatória de computação. Mesmo quando Church provou que a sua definibilidade-λ seria uma definição satisfatória de computabilidade ou calculabilidade efetiva, e equivalente às funções recursivas, Gödel não se convenceu de que a definibilidade-λ seria uma definição satisfatória da calculabilidade efetiva. (DAVIS, 1982a).

Foi apenas com a publicação do artigo de Turing em 1936 que Gödel foi convencido da existência de uma definição de calculabilidade efetiva. Como colocou em uma nota ao seu curso de 1934, escrita em 1965:

> Tarski em sua aula enfatizou (o que considero justo) a grande importância do conceito de recursividade geral (ou computabilidade de Turing). Parece-me que esta importância é devida principalmente ao fato de que com este conceito foi possível, pela primeira vez, ter sucesso em dar uma definição absoluta de uma noção epistemológica interessante, i.e., que não depende do formalismo escolhido. Em todos os outros casos abordados previamente, como demonstrabilidade ou definibilidade, foi possível defini-los apenas relativamente a uma dada linguagem, e para cada linguagem individual fica claro que o que se obteve não era o buscado. Para o conceito de computabilidade, no entanto, embora seja meramente um caso especial de demonstrabilidade ou decidibilidade a situação é diferente (GÖDEL, 1965 [1946], p. 84).[xxvi]

O conceito de computabilidade é imediatamente correspondido à recursividade geral, assim como às noções de demonstrabilidade e definibilidade. Ainda que sejam todos equivalentes, sendo a computabilidade de Turing um caso especial das outras noções, apresenta um caráter distinto. Primeiro, assim como a recursividade geral, a computabilidade não está presa a nenhum formalismo ou linguagem formal específica, o que lhe daria um grau de generalidade ou evidência maior. Segundo, e mais importante, a força desta definição deriva de sua redução à noção de uma máquina finita.

> O maior avanço tornou-se possível através da definição precisa do conceito de procedimento finito, que desempenha um papel decisivo nestes resultados. Existem muitas formas distintas de se chegar a tal definição, as quais, entretanto, levam todas exatamente ao mesmo conceito. O caminho mais satisfatório, em minha opinião, é o de reduzir o conceito de procedimento finito ao de uma máquina com um número finito de partes, como foi feito pelo matemático britânico Turing (GÖDEL [1951] apud SHAGRIR, 2007, p. 400–401).[xxvii]

Se a computabilidade pode ser colocada à parte é por sua associação a uma máquina, ao modo

como esta torna evidente a finitude e o caráter mecânico dos procedimentos de decisão. A máquina possui algo irredutível às outras definições. Haveria na máquina uma evidência intuitiva que não está presente nas outras definições.

Para Gödel, foi modo como Turing coloca a questão da computabilidade que elucidou as outras formas de definir cálculos efetivos. A máquina seria mais *convincente* que outros atores em sua capacidade de definição. Contudo, Shagrir (2007) argumenta que os contemporâneos de Turing, especialmente Church e Kleene, não atribuíram nenhum mérito especial para a máquina de Turing. Para Church a equivalência entre a computabilidade de Turing e a calculabilidade efetiva era imediatamente evidente, e que apesar de possuir um apelo intuitivo mais direto não era mais convincente. A força da máquina de Turing provinha de sua equivalência a outras definições, e não o inverso como afirmava Gödel.

Ainda como argumenta Shagrir (2007), o tratamento dado à computabilidade por Turing é retomada por autores pioneiros do campo da teoria de autômatos, como McCulloch e Pitts (1943), Shannon e McCarthy (1956), sendo que Minsky (1967) cita literalmente toda a descrição do processo de computação feito por Turing (TURING, 1936, seção 9), o que chama de "argumento de Turing". Isto não impede que considerem a validade da máquina de Turing como sendo dependente de sua relação às outras definições, com Minsky (1967) afirmando que "talvez o argumento mais forte em favor da tese de Turing é fato de que, com o passar dos anos, todas as outras tentativas notáveis de dar definições precisas e também intuitivamente satisfatórias de 'calculabilidade efetiva' acabaram sendo equivalentes" (p. 111).[xxviii] Após o livro de Minsky, o argumento de Turing, não foi foi retomado em livros didáticos de lógica ou ciência da computação, sendo abordado novamente apenas em textos mais recentes.

A melhor forma de seguir uma ciência em seu processo de formação é acompanhar suas controvérsias. Na medida em que ainda não está estabelecida uma ciência não decidiu o que será considerado como fato e aquilo que será relegado à esfera da ficção. Na ciência em formação o gesto que produziu as modalizações ainda não foi apagado, pois uma vez que uma controvérsia é resolvida, as afirmações que dai derivam aparecem como que desprovidas de história. Adotam um tom impessoal, atemporal e genérico, não são mais acompanhadas de outros enunciados que às levam às suas condições de produção ou que as apoiam estabelecendo equivalências e redundâncias. Todo o jogo de modalidade positivas e negativas é ocultado sob uma definição única ou padrão. É o caso da "Tese de Church", expressão que veio a condensar as diversas definições de computabilidade e calculabilidade efetiva, com a "Tese de Church-Turing" sendo uma de suas variantes, utilizada quando quer-se dar ênfase

para a concepção de Turing. Uma definição bem estabelecida não encontra-se imune às modalizações. Cleland (2007), Goldin e Wegner (2008) e Hofstadter (1999) apresentam inúmeras interpretações e releituras da Tese de Church, com ou sem Turing, que diferem dos mais variados modos em relação às formulações anteriores. São modalizações que se dão, contudo, após a condensação inicial sob a Tese de Church.

A história da máquina de Turing se dá, como vimos até agora, completamente desprovida de grandes controvérsias. É a exemplar narrativa de sucesso da estabilização de um fato. Para ser aceita a máquina de Turing será modalizada, associada a outras definições, tornada equivalente e até mesmo derivada de tais noções. Um ator que se encontrava isolado passa a fazer parte de um coletivo muito mais forte, sendo o seu ingresso uma das transformações que ampliam sua força. Gödel, contudo, colocava a máquina de Turing como o elemento *mais forte* de tal coletivo, o mais convincente. Apesar de todo o peso e influência que exercia na comunidade matemática, foi Gödel quem se viu isolado, pois a modalização que operou sobre a máquina de Turing, sua centralização, não foi retomada por outrem. Nos trabalhos posteriores a derivação da computabilidade em relação às outras derivações seria ainda mais reforçada.

O que Gödel encontrou na definição de Turing foi a presença da máquina como ator irredutível a outras definições, como a presença de um diferente modo de operação. Ainda que no campo da definição da computabilidade as máquinas de Turing tenham perdido sua autonomia conceitual, a máquina, enquanto ator distinto, não foi apagada em em outros domínios de ação. Veremos primeiro como a noção de máquina foi retomada no campo da lógica e, segundo, o papel que desempenhou na teoria de autômatos e linguagens formais.

2.7 Lógica, teoria de autômatos e linguagens formais

Fora a sua incorporação quase que imediata pelo grupo de Church, o artigo *On Computable Numbers* de Turing teve uma recepção fria, passando desapercebido tanto por lógicos e matemáticos, quanto por engenheiros. Os "engenheiros evitavam o artigo de Turing porque aparentava ser completamente teórico, e teóricos o evitavam por causa das referências a fita de papel e máquinas" (DYSON, 2012, p. 249).[xxix] A introdução de um novo ator em um campo que lhe é estranho não é uma tarefa fácil e, como vimos, foi possível apenas na medida em que a máquina pode ser convertida às outras estruturas matemáticas. Mesmo que Gödel tenha concedido uma especifidade à máquina perante outros atores matemáticos, foi seu pertencimento a uma classe de atores matemáticos já dados que a estabilizou.

Ainda que sua recepção tenha sido inicialmente fraca, nas décadas de 40 e 50 o trabalho de Turing começou a ser explorado, o que levou suas máquinas a se tornarem um *ponto de passagem obrigatório* em determinados campos de saber. A primeira exigência estabelecida por este ponto de passagem foi a equivalência de todas a variações de máquinas à concepção inicial de Turing. Nas décadas que se seguiram as máquinas de Turing foram concebidas das mais diversas maneiras, onde nenhum de seus componentes ficou sem modificação: de uma fita unidirecional passou a ter uma fita bidirecional, ou mais de uma fita; a pluralidade de signos que inscrevia se reduziu a marcas binárias e espaços vazios; seus estados foram reduzidos a dois, ou multiplicados; suas tabelas de instruções transformaram-se em diagramas, fluxogramas ou foram formalizadas; as instruções, que eram definidas com cinco termos, passam a ter quatro ou menos.

Qualquer um pode criar sua máquina de Turing, abrir a caixa-preta, redefinir algum de seus termos ou mecanismos, para então fechá-la de novo. Contudo, todas essas máquinas têm de cumprir com a obrigação de serem equivalentes, através de uma conversão, às máquinas que Turing definiu em seu artigo de 1936. Por mais que critiquem, desmontem e reorganizem as primeiras máquinas propostas por Turing, as novas máquinas devem se colocar como não sendo nada mais que seus desdobramentos. Em todos esses textos, encontramos alguma nota de rodapé referenciando o artigo de Turing, marcando seu pertencimento a tal linhagem. Tais máquinas podem ser mais eficientes, práticas, didáticas ou úteis a determinados fins do que as máquinas que Turing criou, mas não podem ser mais *poderosas*. No contexto da matemática e da lógica, a noção de poder refere-se ao que um sistema formal é capaz de fazer. Quanto mais poderoso, maiores são suas capacidades (HOFSTADTER, 1999). No caso das máquinas de Turing, se uma máquina fosse mais poderosa que a máquina definida por Turing, poderia computar números não-computáveis, o que é impossível para tal máquina. O poder de uma máquina define o seu *limite*, como os limites de uma atualização. A condição mínima de pertencimento à linhagem das máquinas de Turing é a preservação do limite da computabilidade[16].

Se falar de fitas e máquinas era inicialmente estranho a lógicos, logo se tornou um elemento indispensável de quase toda a descrição de uma máquina de Turing, sendo muitas vezes acompanhado por um desenho ou esquema da fita. Mesmo com essa exigência, a máquina de Turing foi rapidamente formalizada, como em Davis (1982b) e Kleene (1971 [1952]), sendo completamente incorporada em sistemas formais axiomáticos, com seus

16 O limite da computabilidade não impede que máquinas mais poderosas sejam concebidas, como as máquinas-oráculo (máquina-o) concebidas pelo próprio Turing, as máquinas pós-Turing, ou o campo da hipercomputação. (COPELAND, 2004; PETZOLD, 2008) Contudo, estas máquinas não serão trabalhadas neste capítulo.

teoremas, definições e derivações. Com o desenvolvimento dos primeiros computadores eletrônicos, a máquina que inicialmente era o modelo de um computador humano passou a ser o modelo do computador eletrônico, do mesmo modo que foi por este definida. Mesmo na lógica, em sua tendência a remover-se completamente à especifidade da materialidade viu-se referindo-se a computadores: "[as máquinas] serão definidas pela analogia com computadores físicos de um certo tipo" (DAVIS, 1982b, p. 3).[xxx]

A ideia de uma máquina universal, tal como foi desenvolvida por Turing, não desempenhou nenhum papel direto na criação dos primeiros computadores eletrônicos, nem nos que logo os seguiram, com exceção do ACE/Pilot ACE, com o qual Turing esteve diretamente envolvido. Os engenheiros utilizavam o termo "propósito geral" [*general purpose*] para referir-se à capacidade de um computador eletrônico executar qualquer tipo de cálculo ou operação simbólica, dentro dos limites de sua memória, velocidade e estrutura, o que corresponderia à noção de universalidade. Se a máquina de Turing, e em especial a máquina universal foi tomada como modelo do computador, não foi tanto como um guia para sua construção efetiva, não era um esquema para o desenvolvimento de *hardware*. Foi como um modelo do computador *na* ciência da computação que a máquina universal de Turing encontrou seu lugar, como objeto de estudo abstrato e não como princípio de uma técnica.

O campo da ciência da computação foi marcado por um nascimento conturbado, permanecendo dividido ainda hoje, pois constituiu-se pelo cruzamento entre pelo menos três vertentes: matemática/lógica, engenharia e ciência. Cada uma focava em aspectos distintos dos computadores e seus usos. Os matemáticos e lógicos preocupavam com o desenvolvimento de leis e princípios formais e gerais acerca da computabilidade. Os engenheiros dedicavam-se aos problemas relativos à construção de computadores, enquanto os cientistas colocavam questões relativas à aplicação de computadores nos mais variados domínios de pesquisa científica (TEDRE, 2015).

A máquina de Turing, assim como a própria figura de Turing, desempenhariam a função de unificar essas três vertentes ao redor de um objeto ou herói comum (BULLYNCK; DAYLIGHT; DE MOL, 2015). Contudo, a máquina de Turing, em sua concepção original, não era um modelo satisfatório dos computadores eletrônicos. Era necessário criar máquinas alternativas que dessem conta dessa discrepância entre o abstrato e o concreto, tal como foi proposto por Wang (1957):

> A teoria de Turing sobre funções computáveis antecedeu mas sem influenciar muito a construção atual e extensiva de computadores digitais. Estes dois aspectos de teoria e prática foram desenvolvidos quase que completamente independentes um do outro. Sem dúvida, a principal razão é que lógicos interessam-se por questões radicalmente diferentes daquelas com as quais matemáticos aplicados e engenheiros

> elétricos estão primeiramente interessados. Entretanto, não deve deixar de parecer um tanto estranho que frequentemente os mesmos conceitos são expressos por termos muito diferentes nos dois campos. É de se questionar se uma aproximação não poderia produzir um bom resultado. Acredito que este artigo será útil àqueles que que desejam comparar e conectar as duas abordagens (p. 63).[xxxi]

Para que tal conexão seja possível, Wang desenvolve um tipo de máquina de Turing, *máquina B*, cujo funcionamento se aproxima dos computadores eletrônicos digitais, o que produz uma máquina em que é mais fácil provar o que não pode ser feito, do que o que pode ser feito. Em todo caso, Wang propõe-se a traduzir a sua máquina em uma linguagem que seja familiar àqueles engajados na construção de computadores.

Nesta divisão entre engenheiros e teóricos temos dois grandes coletivos compostos pelos mais diversos atores e problemas. Do lado da engenharia encontramos as teorias de comutação de circuitos e máquinas sequenciais, que incorporavam a álgebra booleana, isto é, o cálculo binário baseado apenas em 0s e 1s (verdadeiro e falso), incorporado e traduzido nos pulsos e circuitos elétricos (ou eletro-mecânicos). O problema era como associar circuitos, elétrons, suas entradas e saídas e as sequências que formavam, incorporando milhares de componentes eletrônicos não confiáveis de modo a gerar um resultado confiável e com um mínimo de erro. Com a lógica, o problema dava-se em meio a máquinas infinitas, humanos abstraídos e os limites do computável.

Para Mahoney (2011), a álgebra booleana permitia descrever *como* os passos individuais de uma computação eram realizados, tornando possível sua incorporação em circuitos eletrônicos, mas sem abordar ou explicar a natureza de tais operações. A máquina de Turing, por ser infinita, permitia definir o que não podia ser feito, mas não era de grande ajuda em relação ao que poderia ser feito por uma máquina finita. O problema que a ciência da computação deveria dar conta era de como conciliar a finitude dos circuitos com a infinitude das máquinas de Turing, de como um caminho do meio poderia ser traçado.

A Teoria de Autômatos forneceu os termos necessários para estabelcer um relação entre os aspectos finitos e infinitos dos modelos teóricos. O texto fundador da disciplina, tal como é contada por seus praticantes, é o clássico artigo "*A logical calculus of the ideas immanent in nervous activity"* de McCulloch e Pitts (1943), no qual as redes neurais são modeladas abstratamente de modo a constituírem um cálculo lógico, uma formalização. Assim como Turing inseriu a problemática das máquinas na lógica, McCulloch e Pitts o fizeram em relação à atividade neuronal. Esta inserção se deu, contudo, em tal nível de abstração que o comportamento neuronal pode ser diretamente associado tanto às operações lógicas booleanas, pois o disparo de um neurônio segue o padrão de "tudo ou nada", dispara

ou não, tal como uma operação matemática que trabalhe com valores binários, 0 e 1. As redes formadas pelos neurônios poderiam replicar as operações fundamentais da lógica, como seus operadores "E" e "OU", o que permitia, também, que estes neurônios abstraídos servissem de modelo a circuitos eletrônicos.

O artigo de McCulloch e Pitts foi inspirado pelas máquinas que Turing apresentou em seu artigo de 1936, sendo que "cada rede, se lhe for fornecida uma fita, leitores conectados a aferentes, e eferentes apropriados a realizar as ações motoras apropriadas, pode computar apenas os números tal como uma máquina de Turing" (MCCULLOCH; PITTS, 1943, p. 17).[xxxii] Uma rede neural recebe suas entradas a partir de neurônios aferentes, e suas saídas dependem de neurônios eferentes. Deste modo, uma máquina de Turing, ou até mesmo um computador humano, funcionaria como um organismo com os órgãos externos apropriados à leitura escrita e manipulação de símbolos e fitas, sendo governado por um aparato neuronal em forma de rede, tal como o cérebro. A questão é como os neurônios organizam-se a partir das entradas e saídas, como as suas conexões são calculadas e seus valores de verdade obtidos, o que exigiu uma notação em formato de diagrama ou fluxograma para melhor expressar as conexões e fluxos, além da notação estritamente lógica.

Com Von Neumann (1956, 1963) os neurônios de McCulloch e Pitts, assim como as máquinas de Turing, tornam-se autômatos. O termo "autômato" refere-se tanto a seu sentido usual, como um mecanismo automático ou que segue ordens, quanto a uma " 'caixa-preta' com um número finito de entradas e um número finito de saídas. […] O funcionamento interno de tal 'caixa-preta' é equivalente à prescrição que especifica quais saídas serão estimuladas em resposta à estimulação de qualquer combinação dada de entradas, e também do tempo de estimação dessas entradas" (VON NEUMANN, 1956, p. 44–45).[xxxiii] O autômato, deste modo, corresponde ao mesmo tempo a um neurônio e a uma máquina de Turing, que recebe entradas e as computa de acordo com sua prescrição, produzindo uma saída, e formando uma rede de autômatos e suas conexões. Os autômatos são pensados nos dois sentidos, ora indo em direção a uma pura abstração formal, ora os colocando sob a égide de uma determinada materialidade biológica ou técnica. Neste registro material a máquina universal é pensada não mais como uma máquina que computa outra e produz símbolos em uma fita, mas como autômato que lê, ou "consome" uma descrição, produzindo um novo autômato. É uma máquina que além de computar, *produz* outra máquina, ou mais precisamente, é mesmo capaz de se auto-replicar. Do ponto de vista da lógica essa capacidade de auto-replicação tem uma consequência curiosa para o *Entscheidungsproblem*, pois "você pode construir um órgão que faz o que quer que possa ser feito, mas não pode construir um

órgão que diga o que pode ser feito ou não" (VON NEUMANN apud DYSON, 2012, p. 285).[xxxiv] Em outras palavras, um automato, capaz de replicar-se completamente, está sujeito ao limite da computabilidade definido por Turing, ainda que possa fazer muito mais do que pode explicar.

Os trabalhos de McCulloch, Pitts e Von Neumann foram trabalhados por Kleene em 1951, em um texto que circulou amplamente antes de ser publicado em 1956 (KLEENE, 1956). Kleene conseguiu estabelecer uma relação entre os autômatos e a álgebra booleana ao mostrar que se dois conjuntos de sequências de letras podem ser reconhecidas por um autômato, então estão sujeitas às mesmas operações numéricas que são próprias à álgebra booleana. Tais conjuntos foram chamados de "eventos regulares" (MAHONEY, 2011).

A publicação do texto de Kleene se deu no livro *Automata Studies*, organizado por Shannon e McCarthy (1956), com a participação dos principais pesquisadores do campo, como John Von Neumann, Marvin Minsky, Ross Ashby e Martin Davis. Turing foi convidado em 1953 a participar da publicação, mas recusou pois no momento estava dedicado a seus estudos de biologia e morfogênese, vindo a falecer no ano seguinte. A publicação do livro esteve diretamente ligada à organização da Conferência de Dartmouth sobre Inteligênia Artificial, considerada como o local de nascimento da Inteligência Artificial, sendo que a maioria dos textos do livro abordavam temas centrais da cibernética e da inteligência artificial. (KLINE, 2011)

O livro fora organizado em três partes: Autômatos Finitos; Máquinas de Turing; e Sínteses de Automatos. Esta divisão ressaltava a diferença entre autômatos e máquinas de Turing, o que naquele momento já era algo corrente. Ainda que sejam equivalentes a autômatos, as máquinas de Turing são autômatos *infinitos*, enquanto os trabalhos de McCulloch, Pitts, Von Neumann e Kleene concerniam a autômatos *finitos*. Reencontramos aqui a problemática entre a infinitude abstrata das máquinas de Turing e a finitude dos elementos que deveria representar. A diferença é que as máquinas de Turing são deslocadas e modalizadas, e passam a ser uma *classe* de autômatos. Os autômatos finitos, por sua vez, gozam de todas as propriedades das máquinas de Turing, com exceção dos casos em que a finitude da memória estaria envolvida.

Rabin e Scott (1959) afirmam de modo claro porque o autômato finito seria um modelo mais apropriado do computador eletrônico digital:

> Máquinas de Turing são amplamente consideradas como o protótipo abstrato de computadores digitais; praticantes do campo, contudo, têm sentido cada vez mais que a noção de uma máquina de Turing é geral demais para servir como um modelo preciso de computadores atuais. É bem conhecido que para simples cálculos é impossível dar um limite máximo *a priori* para a quantidade de fita que uma

> máquina de Turing precisará para qualquer computação dada. É precisamente esta propriedade que torna o conceito de Turing irrealístico.
> Nos últimos anos a ideia de um *automato finito* tem aparecido na literatura. Estas são máquinas que possuem somente um número finito de estados internos que podem ser utilizados para memória e computação. A restrição de finitude aparentemente possibilita oferece uma aproximação melhor à ideia de uma máquina física. Naturalmente, tais máquinas não podem fazer tanto quanto uma máquina de Turing, mas a vantagem de ser capaz de computar uma função recursiva geral arbitrária é questionável, dado que pouquíssimas destas funções aparecem em aplicações práticas (RABIN; SCOTT, 1959, p. 114).[xxxv]

Neste texto, que tornou-se umas das principais referências do campo de estudo de autômatos (MAHONEY, 2011), Rabin e Scott retomam o modelo de autômatos finitos de Kleene enquanto tentam conservar o formalismo que é próprio às máquinas de Turing. A infinitude é deixada de lado, o que implica que determinados problemas matemáticos não possam ser abordados. Turing deve de conceber as suas máquinas em termos infinitos para poder dar conta do *Entscheidungsproblem*, já que este referia-se a um conjunto arbitrário e ilimitado de questões. Na prática do programador, contudo, a memória, e os erros de programação ou *hardware* surgem muito antes que qualquer problema da ordem absoluta do *Entscheidungsproblem* apareça. O tipo de limitação que os autômatos finitos apresentam é muito mais útil para compreender as limitações efetivas dos computadores digitais, ao invés de seus limites teóricos. Por outro lado, autômatos finitos, assim como máquinas de Turing, são completamente determinísticos, ou seja, a mudança de um estado a partir de uma determinada entrada segue um trajeto necessário e totalmente determinado. Rabin (1963) propõe o modelo de um "autômato probabilístico", cujo comportamento seria apenas relativamente determinado, o que refletiria o fato de que computadores concretos estão sempre sujeitos ao erro, isto é, à indeterminação.

A problemática da finitude será trabalhada por Chomsky (1959) no contexto de definição de gramáticas universais e linguagens formais. "Uma gramática pode ser considerada como um dispositivo que enumera as sentenças de uma linguagem. Estudaremos as sequências de restrições que limitam gramáticas primeiro em máquinas de Turing, […] e finalmente em fontes de estado finito de Markov (autômatos finitos)" (CHOMSKY, 1959, p. 137).[xxxvi] Uma gramática, tal como um automato finito, é um sistema de regras que produz um número infinito de sequências, a partir de um alfabeto ou vocabulário limitado. A produção de sentenças, ainda que seja potencialmente infinita, encontra-se limitada por diversos fatores. Primeiro, há a separação entre sentenças válidas e inválidas em uma determinada língua, isto é, entre aquelas consideradas "naturais" ou gramaticais, e aquelas cuja sintaxe não faz sentido, para um falante nativo da lingua. Segundo, existem sentenças cuja estrutura é perfeita mas que nunca serão pronunciadas, enquanto outras são recorrentes e redundantes. Ao modelar

uma gramática como um autômato finito, Chomsky permite que as sentenças, e seu modo de produção, sejam concebidos como uma rede intrincada de permutações, entradas, saídas e mudanças de estados, onde cada estado corresponde a um elemento do alfabeto. A relação estabelecida entre autômatos finitos e gramáticas se dá nas duas direções, pois permite que autômatos sejam definidos *por* e *como* gramáticas. Correlativamente, a máquina de Turing passa a ser uma *linguagem*. Esta relação entre autômatos e máquinas com linguagens formais terá uma implicação direta no desenvolvimento de linguagens de programação, como será visto no próximo capítulo (HOPCROFT; ULLMAN, 1969).

Esta longa exposição do desenvolvimento da teoria de autômatos compõe apenas uma pequena parte de um campo extremamente vasto e complexo, sendo que deixamos de lado seus desdobramentos mais profundos nos mais variados campos, como matemática, lógica, teoria da comunicação, linguística e engenharia. Ainda que um tanto sumário, podemos ver toda a variedade de atores que foram criados e inseridos, e seus efeitos no agenciamento dos mais variados coletivos. Primeiro há a introdução do neurônio, cujo surgimento faz com que a lógica tenha de ser modificada para dar conta do modo de funcionamento próprio a redes neurais. Os neurônios são logo transformados em autômatos, sendo descritos em termos de caixas-pretas, entradas e saídas e máquinas que se auto-replicam. Em paralelo, as máquinas de Turing tornam-se uma subclasse de autômatos, sendo igualmente associadas à neurônios e redes. Na lógica são retrabalhadas de modo a se aproximarem da linguagem dos construtores de computadores. Os autômatos são retomados e modalizados como entidades finitas e probabilísticas, permitindo que sejam associados ao computador digital, associação que lhes dá mais força por construir uma ponte com um determinado tipo de referência. Por fim, é a própria linguagem que é modificada ao ser transformada em um conjunto de relações entre autômatos finitos ou máquinas de Turing, o que os torna, também, um tipo de linguagem formal.

Podemos dizer que todas estas transformações se deram de acordo com dois vetores de movimento. O primeiro as leva em direção a uma abstração e formalização cada vez maior, produzindo atores como agentes infinitos ou completamente desprovidos de corporeidade, como descritos por Rotman (1993). O segundo movimento direciona as máquinas e autômatos em direção à materialidade dos organismos e computadores digitais. Encontramo-nos a uma certa distância da formulação inicial de Turing, cuja referência à materialidade era dada por um computador humano, ou por determinados elementos da máquina. Se a máquina de Turing se tornou um ponto de passagem obrigatório não foi apenas porque colocou a exigência de equivalência formal entre todos esses atores, mas porque indicou a necessidade de se pensar

algo que seja específico à máquina e o autômato, isto é, introduziu no pensamento abstrato a exigência de uma referência constante a uma certa concretude ou corporeidade. Essa exigência de não abandonar uma certa referência à máquina concreta é colocada por Rabin, que afirma que

> qualquer que seja a generalização da concepção de automato finito que desejamos considerar, deverá ser baseada em alguma noção intuitiva sobre máquinas. Este é provavelmente o melhor guia para frutíferas generalizações e problemas, e nossa intuição acerca do comportamento de máquinas será útil em conjecturar e provar teoremas (RABIN apud MAHONEY, 2011, p. 169).[xxxvii]

Retomando a questão de Agar (2003), de como a máquina universal de Turing pode se tornar o modelo do computador eletrônico digital, podemos dizer que o fez apenas na medida em que se modificou, transformou e estabeleceu alianças. O que a breve e fragmentária história que este capítulo narra é como cada vez mais a máquina universal foi aproximada de outras máquinas, de neurônios, autômatos e circuitos de maneira que se tornasse mais próxima ao computador digital. A máquina universal teve de primeiro transformar o computador humano para poder constituir-se enquanto equivalente do gesto humano de calcular, sendo por sua vez modificada para ser equivalente ao computador digital.

Este capítulo teve como foco o desenvolvimento da máquina universal no âmbito do pensamento e da prática da matemática e lógica, e seus desdobramentos em uma nascente ciência da computação através da teoria dos autômatos. Talvez o maior deslocamento operado pela máquina de Turing e pela máquina universal não tenha sido tanto a sua capacidade de modelar o computador eletrônico, mas por ter inserido no pensamento lógico e matemático um modo próprio de pensar acerca de máquinas. Foi Descartes quem, de acordo com Simondon (2007), introduziu as máquinas no pensamento matemático através da mecânica racional. Turing, contudo, introduz as máquinas sem contudo reduzi-las à linguagem matemática, dando margem a uma linguagem própria à máquina.

No próximo capítulo veremos como essa dualidade da máquina, ao mesmo tempo abstrata e concreta, matemática e algorítmica, terá um impacto no campo da programação e na constituição da figura do programador. Se no domínio da matemática e lógica esta coexistência foi, de certo modo, pacífica, quando os computadores eletrônicos entram de fato em cena, com toda a sua materialidade unida às exigências acadêmicas e da nascente indústria do *software*, a distinção entre máquina e lógica se transformará em uma fratura fonte de diversas controvérsias. A universalidade deixará de ser um fato dado, e até óbvio, para ser um operador de disputas acerca da natureza das linguagens de programação e da própria atividade do programador. É neste âmbito, no cruzamento do computador, do programador e do código

que a máquina universal assumirá um papel mais ativo, tendo as agênia ampliada para além de uma mera classes de máquinas de Turing ou autômatos infinitos.

3 Programa

Por um breve instante, estamos juntos num universo onde dois seres humanos podem entender simultaneamente a afirmação "se o espaço for numérico!".

- Ellen Ullman

3.1 Uma questão de tradução

Em 1976, Knuth e Trabb Pardo (1980 [1976]) publicam o levantamento que fizeram sobre 21 linguagens de programação criadas entre 1945 e 1958. É uma seleção heterogênea de linguagens, desde as que foram apenas projetadas mas nunca implementadas (como "*Plankalkül*" de Zuse ou os "*Flow Diagrams*" de Von Neumann e Goldstine), até aquelas que ganharam ampla adoção (como FORTRAN, que ainda hoje é extensivamente utilizada na pesquisa científica), incluindo também linguagens de programação russas, desconhecidas no ocidente. Boa parte da pesquisa foi baseada em material não publicado.

A grande variedade de notações e modos de funcionamento das linguagens, assim como sua falta de documentação, deixa claro o caráter eminentemente experimental do desenvolvimento de computadores digitais e linguagens de programação nas duas primeiras décadas após a segunda guerra mundial. Para comparar e apresentar a especifidade de cada linguagem, os autores adotaram como metodologia a implementação de um mesmo algoritmo, inicialmente escrito em um dialeto da linguagem ALGOL 60, em cada uma das linguagens selecionadas.

Mais do que a descrição individual da implementação em cada linguagem, ainda que seja muito interessante, o que nos chama a atenção é precisamente o que torna possível o procedimento adotado pelos autores – a ideia de que uma única linguagem de programação pode ser traduzida e implementada nas mais variadas máquinas e linguagens em contextos completamente heterogêneos. A grande profusão de linguagens de programação na década posterior a 1945 pode ser atribuída ao fato de que cada computador utilizava a sua própria linguagem. Todo código escrito para um computador poderia ser utilizado por outro apenas se fosse adaptado, tarefa extremamente trabalhosa quando era possível.

O esforço das primeiras décadas após o surgimento do computador eletrônico foi

direcionado para a construção das máquinas. A atenção dirigia-se ao feito que era construir um computador que simplesmente funcionasse, trabalho ao mesmo tempo experimental e artesanal. A cada máquina correspondia uma arquitetura, um modo singular de computar ao nível do *hardware* e ser programada ao nível do *software*. "Porque cada máquina era única [o programador] sabia muito bem que seus programas possuíam uma significação apenas local, e também porque era patentemente óbvio que a máquina teria uma duração de vida limitada, sabia que muito pouco de seu trabalho teria um valor duradouro" (DIJKSTRA, 1972a, p. 860).[xxxviii] Um programa possuía uma existência volátil, que ainda não fora estabilizada. O fato de Knuth e Trabb Pardo terem de trabalhar em grande parte com material não publicado se deu, possivelmente, pelo período de vida curto das linguagens, que eram suplantadas e esquecidas com o surgimento de novas máquinas.

Ainda que todo computador digital opere em seu nível mais fundamental com uma notação binária, 0s e 1s implementados fisicamente, são inúmeras as formas de construir e computar uma sequência binária (assumindo que uma base binária seja utilizada). As linguagens de programação constituem modos de traduzir a escrita binária, de "baixo nível", em conjuntos de instruções de "alto nível" que sejam mais facilmente compreensíveis e legíveis a um ser humano. Apesar desta tradução, as linguagens de programação ainda dependem da arquitetura de máquina subjacente. Isto fica evidente quando Knuth e Trabb Pardo têm de converter o algoritmo TPK, escrito em uma variante da linguagem ALGOL 60, para as outras linguagens. O algoritmo calcula um determinado valor a partir de um número que lhe é fornecido, escrevendo o valor fornecido e o valor calculado, ou o valor fornecido e a mensagem "TOO LARGE" caso o valor calculado seja maior que 400. Para que a tradução seja possível é necessário que o algoritmo seja modificado. Uma linguagem pode não possuir uma saída alfabética, então a expressão "TOO LARGE" será escrita como "999", ou, em outro caso, a linguagem não trabalhará com números decimais, sendo necessário arredondá-los para números inteiros, enquanto certas linguagens sequer trabalham com a noção de entrada ou saída.

Para os autores é completamente claro que a conversão é um processo opaco, que não há uma tradução cristalina e perfeita. Afirmam mesmo a necessidade de deslocamento e transformação inerentes a toda tradução. É na medida em que as transformações assumem o primeiro plano do texto que as diferenças e incomensurabilidades das linguagens de programação podem ser colocadas em evidência. Se as peculiaridades do processo de tradução não passam desapercebidas deve-se ao fato de que os autores operam em um momento em que a tradução entre linguagens é considerada como algo dado, elemento

subjacente à prática da programação, com todos os problemas e dificuldades que lhe se não inerentes, o que se encontra implicado na utilização de um dialeto da linguagem ALGOL 60.

A linguagem ALGOL foi a primeira linguagem de programação a ser desenvolvida com o intuito expresso de ser geral e independente de referência a qualquer máquina específica, o que a levou a ser concebida por muitos como uma linguagem universal, ainda que seu domínio se restringisse à computação numérica (DAYLIGHT, 2012; MACKENZIE, 2004). A independência da máquina implicaria na possibilidade de um programa ser escrito e executado em qualquer computador capaz de executar programas na linguagem em questão. Bastaria que a máquina possuísse um compilador ou interpretador, programas que traduziriam a linguagem do programa para o código nativo da máquina, tanto antes da execução, no caso do compilador, quanto durante o processo, no caso dos interpretadores.

ALGOL não foi a primeira linguagem a propor esta independência, as linguagens FORTRAN e COBOL também o fizeram, mas não obteve a ampla adoção da qual estas linguagens gozaram. Seu mérito encontra-se no fato de se ter constituído em uma grande influência para o desenvolvimento de linguagens futuras, especialmente por ter colocado a questão da generalidade em primeiro plano (PRIESTLEY, 2011). É aqui que reencontraremos a ideia de máquina universal de Turing, que, ao ser transposta para o domínio da programação, não constituirá mais um modelo de máquina, mas justamente o modo de funcionamento que permitirá conceber uma independência da máquina. Mais precisamente, a máquina universal será concebida como um modelo de tradução entre linguagens, do mesmo modo que uma máquina de Turing deve ser traduzida em código para ser executada em uma máquina universal, uma linguagem universal pode ser traduzida para o código de máquina onde será executada.

Ainda que durante toda a década de 1950 a noção de uma linguagem de programação universal tenha sido defendida e proposta, a implementação e desenvolvimento da linguagem ALGOL constituiu uma fonte de controvérsia, que será abordada na primeira metade deste capítulo. Na lógica e nas teorias que tomaram as máquinas de Turing como seu objeto, a universalidade nunca foi questionada. A máquina universal teve seu lugar garantido como um tipo de autômato finito ou infinito, variando apenas nos modos de construí-la, mas nunca sendo colocado em questão o *por quê* de construí-la. No caso das linguagens de programação, a universalidade encontrava-se em oposição à eficiência. À medida em que uma máquina universal pode imitar qualquer máquina, ou que um programa pode ser traduzido em qualquer outra linguagem, há um custo em eficiência na execução de tal programa. Enquanto uma máquina ou código de máquina específica executa suas instruções diretamente, uma máquina

ou linguagem executada por outra deve primeiro ser lida e interpretada, exigindo mais operações a cada comando.

A eficiência dependia, então, de levar em conta as especifidades de cada máquina, ao custo de produzir uma linguagem limitada e repleta de exceções. A generalidade permitiria o desenvolvimento de uma linguagem mais confiável e menos sujeita a erros. A questão que se colocava era da viabilidade prática da segunda proposta. Por um lado, os defensores da generalidade afirmavam que logo, com o desenvolvimento de computadores mais poderosos, a eficiência não seria mais um problema, enquanto que por outro lado se colocaram a construir compiladores viáveis (DAYLIGHT, 2012).

As duas posições não representavam apenas propostas técnicas distintas, mas eram atravessadas, em maior ou menor grau, por diferentes *ethos*. A busca pela eficiência incorporava uma ética focada em produzir resultados efetivos, de levar em conta a especifidade de cada máquina para aliar-se com as forças que lhe são próprias. Podemos dizer que é uma ética pragmática. A generalidade, por sua vez, ocupava um campo mais teórico, apoiando-se em valores como a formalização e elegância da linguagem, o que refletia o desejo de produzir programas universalmente corretos, apelando antes para um modo de pensamento mais matemático e abstrato do que prático.

À medida em que as linguagens de programação adotavam o modelo de universalidade e generalidade proposto por ALGOL, com a controvérsia sendo resolvida em favor da generalidade, a oposição entre as duas éticas agravou-se. A dicotomia tornou-se mais evidente quando computadores mais poderosos foram eventualmente desenvolvidos, culminando na chamada "crise do software de 1968". Enquanto os computadores tornavam-se mais rápidos e potentes, o desenvolvimento de software era cada vez mais afetado por atrasos, orçamentos estourados e, principalmente, *bugs*. Os proponentes de uma perspectiva mais formal, teórica e matemática da computação adotavam a posição de que todo *software* deveria ser acompanhado de provas matemáticas, pois esta seria a única forma de garantir a ausência de *bugs*. A eles opunham-se aqueles que acreditavam que a solução para a crise se daria tanto pelo desenvolvimento de melhores ferramentas para auxiliar os programadores, quanto pelo desenvolvimento de novas técnicas de gerenciamento. O que estava em jogo era a própria figura do programador e a natureza de sua prática (MACKENZIE, 2004).

Veremos como, na década de 1980, o aparecimento do movimento do *software* livre retoma o problema da universalidade colocando-a em termos éticos e políticos que, em grande parte, inverte e desloca as posições anteriores. Se a universalidade foi assumida por uma perspectiva apoiada em uma ética de formalização, em detrimento de um foco na prática, com

o software livre a universalidade torna-se a base de uma *ética hacker* cujo principal valor é a liberdade. A questão não se restringe mais à possibilidade da universalidade de execução de programas, mas engloba também o acesso a seu código-fonte, a possibilidade de sua modificação e compartilhamento. A universalidade não é mais o que se opõe à especifidade da modificação e adaptação do código à condições particulares, mas é justamente o que permite tal prática, ao mesmo tempo em que constitui um meio de circulação que faz com que tais códigos singulares sejam retomados e reincorporados a uma comunidade. Neste capítulo realizaremos uma breve exploração histórica dos temas e problemas recém expostos, sendo que no capítulo 5 discutiremos de modo mais aprofundado e filosófico acerca da natureza do código, sua relação com algoritmos e linguagens de programação.

3.2 Do código à linguagem

A arquitetura de um computador, similar à arquitetura de um prédio, não se refere especificamente a seus componentes eletrônicos e físicos, sendo um modelo mais abstrato que concerne à organização e ao modo de funcionamento da máquina. Situa-se em um nível intermediário entre *hardware* e *software*, pois especifica as funções que o hardware deve implementar, enquanto define as operações que serão acessíveis ao *software* (DORAN, 2005). Uma das arquiteturas mais básicas e que ainda pode ser considerada como base das arquiteturas contemporâneas, divide o computador em quatro elementos: entrada/saída, memória, unidade aritmética e controle. Nenhuma dessas partes corresponde a um componente eletrônico singular, cada uma pode ser implementada das mais diversas maneiras. A memória, por exemplo, pode ser composta por tubos de mercúrio, tambores magnéticos, memória de ferrite, discos rígidos ou de estado sólido. Dois computadores com *hardware* completamente distintos podem apresentar a mesma arquitetura (SMITH, 1989).

Na arquitetura citada, o computador recebe um dado pela entrada, que é então armazenado em sua memória. Em um computador de programa armazenado, os dados que se encontram na memória podem constituir o próprio programa a ser executado, isto é, são instruções que serão lidas pelo controle, que decidirá a partir das instruções recebidas o que fazer. Se for necessário realizar algum tipo de operação matemática ela será repassada à unidade aritmética, que devolverá o dado à unidade de controle. Uma vez que a computação tenha sido realizada ela é novamente registrada na memória, podendo ser passada para a saída.

Este modelo básico de arquitetura foi proposto inicialmente por Von Neumann em 1945 no "*First draft of a report on the EDVAC*" (VON NEUMANN, 1945), documento que não foi publicado mas que obteve ampla circulação entre os interessados na construção de computadores. Junto com Goldstine e Burks, Von Neumann publica "*Preliminary Discussion on the Logical Design of an Electronic Computing Instrument*" em 1946 (VON NEUMANN; GOLDSTINE, 1955), no qual ampliam as ideias apresentadas no *First Draft* e cuja circulação foi ainda mais ampla. A arquitetura proposta nestes dois documentos e que ainda é considerada como a base das arquiteturas atuais veio a ser chamada de "arquitetura de Von Neumann" (SMITH, 1989).

Além da organização que citamos, uma arquitetura poderia ser composta por muitos outros elementos: unidade aritmética serial ou paralela, com base binária ou decimal; canais de entrada e saída, *buffers* de memória, interruptores, memória indexada, microprogramação, entre outros. Na década de 50 existia uma divisão entre computadores de negócio, que utilizavam uma base decimal por ser mais familiar aos dados utilizados pelas empresas e por sua compatibilidade com cartões perfurados, enquanto os computadores de pesquisa utilizavam uma base binária, que era mais rápida e não era estranha à prática acadêmica e científica.

Do ponto de vista do *software*, a arquitetura definia também o *conjunto de instruções* disponível ao computadores.

> Para realizar uma computação o computador deve ser programado; o conjunto de instruções é a única linguagem em que isso pode ser realizado. Inicialmente, cada computador disponibilizava um conjunto único de instruções. Por exemplo, o primeiro protótipo do Manchester Mark I disponibilizava a subtração como única operação aritmética. Computadores típicos forneciam três ou quatro das operações aritméticas básicas. Programadores trabalhando com um computador sem a operação necessária teriam de substituir a sequência de instruções para obter o efeito da instrução ausente. Portanto, um grande e bem projetado conjunto de instruções frequentemente simplificaria o trabalho dos primeiros programadores (SMITH, 1989, p. 284).[xxxix]

Os primeiros computadores eram criações experimentais. Ninguém sabia ao certo como um computador deveria ser construído. Todas as técnicas e componentes que hoje nos são tão familiares ainda estavam por serem inventados. Deste modo, cada computador possuía uma arquitetura única ou ao menos um conjunto de instruções que lhe era singular. Isto tornava a prática da programação específica a cada computador, exigindo um retrabalho considerável para adaptar um programa a outra arquitetura, quando era possível.

O conjunto de instruções é composto por um *código de máquina*, em que cada instrução é representada por uma sequência numérica e constitui uma ação simples, como mover um número de um lugar para outro na memória, somar, subtrair, comparar, pular de

uma instrução a outra etc. Definir as instruções neste nível é chamado de *codificação* [*coding*] e constitui o nível mais baixo e elementar de programar um computador, onde os comandos são escritos diretamente como serão executados pela máquina (PRIESTLEY, 2011).

Nos anos 40 ainda não existia, propriamente, a ideia de um programa ou a figura do programador. O *software* não era considerado um elemento importante do computador, consistia em uma etapa menor e irrelevante do problema muito mais importante e difícil de construir computadores eletrônicos que funcionassem consistentemente. No caso do ENIAC, o planejamento e execução da computação a ser realizada implicava em uma divisão de trabalho e, também, uma divisão de gênero. No topo, haveria um planejador, quase sempre um homem, que definiria qual o problema e algoritmo para sua resolução. Na etapa final o algoritmo seria traduzido para o código por seis mulheres[17] que desempenhariam a função de *coders*. O código, no caso do ENIAC, era operado primeiro pela ligação direta de cabos em um painel e posteriormente com cartões perfurados. O trabalho da *coder* era visto como uma mera extensão da atividade de um computador humano, função desempenhada em sua grande maioria também por mulheres. Deste modo, as mulheres cumpririam uma função subalterna e mecânica enquanto todo o planejamento seria quase que exclusivo aos homens que ocupavam cargos mais altos. Na prática, contudo, quase todo o trabalho de codificação e planejamento acabou ficando na mãos das *coders*, pois cada vez mais se viu que o trabalho de programar não era nem um pouco trivial, especialmente no que se referia à busca e correção e depuração de erros (*debugging*). A programação apresentava uma lógica e uma dinâmica própria que dependia de uma prática situada (ENSMENGER, 2010).

A codificação constituía um trabalho complexo e cognitivamente difícil, as instruções em sua forma numérica não continham nenhuma informação acerca da operação que realizavam. Em alguns casos, como foi utilizado no computador EDSAC e que serviu de base para outros sistemas, as instruções numéricas eram substituídas por algum tipo de abreviação simbólica, como letras ou palavras. O programa escrito desta maneira precisaria ser traduzido para o código de máquina antes de ser executado, o que era feito à mão ou, como veio a ser mais comum, pela própria máquina. Ainda que o uso de referências simbólicas auxiliasse na compreensão do código, o conjunto de instruções permanecia inalterado, mantendo uma correspondência unívoca entre cada termo (PRIESTLEY, 2011).

Na década de 50 o processo de codificação começou a distinguir-se do que se chamava de "programação". Enquanto a codificação era uma atividade basicamente

17 Kathleen McNulty, Frances Bilas, Betty Jean Jennings, Elizabeth Snyder Holberton, Ruth Lichterman e Marlyn Wescoff.

mecânica, focada mais nas necessidades da máquina do que do usuário, a programação voltava-se à resolução de problemas específicos, que neste momento referiam-se em grande parte a problemas matemáticos e numéricos. A expressão de uma fórmula ou um problema escrito na notação matemática para o código de máquina exigia um trabalho considerável, dada a necessidade de traduzir as operações matemáticas nas instruções muito mais elementares da arquitetura em questão (por exemplo, uma exponenciação deveria ser escrita ao nível do código como uma série de multiplicações ou até mesmo adições). Seria muito mais fácil se o programa pudesse ser escrito diretamente na notação matemática, ou em uma aproximação.

Propôs-se então uma forma de *codificação automática* em que o programador escreveria o programa em uma *linguagem de programação* que seria traduzida pela própria máquina em código. O importante é que não haveria uma correspondência direta entre linguagem e código, permitindo que uma instrução escrita em uma linguagem de programação, o chamado código-fonte, fosse traduzida em diversas instruções do código de máquina, o que pouparia muito trabalho ao programador, multiplicando sua eficiência. Contudo, o processo de tradução exigia tempo da máquina também, podendo se dar com o uso de *interpretadores* ou *compiladores*. Um interpretador traduz o código-fonte na medida em que o lê, o que acarreta em um aumento no tempo de execução do programa, já que cada instrução deve ser interpretada além de ser executada. Um compilador realiza a tradução antes da execução do programa, o que pode ser um processo demorado. O interpretador permite que o código seja executado imediatamente, a custo de performance, enquanto o compilador permite uma execução com boa performance a custo de uma espera inicial (PRIESTLEY, 2011).

Como o trabalho de Knuth e Trabb Pardo (1980 [1976]) mostrou, nas décadas de 40 e 50 houve uma imensa proliferação de linguagens de programação, desde as que se limitavam a abreviações simbólicas até as que implementavam funções de alto nível e de grande complexidade, como a linguagem FORTRAN. Essa proliferação se deu sobretudo pela grande variedade de arquiteturas de computadores, o que refletia o caráter experimental da época. Certas arquiteturas exigiam linguagens muito específicas e diretamente atreladas ao seu modo de operação e peculiaridades de implementação eletrônica. Todavia, o desenvolvimento de linguagens de programação permitiu, ao mesmo tempo, uma certa independência do código de máquina e abriu a possibilidade para o compartilhamento de código-fonte, isto é, o programa escrito em uma linguagem de alto nível.

Pensando em termos de coletivos, podemos dizer que cada arquitetura constitui um

modo distinto de agenciamento de atores. Cada computador é, na verdade, uma pequena sociedade, ou mais precisamente, um grupo, com suas regras, convenções, rituais de entrada. Além de todas as restrições que, neste momento, impediam o acesso a um computador, como a necessidade de fazer parte de um grupo de pesquisa nas poucas universidades, institutos ou empresas que estavam desenvolvendo computadores, a articulação *com* o computador dependia das singularidades do coletivo que formava. Quando falamos de computadores referimo-nos então a um agenciamento de atores humanos e não-humanos: circuitos eletrônicos, códigos de máquina, arquiteturas lógicas, programadores, codificadores, gerentes de projetos, compiladores, interpretadores. Especificamente no caso da programação, esta se dava como um ato compartilhado com a máquina, que junto ao programador também lia, escrevia, interpretava e traduzia.

Para utilizar outra analogia, cada computador existia como uma mônada no sentido leibniziano. Ainda que fosse uma máquina de propósito geral, ou máquina universal, era um universo fechado em si mesmo, que não se comunicava com outras mônadas. Cada máquina encontrava-se isolada e limitada em relação aos aliados que poderia arregimentar. A introdução de compiladores e interpretadores, contudo, permitiu que essas mônadas se abrissem a um universo maior.

A linguagem FORTRAN, desenvolvida entre 1954 e 1957 inicialmente para computador IBM 704, foi uma das primeiras linguagens a possuir um compilador eficiente. O grande problema das linguagens que a precederam era a perda de eficiência causada pelo processo de tradução (CAMPBELL-KELLY, 2003). Ainda que o código fosse compilado, isto é, convertido antes da execução do programa, a tradução não era idêntica à realizada por um humano. Isto se dá porque as linguagens de programação, assim como qualquer linguagem natural, estão sujeitas a ambiguidades. Uma mesma expressão pode ser traduzida de diversas maneiras ao nível do código de máquina, com diferentes implicações para sua eficiência. O compilador deve tentar resolver qual o melhor caminho a ser tomado, de acordo com pressupostos programados. Um compilador eficiente é programado para resolver melhor as ambiguidades e a tradução de um código para outro.

Logo foram criados compiladores de FORTRAN para outros computadores e a linguagem tornou-se o padrão da indústria fazendo com que, inclusive, competidores da IBM vendessem computadores com compiladores de FOTRAN para serem competitivos. Mais precisamente, na década de 1960, FORTRAN tornou-se a linguagem mais utilizada para a pesquisa científica (o que se mantém até hoje), enquanto a linguagem COBOL foi adotada pelas aplicações comerciais. Esta diferença se dava porque FORTRAN era uma linguagem

focada no cálculo numérico, e escrita com fórmulas matemáticas (daí seu nome, acrônimo de "Formula Translator"), e que se adequava mais à pesquisa científica em que era necessário alimentar a máquina com dados numéricos e fórmulas, obtendo um resultado ao final. Em aplicações de negócios, por suas vez, os dados não eram necessariamente numéricos, sendo necessário computar valores textuais, e os programas também manipulavam bases de dados onde determinadas informações deveriam se manter de modo persistente, exigências às quais a linguagem COBOL atendia (CAMPBELL-KELLY, 2003).

Apesar de sua popularidade, nem FORTRAN nem COBOL foram projetadas para serem linguagens universais. Sua dispersão e sucesso podem ser atribuídos a uma série de fatores, mas não ao fato de terem sido criadas para tal fim. Entretanto, esta ainda não era a situação ao final da década de 1950, quando estas linguagens haviam sido recém inventadas. Em paralelo a seu desenvolvimento, existia a preocupação por parte de certos pesquisadores em desenvolver linguagens feitas para serem independentes de qualquer máquina, o que permitiria aos programadores se preocuparem exclusivamente com os problemas que deveriam resolver do que com os detalhes e peculiaridades do *hardware*.

> A lição que usuários de computadores digitais têm aprendido durante os últimos dez anos é que, estranhamente, o principal uso para estas novas máquinas de informação talvez não seja ao todo como computadores, mas como "manipuladores de símbolos".
> É como um manipulador simbólico – avaliando e interpretando símbolos de seu próprio meio – que computador digital pode ser capaz de realizar suas tarefas mais espetaculares.
> Todavia, antes que o usuário da máquina possa fazer uso desta lição, deve adotar uma "vista de fora" do computador digital. Deve ser tornar muito mais orientado ao problema do que ao equipamento. Deve compreender que é o método de descrição do problema que é importante e não o *hardware* físico ou até mesmo a estrutura lógica em que o *hardware* está interconectado.
> Como um usuário descreve um problema? Geralmente, desenvolve uma correspondência ou mapeamento entre elementos e relações do mundo real e um conjunto de símbolos que ele, e posteriormente a máquina, manipulará e agirá sobre (CARR III, 1958, p. 21).[xl]

A independência em relação à máquina não representa apenas uma economia na escrita de programas, mas consiste em um modo completamente diferente de conceber o funcionamento do computador digital, menos como uma calculadora e mais como um manipulador de símbolos. Esta concepção do computador é ligada por Carr à máquina universal de Turing.

A máquina universal, ainda que tenha sido transformada no modelo abstrato do computador, não foi necessária para a criação de um *hardware* de propósito geral. Para os construtores de *hardware* o problema não era tanto se a máquina que construíam era universal, mas o quão eficiente poderia ser ou qual a melhor arquitetura para o tipo de tarefa que iria realizar. Se na teoria da computação todos os computadores pudessem ser

considerados como equivalentes a uma máquina universal, na prática eram universos únicos, tal como mônadas.

> Não há um computador, mas apenas computadores. Até na forma mais geral de uma máquina de Turing finita, o computador é um esquema ao invés de um dispositivo singular. O que pode fazer depende da tabela de estado finito e dos conteúdos da fita. Em sua forma mais geral, ou universal, pode fazer qualquer coisa para as quais possamos prover instruções apropriadas, isto é, a tabela para uma máquina de Turing específica. Esta é a fonte de seu poder, é uma máquina proteana. Entretanto, precisamente porque pode fazer tudo, não é capaz de fazer nada por si mesmo. Faz coisas apenas na medida em que lhe provemos programas que fazem a máquina universal emular máquinas de nosso design (MAHONEY, 2011, p. 86).[xli]

A máquina universal é, ao mesmo tempo, a mais forte e a mais fraca. Por ser universal, é capaz de fazer tudo o que *qualquer* máquina computável pode fazer, mas justamente por causa de tal universalidade não pode fazer nada sozinha. A máquina universal existe apenas enquanto acoplada, enquanto estabelece alianças. É nesse ponto que, seguindo Latour (1988), podemos dizer que os termos "força" e "fraqueza" são intercambiáveis. Pois "nenhum actante é tão fraco que não possa engajar outro. Os dois unem-se, então, e formando uma unidade para um terceiro podem, portanto, deslocá-lo mais facilmente". (p. 159)[xlii] Quando uma máquina universal está executando outra máquina, como podemos diferenciá-las? Ao mesmo tempo, uma máquina universal isolada, e um programa sem uma máquina para executá-lo, não fazem nada sozinhos, mas *podem ao menos aliar-se*, e tal aliança é possível exatamente por conta dessa fraqueza.

Mas a força/fraqueza da máquina universal existe apenas em sua concepção teórica. Na prática, nenhum computador é de fato uma máquina universal. Não apenas por ser finito, mas porque seu modo de ação será definido de acordo com os programas que utilizará, isto é, o coletivo no qual se insere e participa da constituição. Podemos dizer que os computadores são ainda mais fracos que as máquinas universais, ou mais precisamente, que modalizam a máquina universal a associando a um *hardware* específico, assim como a interesses econômicos, políticos e científicos que determinarão o limite de sua universalidade através dos tipos de programas que serão efetivamente executados. Nesse sentido, a modalização integra a máquina universal a determinados grupos ao mesmo tempo em que a isola e afasta da universalidade propriamente dita.

Seria mais correto dizer que nenhum desses computadores implementa uma máquina universal, pois esta não serviu de modelo para sua construção neste período. É apenas *a posteriori* que podemos fazer essa associação. Quando Carr (1958) evoca a máquina universal, não é para afirmar a sua identidade com o computador, mas justamente porque este *precisa se transformar em uma máquina universal.* Deve deixar de ser uma calculadora

glorificada para se tornar uma máquina manipuladora de símbolos.

Tal modificação não exige, a princípio, nenhuma transformação ao nível do *hardware*. O que deve mudar é o coletivo de atores humanos e não-humanos e o conjunto de práticas a eles associados. Tanto programadores quanto programas devem ser deslocados, aproximando-se de uma prática voltada a problemas e distanciando-se da máquina. O híbrido formado pela combinação entre programador, programa e máquina deixa de ser "computador que realiza cálculos matemáticos na maior velocidade possível em seu *hardware*" para "computador que manipula símbolos e problemas".

A máquina universal é reintroduzida na prática da programação não como um modelo abstrato, mas como um modo de conceber a programação, que neste momento, início dos anos 60, ocupa uma posição cada vez mais central no domínio da computação. Na primeira Conferência de Trabalho sobre Programação Automática de Computadores Digitais, que aconteceu em 1959, Booth (1960) afirma no início de sua fala de abertura que Turing foi, também, o fundador da área de programação automática.

> [O teorema de Turing] afirma que qualquer máquina de computar que possua o número mínimo de instruções apropriadas pode simular qualquer outra máquina de computar, independentemente de quão grande seja o repertório de instruções desta. Todas as formas de programação automática são meras incorporações deste teorema muito simples e, ainda que de tempos em tempos possamos ficar em dúvida sobre como, por exemplo, FORTRAN difere de MATH-MATIC ou o AUTOCODE de Ferranti difere de FLOW-MATIC, seria mais fácil de compreender ao ter em mente que são simples consequências do teorema de Turing (p. 1).[xliii]

A máquina universal, nesta fala, é tomada de duas maneiras distintas. A primeira a coloca como princípio das mais variadas linguagens de programação automática, pois permite que instruções de uma máquina sejam traduzidas para outra. O segundo modo implica que uma máquina universal pode simular uma máquina *mais poderosa*, isto é, uma máquina que contenha um repertório de instruções ausente na primeira. Uma posição similar também é adotada por Gorn (1957).

Na mesma conferência, Gill (1960) propõe uma divisão similar, entre tradução e simulação, e liga a máquina universal à segunda. Dentro de suas limitações de *hardware* e memória, um computador "junto com um programa adequadamente projetado, pode ser equivalente a outro computador que talvez não exista como uma entidade separada" (p. 181)[xliv], incluindo um computador muito mais complexo. Enquanto que a tradução de linguagem para código, que foi vista até agora, permitia que um programa fosse executado em qualquer máquina que o traduzisse, a simulação ou virtualização, através do *software*, permitiria que um computador inteiro fosse simulado ou virtualizado dentro de outro, incluindo a implementação de computadores com arquiteturas inexistentes materialmente. A

virtualização, contudo, tem um grande impacto no desempenho da máquina, então mesmo que possa simular um computador mais poderoso, será de um modo muito mais lento do que sua contraparte física, do mesmo modo que uma máquina específica provavelmente será mais rápida que uma máquina executada por uma máquina universal.

A virtualização fortalece a ideia de programas que podem ser programados independentemente do *hardware* em que serão executados, por sua tradução por compiladores e interpretadores ou porque a máquina para a qual foram desenvolvidos será simulada (o que implicará em uma máquina universal executando outra que seja capaz de ler o programa em questão). Se no caso da teoria de autômatos e na lógica a máquina universal teve de ser modificada e transformada para que se tornar o modelo abstrato do computador, independentemente de qualquer implementação específica, na programação as linguagens automáticas, enquanto máquinas universais, modificam o programa para que este seja executado em uma máquina. Tanto na tradução quanto na simulação a máquina universal, por um lado, se alia ao *hardware,* o retoma seja ao nível do computador, seja através da virtualização, enquanto que por outro lado distancia o *software* da máquina. Para que exista uma independência da máquina, é necessário que a máquina seja levada em conta.

3.3 Controvérsias de uma linguagem universal

A linguagem FORTRAN veio a desempenhar efetivamente o papel de uma linguagem universal no campo da pesquisa científica, dada a sua ampla adoção que, contudo, não foi resultado de alguma propriedade intrínseca da linguagem, mas de seus compiladores. O desenvolvimento da linguagem FOTRAN foi focado menos na especificação de uma linguagem geral e independente de qualquer máquina, e mais na construção de um compilador rápido e realmente efetivo, tornando-se a primeira linguagem compilada a trazer um verdadeiro benefício perante as linguagens específicas.

A iniciativa de desenvolver uma linguagem algébrica internacional, ALGOL, pode parecer redundante em um primeiro momento, dado que ocuparia o mesmo nicho de programação científica da linguagem FORTRAN. A linguagem ALGOL seria, todavia, o produto de uma iniciativa internacional, envolvendo pesquisadores norte-americanos e europeus, representando as mais diversas associações e grupos ligados à área da computação, enquanto o desenvolvimento da implementação oficial da linguagem FORTRAN encontrava-se sob controle da IBM (ENSMENGER, 2010; MACKENZIE, 2004).

Ser uma "linguagem algorítmica/algebraica internacional" indicava, além de sua origem transnacional, o desejo de não estar associada a nenhuma empresa específica. Tinha como propósito constituir-se como uma linguagem de publicação, assim como linguagem de programação. Tais objetivos foram formulados da seguinte maneira no relatório ALGOL 58:

> I. A nova linguagem deve se aproximar o máximo possível à notação matemática padrão e ser legível sem a necessidade de mais esclarecimentos.
> II. Deve ser necessário utilizá-la para a descrição de processos computacionais em publicações.
> III. A nova linguagem deve ser mecanicamente traduzível em programas de máquinas (PERLIS; SAMELSON, 1958, p. 9).[xlv]

Aos três objetivos deveriam corresponder três linguagens: linguagem de referência, linguagem de publicação e linguagem de representação de *hardware*. A diferença entre as três linguagens consistiria apenas no conjunto de símbolos utilizados por cada uma, já que nem todas as máquinas possuiriam os mesmos caracteres de entrada e saída, assim como certos caracteres seriam melhores para a publicação. Em todo caso, nem a sintaxe nem o significado dos símbolos seriam modificados.

Os objetivos foram apresentados em um relatório preliminar publicado em 1958, resultado de uma reunião ocorrida no mesmo ano, que definiu uma primeira versão da linguagem, vindo a ser chamada de ALGOL 58. Os objetivos mantiveram-se os mesmos em versões posteriores da linguagem. A linguagem seria definida nos termos de sua linguagem de referência. Esta, como é colocada no primeiro objetivo, deveria se aproximar ao máximo da notação matemática tradicional, permitindo uma familiaridade imediata entre pesquisadores e código. O primeiro objetivo estabeleceria uma referência comum entre os utilizadores da linguagem, facilitando a associação de pesquisadores, equações e algoritmos, cuja aliança seria facilitada pela clareza e generalidade da linguagem.

O segundo objetivo encontrava-se muito próximo ao primeiro e desempenhava quase a mesma função de comunicação entre pares. A linguagem de referência, contudo, deveria ser considerada como um modelo para implementação de compiladores e para as discussões acerca da própria linguagem, enquanto a linguagem de publicação deveria ser utilizada para expressar problemas específicos e sua comunicação em meios nos quais o foco encontrava-se nos problemas e nas questões tratadas e não em implementações específicas. Por sua vez, o terceiro objetivo, que concerne às representações de máquina, designava o domínio próprio de uma linguagem de programação, isto é, que deveria ser traduzível em código de máquina e executada. Este nível não expressaria nem modelos nem problemas, mas instruções.

Em 1960 é realizada uma conferência em Paris na qual a nova versão da linguagem, agora batizada como ALGOL 60, é definida, a partir dos trabalhos realizados em conferência

e discussões preliminares, após 1958. Está será a a versão da linguagem que se tornará o padrão, expandindo e definindo com mais precisão a versão de 58. Todo o trabalho das conferências voltou-se para a definição de uma linguagem da referência, a partir da qual os compiladores da linguagem seriam implementados. Este desenvolvimento seguia um caminho eminentemente teórico, enquanto outras linguagens de programação eram produzidas de um modo mais orgânico, em que especificação e compiladores eram produzidos conjuntamente, sem um planejamento tão extenso e formal, como foi o caso de FORTRAN (DAYLIGHT, 2012).

A definição de uma linguagem da referência *antes* de sua implementação nas mais diversas máquinas permitiria que a linguagem ALGOL fosse pensada como uma linguagem efetivamente universal ou geral, já que não teria como base ou modelo nenhum tipo de *hardware*. O modelo precederia a máquina. Entretanto, o desenvolvimento do aspecto universal da linguagem não se deu sem controvérsias. Logo os participantes das conferências se encontraram em meio a uma dicotomia entre especialização e generalidade:

> Especialização refere-se à restrições de linguagem, soluções estáticas (em tempo de compilação), e à exploração de propriedades específicas das máquinas – no interesse da eficiência. Generalização refere-se a constructos gerais de linguagem, soluções dinâmica (em tempo de execução), e um designa independente da máquina – no interesse de correção e segurança (DAYLIGHT, 2012, p. 46).[xlvi]

A especialização tem como objetivo central a eficiência. Neste momento, final dos anos 50, havia uma grande ceticismo em relação à possibilidade de implementação efetiva de uma linguagem geral. A generalidade teria um custo muito alto em desempenho para ser utilizável, sobretudo porque determinadas funções propostas pela linguagem seriam mais difíceis de implementar em determinadas máquinas. Deste modo, ao colocar restrições na linguagem, como a exclusão ou limitação de funções recursivas, certos tipos de arquitetura ou de *hardware* poderiam ser favorecidos, permitindo que determinadas especificidades de seus funcionamentos fossem consideradas tanto pelo compilador quanto pelo modo de utilizar a linguagem. Por sua vez, a generalidade permitiria a produção de um código menos sujeito a erros, já que não dependeria das especifidades da máquina em que seria executado e não estaria sujeito a erros que seriam exclusivos àquela arquitetura, fora *bugs* de implementação. Na medida em que o código-fonte escrito com foco na generalidade não teria de levar em conta o código de máquina, a descrição do processo algorítmico ganharia prioridade e evidência, permitindo que fosse analisado com mais facilidade expondo eventuais erros lógicos.

O problema então era demonstrar que a construção de um compilador sem restrições

à nova linguagem era viável. Foi o que fizeram Wijngaarden e Dijkstra, dois pesquisadores holandeses, que em 1960 construíram um compilador de ALGOL para o computador X1, o que pegou de surpresa muitos dos defensores da eficiência e constituiu aliado forte em favor da generalidade (DAYLIGHT, 2012). Como Latour (2000) coloca, na ciência quando as controvérsias se acirram, as discussões tornam-se mais técnicas. Caso a controvérsia não seja resolvida através de discussões, artigos, publicações e conferências, apela-se então ao laboratório. No caso da matemática o "laboratório" é o próprio texto, onde a demonstração funciona como um experimento mental. Nas linguagens de programação, especialmente quando se trata de sua relação com o *hardware*, o problema é muito mais empírico e técnico, a experimentação é a própria construção da máquina ou programa a ser testado.

Como vimos, atores e actantes são produzidos como resultados de teses, de provas que colocam sua agência em questão e definem suas competências. Neste caso, a construção de um compilador sem restrições produz uma distinção entre boas e más máquinas de computar.

> Nossa solução é para "boas máquinas de computar", em que significamos "bom" como a completa liberdade de determinar como o computador deve ser usado, isto em contraste com máquinas para as quais considerações de eficiência nos forçam a praticar um modo especial de uso, isto é, nos força a considerar propriedades específicas e peculiaridades da máquina (DIJKSTRA, 1961, p. 2).[xlvii]

As boas máquinas são aquelas que passam pelo teste da generalidade, cuja potência não está aprisionada pelas exigências de eficiência. Nesta separação há, contudo, uma certa ambiguidade. Uma linguagem geral é justamente aquela que pode ser concebida independente de máquinas específicas, *desde que* tais máquinas sejam boas o suficiente. Se passar pelo teste, a máquina pode aceder ao status de generalidade, abdicando de suas peculiaridades. Ao contrário, quando a máquina falha, é levada à sua condição específica e encontra-se apartada da irmandade de todas as outras máquinas que passam a ter uma linguagem comum. São, respectivamente, casos de modalizações positivas e negativas, de fortalecimento e associação por um lado, e enfraquecimento e isolamento por outro.

Do lado da eficiência, não havia menos ambiguidade, pois a restrição da generalidade da linguagem é o que garantia que mais máquinas implementassem compiladores de ALGOL. Ao restringir a linguagem, esta se tornaria mais forte por arregimentar um número maior de aliados, o que os fortaleceria também. A especificidade seria um caminho para uma generalidade restrita ainda que com uma amplitude maior. Se uma distinção poderia ser traçada em nome da generalidade é apenas porque eventualmente, em um futuro muito próximo, a eficiência não seria mais um problema com o desenvolvimento de

máquinas mais potentes. A controvérsia se resolveria sozinha, transformando a generalidade em caixa-preta, apagando a própria distinção entre boas e más máquinas, que não seriam nada mais que um episódio menor e irrelevante da história de linguagens de programação.

3.4 A crise do *software*

No mesmo período em que a linguagem ALGOL estava sendo discutida e definida, a linguagem COBOL também era formulada em encontros e conferências. Enquanto ALGOL incorporava os interesses de um grupo em grande parte acadêmico, COBOL fora planejada e desenvolvida por representantes das maiores empresas de computação, processamento de informação e também pelo governo, sem a presença de nenhum pesquisador ligado a universidades. Em outras palavras, não havia intersecção nem trocas entre os dois grupos, pois cada um estava ligado a interesses distintos. ALGOL respondia às necessidade acadêmicas de uma linguagem geral que facilitasse a comunicação entre cientistas, assim como da própria programação, ocupando um espaço similar ao de FORTRAN. A linguagem COBOL, por sua vez, atendia às necessidades de aplicativos comerciais (ENSMENGER, 2010; MACKENZIE, 2004).

Em COBOL não era necessário desenvolver uma notação para expressões matemáticas muito complexas, dado que raramente eram utilizadas no âmbito comercial e empresarial. Assim como ALGOL e FORTRAN, COBOL fora projetada pensando na independência de máquina, o que permitiria o desenvolvimento de aplicativos comerciais que poderiam ser executados nas mais diversas arquiteturas, necessitando apenas pequenas alterações no código. O foco da linguagem encontrava-se no processamento envolvendo palavras e expressões linguísticas, além de possuir uma sintaxe que facilitaria a sua utilização inclusive por quem não era um programador. Na sua sintaxe era permitido a utilização de "*noise words"*, palavras e expressões que não seriam traduzidas para nenhuma instrução de máquina. Por exemplo, a expressão

READ *file1* RECORD INTO *variable1* AT END *goto procedure2*

É equivalente à expressão

READ *file1* INTO *variable1* END *goto procedure2*

As palavras “RECORD” e “AT” são supérfluas do ponto de vista da compilação e execução do código, sendo descartadas. Contudo, auxiliam a leitura e compreensão por parte de humanos. A primeira expressão pode ser facilmente lida e compreendida até por quem não tenha familiaridade com programação.

Por conta deste status mais direto e pragmático, a linguagem COBOL foi duramente criticada e rechaçada pelo âmbito acadêmico, com Dijkstra chegando a dizer que “COBOL danifica a mente” (DIJKSTRA apud ENSMENGER, 2010, p. 100)[xlviii], enquanto outros afirmaram que era “terrível” e “feia”. Do mesmo modo, tanto gerentes, empresários quanto programadores de aplicativos comerciais consideravam as contribuições acadêmicas como muito teóricas e completamente desconectadas dos interesses comerciais, sem falar de sua atitude elitista e focada mais em critérios abstratos do que em problemas reais.

As críticas acadêmicas não prejudicaram a ampla adoção de COBOL no âmbito comercial, o que inicialmente se deu, em parte, pela pressão do governo norte-americano ao estabelecer que apenas utilizaria programas escritos em COBOL. Hoje ainda existem mais de 240 milhões de linhas de código escritas em COBOL, e no ano 2000 acerca de 90% das transações bancárias eram processadas em aplicações escritas em COBOL, assim como 75% do processamento de dados de negócios. Boa parte de todo este código fora produzido antes de 1980 (ENSMENGER, 2010).

No início da década de 60 existiam cerca de 5400 computadores instalados nos Estados Unidos. Em 1970 esse número crescera para 74 mil. Desde seu começo nos anos 50, a indústria de *software* viu-se permanentemente em falta de programadores, especialmente de bons programadores. Os projetos encontravam-se cronicamente com falta de pessoal qualificado para executá-los, um dos motivos que levou em 1968 à culminação da “crise do *software”*. Empresas como IBM, SDC, CDC foram responsáveis por treinar grande parte da primeira geração de programadores para o desenvolvimento de aplicativos comerciais, o que contudo não era suficiente para suprir a demanda do mercado, o que produzia também uma alta rotatividade entre programadores nas empresas (CAMPBELL-KELLY, 2003).

Um dos grandes apelos da linguagem COBOL encontrava-se em sua sintaxe, no fato de que poderia ser compreendia até por não programadores. Isto abria a possibilidade para que tantos gerentes quanto outros funcionários não-qualificados de uma empresa pudessem escrever ou pelo menos compreender e utilizar o código dos programas. Por um lado, isto daria conta da falta de programadores, por outro, diminuiria a dependência da equipe e, em especial da gerência, do programador, figura que geralmente era vista como estranha ao modo de trabalho e gerenciamento de empresas.

Durante toda a breve história da programação de computadores, a figura do programador foi considerada a partir de dois polos. O primeiro o colocava como um artesão, técnico, ou artista, dotado de um espírito criativo que necessitaria de espaço e isolamento para exercer sua atividade. O segundo tratava o programador como um profissional, como alguém que deveria ser gerenciado e capaz de seguir a disciplina estrita exigida pelas especifidades lógicas e exatas da programação, assim como as necessidades da empresa ou da equipe em que trabalhava. A origem da concepção do programador como artista pode ser estabelecida a partir do modo como a atividade era exercida na década de 50. Backus (1980) relata como era a experiência de programar naquela época:

> O programador tinha de ser um inventor engenhoso para adaptar seu problema às idiossincrasias do computador: deveria fazer o programa caber em uma memória minúscula e superar dificuldades bizarras ao inserir e obter informação do computador, tudo isso enquanto utilizava conjunto de instruções limitado e usualmente peculiar. Tinha de empregar toda artimanha que pudesse pensar para fazer um programa rodar em uma velocidade que justificasse o alto custo de executá-lo. E ele tinha que fazer tudo isso por sua própria engenhosidade, pois a única informação de que dispunha era um problema e um manual da máquina.
> […]
> Programar no começo da década de 1950 era uma arte negra, um assunto arcano privado envolvendo apenas um programador, um problema, um computador e talvez uma pequena biblioteca de subrotinas e um programa de *assembly* primitivo (BACKUS, 1980, p. 125–126).[xlix]

É possível ver todas as dificuldades que levaram ao desenvolvimento de linguagens automáticas e as linguagens de programação independentes de máquinas. Mas podemos ver também que era exigido do programador, como indivíduo, uma artificiosidade que lhe dava a flexibilidade necessária para superar as mais variadas adversidades. Sem a flexibilidade proporcionada por um modo de pensar engenhoso e criativo o programador não seria capaz de criar soluções específicas a cada problema, máquina e situação. O produto final do processo era como um feitiço, um código tão específico que não poderia mais ser compreendido exclusivamente a partir de princípios ou regras gerais, conhecimento restrito apenas aos iniciados de um saber arcano.

A noção de que o programador deveria ser alguém criativo, além de lógico, tornou-se um dos pilares dos processo de recrutamento de pessoal na indústria do *software*, ao qual somou-se a característica de que era alguém que se dava melhor com máquinas do que com pessoas. Também era assumido que ser um "bom programador" constituía uma aptidão inata do sujeito, algo que não poderia ser ensinado. Foram desenvolvidos diversos testes psicológicos com o intuito de selecionar programadores com essas características (ENSMENGER, 2010).

Todos esses fatores contribuíram para que o programador encontrar-se em uma

situação única dentro da instituição. Primeiro, por ser o portador de um conhecimento arcano, era o único capaz de se comunicar com o misterioso *software*, tal como um sacerdote. Isto ao mesmo tempo alienava seus colegas não programadores, incluindo supervisores e gerentes, enquanto dava ao programador um poder considerável, sendo o único capaz de controlar, modificar e entender o *software*. Segundo, em um ambiente onde o trabalho em equipe, adequação a valores e práticas definidas pela gerência são valorizados, o programador é aquele que aparece como não gerenciável, que trabalha em um período diferente dos outros funcionários (durante o dia o programa é utilizado, então a noite é o melhor período para ser programado), veste-se de outro modo e tem toda a liberdade para exercer sua criatividade.

Dada a escassez de programadores no mercado, essa atitude era suportada, quando não encorajada. Contudo, a tensão entre gerenciamento e programadores crescia progressivamente. A isto veio juntar-se a crescente percepção durante a década de 60 de que a indústria do *software* encontrava-se em meio a uma crise. Especificamente, os problemas que levaram à crise do software, incluíam o atraso de projetos, custos acima do esperado, pouca eficiência, muitos erros e dificuldade de gerenciamento.

O que estava em jogo era a questão de como programar um computador. É em meio a essa crise em que a figura do programador assume o primeiro plano, na medida em que seu fazer, junto aos novos computadores, torna-se uma fonte de controvérsias. Em 1968, na Alemanha, foi organizada a primeira "Conferência de Engenharia de Software da OTAN"[18], cujo objetivo era discutir os modos pelos quais tais problemas poderiam ser evitados, através da criação de métodos eficientes de programação e gerenciamento de projetos, ou adotando um modelo formal de verificação de *software* (MACKENZIE, 2004).

O maior problema identificado dentre as causas que levaram à crise do *software* foi a ausência de uma metodologia apropriada de programação e desenvolvimento, tal como se dava com a produção industrial. A programação estava muito mais próxima de um ofício ou técnica artesanal do que propriamente uma engenharia, mais para arte negra do que ciência. A criatividade, como capacidade de analisar e solucionar problemas ao invés de simplesmente limitar-se à codificação mecânica, passou a ser uma caraterística indesejada. Pois era a demasiada liberdade dada aos programadores, acompanhada por um modo de programar sem sistematização, que levava ao excesso de *bugs*, a técnicas ineficazes de detecção de erros e consequentemente a atrasos e ao estouro do orçamento.

Havia entre os participantes um consenso geral acerca da necessidade de ser criada uma "engenharia de *software"*, com o termo "engenharia" marcando o desenvolvimento de

18 NATO Sofware Engineering Conference.

técnicas padronizadas de produção de *software*, tal como era realizado em outros campos da indústria. Contudo, os participantes discordavam em relação à sua constituição e a natureza das técnicas a serem implementadas (MACKENZIE, 2004). *Grosso modo*, a divisão se dava entre duas grandes perspectivas. A primeira defendia novas práticas de gerenciamento, que iam desde modelos extremamente hierarquizados até modelos de uma programação "sem ego". A segunda posição afirmava um modelo de verificação formal do *software*, onde erros seriam evitados pela utilização de provas matemáticas. As duas perspectivas apoiavam-se também na utilização e desenvolvimento de novas ferramentas e tecnologias, como foi o caso da programação estruturada, que se adequava tanto a um modelo hierárquico de gerenciamento quanto a uma concepção formal da prática de programação (ENSMENGER, 2010; MACKENZIE, 2004).

É comum que organizações possuam uma estrutura hierárquica, que define cargos, funções e vias de comunicação em formato de pirâmide, especialmente neste período que abordamos, entre anos 60 e 80. Como vimos, o programador não se adequava muito bem a esta estrutura, seja pela natureza peculiar da própria programação, seja por seus traços de personalidade anti-social ou até mesmo "esquizofrênica", de acordo com críticos da época (ENSMENGER, 2010). A questão era como incorporar os programadores nesta estrutura.

Para que o programador fosse reincorporado na organização era necessário que não concentrasse mais tantas funções em sua mão, transformando-se em um funcionário especializado com trabalho rotinizado e mecânico. O programador não deveria possuir habilidades distintas das necessárias para montar um carro, sendo apenas mais um trabalhador em uma linha de montagem de *software*. A viabilidade deste modelo dependia de alguns desenvolvimentos técnicos. Primeiro, a utilização de linguagens automáticas como COBOL permitiria que programadores menos qualificados, ou até não-programadores, participassem do processo de produção. Segundo, o *software* deveria ser concebido como um conjunto de componentes modulares, independentes uns dos outros, o que permitiria seu desenvolvimento e compreensão por programadores especializados. "Os princípios subjacentes a esta abordagem eram essencialmente aqueles que se mostraram tão bem-sucedidos na fabricação tradicional: partes substituíveis, tarefas simples e repetitivas, e uma divisão de trabalho estrita" (ENSMENGER, 2010, p. 201).[1] A utilização de linguagens de programação estruturadas, paradigma que foi em grande parte inspirado por ALGOL, permitiria ao *software* ser desenvolvido e programado utilizando abstrações, dividindo-o em problemas a tarefas menores, o que facilitaria a concepção hierárquica tanto do trabalho quanto do programa em si.

O grande apelo do modelo hierárquico provinha de sua facilidade em se enquadrar nos moldes organizacionais tradicionais, atendendo à demanda dos gerentes e supervisores, sem conhecimento técnico, em conseguir controlar programadores que agora passavam a ser funcionários desqualificados, rotinizados e leais à empresa. Todavia, na prática este modelo não obteve tanto sucesso. Programar era uma atividade muito mais difícil e complexa do que era proposto e não podia ser facilmente reduzida ao modelo de linha de montagem. Os próprios programadores também resistiam à nova identidade profissional que lhes era imposta.

Outro modelo proposto foi o da "equipe cirúrgica", no qual o trabalho seria dividido em pequenas equipes, com cada equipe liderada por um programador experiente e hábil, o superprogramador, que assim como um cirurgião, seria responsável por "definir as especificações e projetar o programa, programá-lo, testá-lo, e escrever a documentação" (ENSMENGER, 2010, p. 207).[li] Seria auxiliado por um segundo programador um pouco menos habilidoso, que poderia substituí-lo se necessário. Os membros restantes da equipe desempenhariam funções específicas, que iam desde atividades administrativas não ligadas diretamente à programação, incluindo duas secretárias, chegando aos programadores que seriam chamados de acordo com necessidades específicas relativas à sua especialidade, como testes, especificidades da linguagem e outras ferramentas. Ainda que seguisse um modelo hierárquico, com diferença de níveis na divisão do trabalho, os programadores não faziam mais parte de uma grande pirâmide monolítica estratificada. Cada grupo preservava uma independência relativa, sendo que o superprogramador ocupava posição similar a do programador artista, que podia dar vazão à sua criatividade, tendo que, contudo, conciliá-la com seu papel de liderança. Em certa medida, este modelo buscava conciliar as necessidades organizacionais com a independência do programador. A questão que se colocava é se seria um método eficaz para o desenvolvimento de grandes projetos de *software.*

O terceiro modelo foi proposto por Gerald Weinberg no clássico livro "*The psychology of computer programming*" (WEINBERG, 1971). Para Weinberg o problema encontrava-se na ideia de que programadores deveriam agir de modo isolado, transformando os programas produzidos em produto exclusivo de sua capacidade pessoal, onde tanto os sucessos quanto, especialmente, os erros dependeriam exclusivamente do indivíduo. Concepção que, como vimos, estava arraigada nos processos de seleção de programadores, onde a habilidade seria uma propriedade inata, assim como a criatividade.

O foco na produção individual centrada no ego produziria uma atitude defensiva por parte do programador, que evitaria ao máximo reconhecer ou aceitar os erros de seu

programa, levando-o a um tipo de dissonância cognitiva, em que os erros simplesmente não seriam considerados ou em que seriam atribuídos a outrem – pessoas ou máquinas. Em todo caso, a prioridade estaria em evitar qualquer tipo de punição ou repreensão individual, ao mesmo tempo em que o reconhecimento pessoal era perseguido.

Com o modelo de programação sem ego [*egoless*], o trabalho em grupo é valorizado. O código produzido por cada indivíduo é revisto e até retrabalhado por seus colegas e erros que escapassem ao autor original poderiam ser detectados mais facilmente. Além do mais, o código já seria escrito levando em conta a sua leitura por outrem, o que melhoraria a sua comunicação e manutenção. Tanto o design quanto a implementação dos projetos seria realizada e decidida pelo grupo como um todo. Os membros participariam e se envolveriam em todas as etapas do projeto, apesar de que aqueles que fossem melhores em determinadas áreas pudessem assumir a liderança. Por exemplo, na fase de testes, o participante com a maior habilidade nessa área se tornaria o líder temporário. O *software* não seria propriedade de um indivíduo, mas do grupo como um todo. Este modelo, também chamado de "programação adaptativa" ou democrático, permitiria um desenvolvimento mais flexível, pois se apoiaria nas variações individuais para dar conta das novas situações que surgissem.

Apesar do sucesso e popularidade do livro de Weinberg, a adoção deste modelo não era simples, pois implicava uma série de mudanças ao modo como o gerenciamento e a prática da programação eram concebidos, sobretudo porque tais grupos se organizavam de forma não estruturada, dificultando sua integração aos modelos tradicionais de gerenciamento, impedindo sua adoção pelos líderes de organizações. Todavia, há indícios de que formar grupos informais e não estruturados de programadores era uma prática comum na indústria. A linguagem da programação sem ego foi, entretanto, adaptada e transformada aos interesses de gerentes tradicionais. A perspectiva de um código não-proprietário, escrito de tal forma que qualquer programador pudesse compreendê-lo, permitia criar um *software* cujo grupo de desenvolvedores pudesse ter seus membros facilmente substituídos, o que ia de encontro à ideia original da programação sem ego de constituir grupos orgânicos e dinâmicos (ENSMENGER, 2010).

Os três modelos de gerenciamento propostos, o hierárquico, o cirúrgico e o sem ego, apesar de todas as suas diferenças possuíam muitos traços comuns no que se refere à natureza do programador, de sua atividade e da habilidade envolvida. Assumiam "que a programação era essencialmente um empreendimento criativo; que programadores individuais variavam enormemente em termos de estilo e produtividade; e que as práticas de programação correntes lembravam muito mais um ofício do que uma ciência" (ENSMENGER, 2010, p. 215).[lii]

Acreditavam também que a produção de *software* poderia ser controlada e gerenciada dentro de certos limites. Este conjunto de soluções correspondia a um modo mais pragmático e social de propor uma solução à crise do *software* e de possíveis rumos a uma engenharia do *software*. O outro conjunto de soluções, que será abordado na próxima seção, indicava a necessidade de uma abordagem mais científica à engenharia do *software*, apoiada no desenvolvimento de uma ciência da computação que seguiria os moldes da matemática, colocando a prática de verificação formal de programas como a saída para o problema dos erros e da correção do código. É nesta perspectiva e reencontramos o conflito entre universalidade e eficiência e suas implicações para o desenvolvimento de *software*.

3.5 Verificação formal e a bala de prata

Os primeiros computadores eram máquinas instáveis e pouco confiáveis, com uma alta taxa de falhas e problemas. Conseguir fazer com que um computador funcionasse por longos períodos de tempo sem apresentar problemas já era uma grande vitória. Não é à toa que, neste momento, o *software* não se apresentasse como um problema, pois toda a atenção voltava-se para as grandes máquinas caprichosas. Acreditava-se que a programação não consistiria em nada mais do que um processo de tradução do algoritmo realizado por *coders*, como as ENIAC Girls. Quando computadores mais robustos e confiáveis foram desenvolvidos, a dificuldade e complexidade de programação tornou-se cada vez mais evidente. Foi o que constatou Wilkes, um dos autores do primeiro manual de programação e ligado à construção do computador EDSAC:

> Simplesmente não me havia ocorrido que haveria qualquer dificuldade em fazer os programas funcionarem. E foi com um pouco de choque quando me dei conta de que pelo resto da minha vida passaria uma boa parte de meu tempo encontrando erros que cometi em programas (WILKES apud TEDRE, 2015, p. 105 [1967]).[liii]

Passar o resto da vida à caça de erros e *bugs* é o que definiu, e ainda define, uma parte considerável do trabalho do programador. Ou, como colocou Dijkstra (1972a), "enquanto não havia máquinas, programar não era um problema; quando possuíamos poucos computadores fracos, programar tornou-se um problema médio, e agora que temos computadores gigantescos, programar tornou-se um problema igualmente gigantesco" (p. 861).[liv] À medida em que os computadores foram se tornando mais eficientes e poderosos, os problemas de programação tornaram-se ainda mais graves, o que acabou culminando na crise do *software* de 68 e que permanece na retórica atual de uma crise crônica do *software*.

O desenvolvimento de linguagens automáticas foi uma das primeiras tentativas de amenizar ou evitar os erros imanentes à programação, já que o trabalho repetitivo de tradução para código de máquina, propenso a erros, seria deixado a cargo do computador. Mas como a retórica da crise do *software* atesta, os erros não só diminuíram como aumentaram, especialmente pelo crescimento da demanda de novos programas e pela entrada de programadores não qualificados no mercado.

A criação da linguagem ALGOL se inseriu na saída proposta pelas linguagens automáticas, com o diferencial de que pretendia ser mais do que uma linguagem de programação, mas também um meio de comunicação científica. A universalidade não se referia apenas à sua independência da máquina, mas à promessa de se tornar uma *língua franca* de expressão de programas. No texto de Knuth e Trabb Pardo, é justamente esse o papel que o algoritmo escrito de uma dialeto de ALGOL 60 assume: um modelo universal de comparação para as mais diversas linguagens.

O apelo à universalidade, como independência de máquina, tornou-se o centro da controvérsia acerca da concepção da linguagem ALGOL, pois marcava o choque entre duas tendências ou até mesmo entre duas éticas. A primeira defendia a restrição da linguagem como caminho para incorporação das especifidades das máquinas em favor da eficiência, isto é, era a perspectiva de uma arte negra da programação, que se guiava de um modo ainda artesanal ou artístico, focado na prática e em todas as exceções e especifidades que esta carregava. O programador é aquele que conhecia a máquina de perto, familiar a todos os seus desejos e caprichos, cuja prática lhe permitiria aproveitar o máximo de sua potência, ainda que por caminhos obscuros e tortuosos.

A segunda tendência, do lado da universalidade, propunha-se a criar uma linguagem elegante e que facilitasse a escrita de programas corretos. Pior do que um programa ineficiente era um programa com erros. Ao utilizar uma linguagem universal, que descartasse qualquer particularidade da máquina em que fosse executada, o desenvolvimento do programa seria guiado apenas pelos problemas em questão e pelos princípios gerais da linguagem. Este modo de operação reflete a influência da pensamento matemático, que opera com abstrações e deduções de princípios gerais.

Para Brooks (1975), no clássico livro "O mítico homem-mês", o programador,

> assim como o poeta, trabalha apenas levemente afastado do material de pensamento puro. Ele constrói castelos no ar, feitos de ar, criando pelo uso da imaginação. […] A magia do mito e da lenda se tornou verdadeira em nosso tempo. Ao digitar o encantamento correto no teclado o monitor vem à vida, mostrando coisas que nunca foram nem poderiam ser (BROOKS, 1975, p. 7–8).[lv]

O caráter mágico e arcano da programação, contudo, não é composto por uma pura liberdade, pois assim como nos mitos, se o encantamento não é realizado da forma correta simplesmente não funciona, e isto quando não se volta contra seu criador. O encantamento deve ser perfeito e "seres humanos não são acostumados a serem perfeitos. Adaptar-se à exigência da perfeição é, penso, a parte mais difícil de aprender a programar" (BROOKS, 1975, p. 8).[lvi] Nas palavras de Brooks encontramos condensadas as duas grandes perspectivas acerca da natureza da programação. De um lado, a criatividade quase mágica, do outro a perfeição e a exatidão.

Para os defensores da universalidade, a perfeição só poderia ser atingida caso fossem abandonadas as práticas arcaicas e não científicas da programação. A prática deveria ser pautada por uma teoria da computação baseada em princípios científicos. A solução dos problemas práticos da programação exigia a criação de uma ciência da computação capaz de fornecer os princípios teóricos que definiriam a melhor forma de programar.

A criação da linguagem ALGOL constituiu um momento importante para reunir diversos pesquisadores que possuíam um interesse mais matemático pela computação. Se a busca da perfeição era o caminho mais eficaz para a solução dos inúmeros problemas inerentes à prática da programação, nenhum outro domínio do saber se apresentava como tão propício a servir de modelo como a matemática. A prova e a dedução matemática, diferente da indução na ciência, apresentaria um caráter certo e absoluto. Não operaria por tentativa e erro, ou pela afirmação de verdades parciais, mas constituiria um processo de derivação de uma certeza a outra (MACKENZIE, 2004).

Como afirma McCarthy, em 1962, ao propor uma ciência da computação matemática, "em uma ciência matemática é possível deduzir a partir de pressupostos básicos as propriedades importantes de entidades tratadas pela ciência" (MCCARTHY, 1993 [1962], p. 35).[lvii] A ciência da computação teria como objeto os computadores enquanto autômatos finitos e deveria estudar "as muitas formas em que espaços de dados são representados na memória do computador e como procedimentos são representados por programas de computadores" (p. 36).[lviii] O objeto não seria propriamente o computador, mas seu modelo abstrato, o que permitira que uso da dedução.

A princípio, programas de computador poderiam ser concebidos como sistemas formais que seguiriam os mesmos princípios da lógica e da matemática. Considerar as consequências de um programa não seria muito diferente de uma dedução ou prova lógica. Nesta perspectiva a

> programação de computadores é uma ciência exata, em que todas as propriedades e todas as consequências de sua execução em qualquer ambiente dado poderia se encontrado, em princípio, a partir do próprio texto do programa, através de puro

> raciocínio dedutivo. O raciocínio dedutivo envolve a aplicação de regras de inferência válidas a conjuntos de axiomas válidos. Portanto, é desejável e interessante elucidar os axiomas e regras de inferência que subjazem nosso raciocínio sobre programas de computador (HOARE, 1969, p. 576).[lix]

A posição de Hoare é uma das mais extremas em afirmar a identidade quase completa entre programação e matemática, colocando a primeira sob tutela da segunda.

Durante a década de 60, junto às propostas de considerar a ciência da computação ainda em formação tendo a matemática como base e modelo, foi sendo desenvolvida também a ideia de verificação formal de programas. No modo como o desenvolvimento de *software* era (e ainda é) realizado, os testes e a depuração [*debugging*] constituíam os dois principais métodos para detecção e correção de erros. Com os testes, o programa era executado nas mais diversas situações, previstas ou não, permitindo que erros e comportamentos que desviassem de sua especificação fossem identificados. A depuração consistia em rodar o programa acompanhando sua execução e instrução passo a passo de modo a localizar, precisamente, a causa do problema identificado por testes ou pela utilização normal do programa.

Os testes poderiam ser realizados tanto por pessoas quanto por programas especializados, que forneceriam determinados *inputs* ao programa, observando suas saídas, e as julgando a partir de resultados pré-determinados. Em ambos os casos, nenhum teste jamais poderia dar conta de todos os casos e combinações de situações possíveis em que um programa viesse a se encontrar, pelo menos não sem levar alguns milhares de anos. Mesmo com a utilização dos mais variados critérios para tentar situações plausíveis de testagem, alguma coisa sempre escaparia e, às vezes, com resultados catastróficos, como foi o caso da sonda Mariner, destinada a Vênus, que, em 1962, explodiu durante seu lançamento por conta de um único símbolo errado em todas as milhares linhas de código do seu programa de navegação (MACKENZIE, 2004). Em determinados casos, nada menos que um código perfeito seria aceitável, perfeição que os testes, de natureza essencialmente estatística, não poderiam garantir. Como Dijkstra (1972b) viria a afirmar, "testes de programas podem ser utilizados para mostrar a existência de *bugs*, mas nunca para mostrar sua ausência!" (p. 6).[lx]

Um programa é implementado a partir de um conjunto de especificações que descrevem detalhadamente todas as funções a tarefas que deverão ser executadas. Sem erros, um programa é completamente fiel à sua especificação. A verificação formal consistiria em um método para, a partir do texto do programa, o código-fonte, derivar provas que comprovariam sua adequação ou não às especificações do sistema. Pode-se dizer que o programa seria como um conjunto de axiomas, enquanto as especificações seriam os teoremas

a serem provados. “Quando a correção de um programa, seu compilador e o *hardware* do computador forem todos estabelecidos com certeza matemática, será possível dar muita legitimidade aos resultados do programa, e predizer suas propriedades com certeza limitada apenas pela confiabilidade dos componentes eletrônicos” (HOARE, 1969, p. 579).[lxi] A verificação formal não deveria se limitar apenas ao código, mas ao compilador e ao próprio *hardware*, isto é, à todos os níveis em que o programa seria executado. No caso do *hardware*, a prova se referiria à especificação dos componentes eletrônicos e se estes corresponderiam à necessidade do programa. Tal especificação dependeria, todavia, do bom funcionamento do *hardware* na prática, já que estando sujeito às condições materiais e físicas no mundo não seria completamente previsível.

A perspectiva da verificação formal teve uma grande proeminência na academia durante os anos 60 e 70, mas perdeu muito de sua força na década de 80 a partir de um número de críticas cada vez maior que colocavam em questão a viabilidade da proposta. Primeiro, o grande problema da verificação formal era a sua dificuldade e complexidade. Um programa simples, com algumas centenas de linhas, poderia gerar uma prova com vinte mil linhas. Produzir a prova de correção de um programa era um processo demorado e dispendioso, e, logo se viu, que não seria viável nem praticável para qualquer programa. Em mais de uma década de trabalhos no campo, poucas provas de programas reais foram produzidas.

Dentre as críticas ao modelo de verificação formal, a apresentada por De Millo, Lipton e Perlis (1979) destacou-se por tomar como objeto a prova matemática como um *processo social.* Em outras palavras, para os autores o maior problema das provas de correção de programa é que elas não funcionavam do mesmo modo que as provas matemáticas. Uma prova não é automaticamente verdadeira. Para que seja crível deve ser, antes de mais nada, lida, tendo ainda de atrair e convencer seus leitores.

> O estágio da fala é o primeiro filtro da prova. Se não gera nenhum entusiamo ou crença em seus amigos, o sábio matemático a reconsidera. Mas se acham a prova toleravelmente interessante e crível, ele a escreve. Após ter circulado o esboço por um tempo, se ainda parecer plausível, faz uma versão mais clara e a submete para publicação. Se os avaliadores também a considerarem atraente e convincente, será publicada para que possa ser lida por um público mais amplo. Se um número suficiente de membros do público em questão acreditam e gostam da prova, após um período de devida reflexão as publicações de *reviews* dão uma olhada mais vagarosa, para ver se a prova é realmente agradável como pareceu inicialmente e se, após uma calma consideração, realmente acreditam nela.
>
> E o que acontece a uma prova quando é acreditada? O processo mais imediato provavelmente será a internalização do resultado. Isto é, o matemático que lê uma prova e nela acredita tentará parafraseá-la, colocá-la em seus próprios termos, inseri-la em sua visão pessoal do conhecimento matemático. Não há dois matemáticos que internalizarão um conceito matemático exatamente da mesma maneira, então este

> processo leva usualmente a múltiplas versões do mesmo teorema, cada uma reforçando a crença, cada uma contribuindo ao sentimento da comunidade matemática de que a afirmação original provavelmente é verdadeira. […] A transformação mais convincente que pode acontecer é a generalização. Se, pelo mesmo processo social que opera no teorema original, o teorema geral passa a ser acreditado, então a afirmação inicial ganha muito em plausibilidade (DE MILLO; LIPTON; PERLIS, 1979, p. 273–274).[lxii]

Podemos encontrar aí praticamente todos os elementos que Latour (2000) utiliza para explicar o destino de um texto científico. Uma prova só se torna de fato uma prova na medida em que é modalizada, traduzida, parafraseada e generalizada. Deve associar-se, conseguir aliados, resistir e vencer um número crescente de testes e desafios. Mas deve ser considerada, acima de tudo, atraente, relevante e interessante. Por mais que a matemática e a lógica afirmem o caráter absoluto de suas provas, este ainda depende do modo como é lido e aceito pela comunidade. Em um exemplo dado pelos autores, dois grupos, um japonês e outro norte-americano, produziram provas sobre um mesmo objeto matemático. As provas, contudo, eram contraditórias. Após uma cuidadosa análise foi constatado que nenhuma delas apresentava erros. Um terceiro grupo foi chamado para avaliar o problema, o que o levou a produzir uma terceira prova que confirmava os resultados do grupo norte-americano, fortalecendo sua posição. A verdade neste caso não depende de mera dedução correta, apesar de ser necessária, mas cai, em última instância, no âmbito de um jogo de forças e alianças.

Uma prova matemática, antes de ser uma afirmação ou asserção verdadeira, é uma mensagem. Só tem sentido na medida em que é produzida para uma comunidade. No caso da verificação formal, suas provas não serão lidas por ninguém. Enquanto um matemático não perde tempo em buscar a opinião de seus colegas acerca de uma prova que acabou de produzir, uma prova de correção de *software* não é produzida tendo esse compartilhamento em mente. Além do mais, dado que uma prova nunca é absoluta, já que sua verdade depende em última instância de sua aceitação pela comunidade, ela adquire um caráter muito mais probabilístico, o que a aproxima dos testes.

Um grande problema da verificação formal se dá pelo fato de que o programa não é uma entidade estática. Primeiro, a prova de correção necessita que as especificações não sejam modificadas. No processo de desenvolvimento de um *software*, as especificações são constantemente alteradas de acordo com as necessidades que surgem durante o processo, criando uma relação de transformação recíproca entre programa e especificação. A verificação formal só é possível quando há uma separação entre ambos. Qualquer modificação que o programa venha a sofrer exige o desenvolvimento de uma nova prova. Ao mesmo tempo, a prova não pode julgar se uma especificação é correta ou não, pois o fato de uma especificação

existir não garante que esteja bem definida nem que expresse aquilo que o cliente realmente quer, supondo também que isto lhe seja claro, o que nem sempre é o caso.

A crítica mais forte a este modelo, desenvolvida nos trabalhos de Fetzer (1988) e Smith (1985), parte da constatação de que existem lacunas entre especificações, programas e o mundo no qual são executados. Mais precisamente, um programa existe e *age no mundo físico*, estando sujeito a todas as incertezas materiais. Hoare reconhece a limitação de certeza imposta pela natureza do mundo físico, mas isto seria muito mais uma dificuldade do que uma verdadeira limitação. Uma especificação, assim como uma prova, é um modelo de como o programa deve ser e de como se comportaria em condições ideias. A partir do momento em que é executado, estas condições ideais desaparecem, e toda a certeza que era garantida pela prova torna-se, no mínimo, probabilística. Não há uma correspondência necessária entre especificação, prova e um programa agindo no mundo.

> Quando você demonstra que um programa atende a suas especificações, tudo o que você fez foi mostrar que duas descrições formais, com características ligeiramente diferentes, são equivalentes. É por isso que penso ser entre enganador e imoral para cientistas da computação chamarem isso de "correção". O que é chamado de prova de correção é realmente uma prova da compatibilidade ou consistência entre dois objetos formais de um tipo extremamente similar: programa e especificação (SMITH, 1985, p. 23).[lxiii]

A prova de correção não seria nada mais que um jogo puramente formal que não poderia dar conta de toda a variedade, acidentes e contingências do mundo real.

Como Tanenbaum (1976) afirma, "não há programa incorreto" (p. 64).[lxiv] Ao nível de seu funcionamento um *software* não está nem certo nem errado. Esta distinção será feita pelo cliente ou pelo usuário, de acordo com os desejos e expectativas de cada um, que nem sempre coincidem ou apresentam consistência. Mesmo que o programa possua uma prova, nada garante que funcionará do modo como foi especificado, especialmente se o problema ao qual atende não estiver claro ou bem definido. É aí que os testes se fazem necessários, pois ainda que não sejam perfeitos nem esgotem todas as possibilidades ao menos colocam o programa em situações que se aproximam muito às suas condições de uso efetivas.

Em todas as controvérsias e debates acerca das possíveis soluções para a crise do *software,* nenhuma acabou se tornando a solução definitiva que resolveria todos os problemas do desenvolvimento de *software*. Brooks (1986) sintetiza esta situação com a expressão "não há bala de prata". O desenvolvimento de *software* seria como um lobisomem, apresentando-se inicialmente como uma presença inofensiva e familiar para, inesperadamente, transformar-se em uma criatura terrível a aterrorizante que poderia ser derrotada apenas com uma bala de prata. Para Brooks, no caso do *software*, a bala de prata não existe, nenhuma das muitas

soluções propostas pode, sozinha, derrotar a terrível e incontornável complexidade do processo de desenvolvimento. O autor distingue entre dificuldades essenciais e acidentais, isto é, entre problemas inerentes a todo programa e dificuldades circunstanciais. A maioria das soluções propostas, como linguagens de alto nível, sistemas operacionais e ambientes de desenvolvimento, dão conta apenas de elementos acidentais. As dificuldades essenciais são a complexidade, conformidade, mutabilidade e invisibilidade. Todo programa é essencialmente complexo, tendo de se conformar às mais variadas e contraditórias exigências, ao mesmo tempo em que é constantemente modificado, além de ser quase impossível de visualizá-lo em toda a sua complexidade de modo que se torne mais compreensível.

Toda a retórica da crise do *software* não para de reconhecer e indicar as mais diversas dificuldades que afetam, prejudicam o desenvolvimento de programas, ao mesmo tempo em que tenta apontar uma solução última, e quase messiânica, a todos os problemas. O reconhecimento de que não há uma bala de prata, por um lado, implica que uma combinação de práticas e soluções deve ser considerada em cada caso como medida para tentar diminuir erros e atrasos que possam afetar o desenvolvimento do sistema. Por outro lado, é retomada no discurso de uma crise crônica do *software*, cujos termos e problemas ainda hoje são quase os mesmos de 50 anos atrás.

Em meio a todas as idas e vindas de problemas e soluções para a crise do *software*, para uma engenharia do *software* ou ciência da computação, o programador se constituiu como uma figura limítrofe. Mesmo no auge das disputas acerca da concepção da programação como arte ou ciência, os programadores encontravam-se muito mais próximos uns dos outros do que gerentes e membros da indústria ou do exército. “Apesar das muitas diferenças em objetivos profissionais e orientações teóricas que existiam entre programadores vocacionais e cientistas da computação acadêmicos, a força de seus valores estéticos compartilhados e as tradições de oficio forneceram uma base para solidariedade de comunidade” (ENSMENGER, 2010, p. 230).[lxv] É este senso de comunidade que será retomado pelo movimento do *software* livre, ao colocar a questão da universalidade em termos liberdade, concebendo uma outra forma de abordar a programação como arte e sua relação com a generalidade, retomando, no processo, muitos dos problemas e temas da crise do *software*.

3.6 Liberdade

Em 1952, o primeiro computador comercial, IBM 701 é lançado. Enquanto todos os

computadores anteriores eram máquinas únicas, a partir deste momento passariam a existir famílias e classes de computadores, arquiteturas propriamente ditas. Um programa não estaria mais limitado a uma máquina específica, mas poderia rodar, a princípio, em qualquer computador do mesmo tipo. Entretanto, nenhuma das máquinas saia da fábrica instalada com *software*. Cabia aos próprios usuários programá-la do zero, criando suas rotinas, bibliotecas, compiladores e sistemas operacionais sozinhos (SALUS, 2008). Naturalmente, isto levava à reduplicação e fragmentação de trabalho. Dizia-se, em tom de brincadeira, que "existiam 17 clientes com 701s e 18 programas diferentes de *assembly*" (MCCLELLAND; PENDERY apud CAMPBELL-KELLY, 2003, p. 32).[lxvi]

Três anos depois, em 1955, o novo modelo IBM 704 foi anunciado. Seria o sucessor do modelo 701, ainda que as duas máquinas fossem incompatíveis. Era preciso, então, converter os programas escritos no 701 para o 704, que fora adquirido por muitos dos clientes do primeiro modelo. Com o intuito de dividir o trabalho de conversão e permitir o compartilhamento e colaboração no desenvolvimento de programas, o grupo de usuários SHARE foi criado (AKERA, 2001). Além de financiar os encontros, a IBM disponibilizou ao grupo uma biblioteca de mais de 300 programas. A criação de *standards* para o desenvolvimento de *software* também se tornou um dos objetivos do grupo.

Os membros que faziam parte do SHARE eram todos empresas altamente competitivas que apesar de seus interesses privados conseguiram criar um ambiente que colocava os objetivos comuns e os interesses técnicos em primeiro plano. A tradição de colaboração, do qual o SHARE fazia parte e foi sua culminação, teve origem na indústria de aviação da Califórnia antes da segunda guerra mundial e da existência de computadores digitais eletrônicos. A escassez de recursos no entreguerras levou as diversas empresas de aviação a deixarem a competição para áreas periféricas, permitindo que colaborassem em diversas iniciativas de engenharia. Quando estas empresas adquiriram os primeiros computadores da IBM, já se encontravam em meio a uma cultura que valorizava a cooperação entre competidores (AKERA, 2001). Logo, grupos de usuários de computadores de outras empresas também foram criados, como o grupo USE, do computador ERA 1103A da Remington Rand, o grupo DECUS de usuários de computadores da DEC, assim como grupos de usuários das empresas Burroughs, Bendix, Prime e Apollo (SALUS, 2008).

Em paralelo às experiências de cooperação em grupos de usuários no âmbito comercial, especialmente nas universidades, desenvolvia-se uma outra cultura de compartilhamento, que viria a ser definida pela figura do *hacker*. O termo *"hacker*" já fazia parte da cultura do MIT (Massachusetts Institute of Technology), mas foi utilizado em seu

sentido atual pelo Comitê de Sinais e Energia do *Tech Model Railroad Club* (TMRC), um clube de interessados em modelos de trens. Seus membros possuíam um grande interesse por eletrônica, o que os levou a tomarem os computadores do MIT como seu passatempo favorito, principalmente o TX-0 e depois o PDP-1. Os hackers do TMRC viriam a se tornar o núcleo do Laboratório de Inteligência Artificial do MIT (RAYMOND, 2001 [1992]).

A cultura, ou a ética hacker, como mais tarde viria a ser codificada e compilada por Levy (2010 [1984]), assumia um atitude de que a melhor forma de aprender tanto sobre computadores quanto sobre o próprio mundo é desmontando e desconstruindo objetos de interesse para entender como todas as suas partes se conectam e funcionam. Para isso, o acesso à informação, e aos meios de modificar máquinas e instrumentos deveria ser irrestrito. Isto naturalmente levava a uma desconfiança da autoridade, pois o único critério que poderia ser utilizado para um *hack* seria o próprio *hack* e não critérios externos como convenções de classe, títulos, idade ou raça. Para o *hacker,* o mais importante era poder criar seus *hacks* e compartilhá-los com seus pares, tanto como um meio de obter apreciação e reconhecimento, quanto de criar algo que fosse belo e incrível, como uma obra de arte.

Foi neste mesmo período que a ARPANET, rede predecessora da Internet, viria a conectar diversas instituições educacionais e governamentais nos Estados Unidos. Através da nascente rede de computadores, a cultura *hacker*, com todo um jargão que lhe era própria, além de seu ethos, pode se propagar por todo o pais. Os valores da cultura *hacker* não eram exclusivos aos membros do MIT, mas encontraram aí os termos de uma expressão ou linguagem própria. No início dos anos 80, existiam três culturas que se concentravam ao redor de tecnologias distintas: a cultura do computador PDP-10, ligada ao TMRC, com seu sistema operacional próprio, o ITS (*Incompatible Time-Sharing)* e que utilizava a linguagem LISP; o grupo ligado ao computador PDP-11, ao sistema operacional UNIX, à linguagem C e à utilização de linhas telefônicas; por fim, havia uma "horda anárquica dos primeiros entusiastas de microcomputadores determinados a levar o poder do computador ao povo" (RAYMOND, 2001 [1992], p. 10).[lxvii]

O IBM 704 foi o computador com o primeiro compilador de FORTRAN. Neste momento o problema da compatibilidade de programas dependia diretamente da arquitetura de cada máquina, o que levava à criação de grupos de usuários de máquinas específicas, e das linguagens que aquela máquina suportaria. As linguagens automáticas e de alto nível eram a medida de universalidade de um programa. A partir dos anos 60, os primeiros sistemas operacionais começaram a ser desenvolvidos. Um sistema operacional é um programa que é executado quando o computador inicializa e coordena a execução de outros programas,

mediando seu acesso à máquina. A questão da compatibilidade passava da linguagem, e de seu compilador, ao sistema operacional.

Em 1969, no mesmo ano que a ARPANET é ativada, o sistema operacional UNIX era criado por Dennis Ritchie e Ken Thompson, junto com a linguagem de programação C. O sucesso desse sistema operacional se deu não apenas por causa das inúmeras inovações técnicas que propunha, mas porque seu código-fonte fora distribuído amplamente, ainda que os direitos autorais pertencessem a empresas específicas. A existência da ARPANET colaborou para sua propagação, mas não foi o único meio utilizado. Em 1974, John Lions, da University of New South Wales (UNSW), queria utilizar o UNIX para dar aulas sobre sistemas operacionais. Na ausência de acesso público ao material, Lions escreveu um comentário sobre o código do sistema e conseguiu a autorização da Western Electric, que possuía os direitos daquela versão de UNIX, para publicar, com fins educacionais, o livro com o código e o comentário. Em 1977 cópias do livro foram disponibilizadas ao grande público.

> No decorrer dos anos, por quase duas décadas, o Código e Comentário de John Lions se tornou o trabalho mais copiado em computação. Eles levavam as devidas notificações de direito autoral e restrições a licenças, mas não havia como a Western Electric impedir sua circulação. […]
> Por que se preocupar? Porque aqui estamos no meio da década de 1970 com os usuários assumindo o controle e determinando o que distribuir no que concernia à informação. Felizmente, a Western Electronic não obtivera mais sucesso em controlar a informação do que os Papas Paulo V e Urbano VIII quando Galileu escreveu sobre heliocentrismo. Mas repare novamente: nos anos 70 você recebia o trabalho de Lions em cópia de papel, através de correiro aéreo vindo de Sydney, Austrália (SALUS, 2008, cap. 3).[lxviii]

Desde a criação do grupo SHARE e durante as décadas posteriores, as questões legais acerca da propriedade do código-fonte dos programas e dos limites de sua utilização, cópia, modificação e distribuição eram, no mínimo, ambíguos e incertos. Mesmo quando havia uma clara declaração de posse através de licenças e *copyright* isto não impedia que o código fosse compartilhado e modificado por seus usuários, à revelia daqueles que seriam seus donos oficiais. Existia um clima de permissividade, e até mesmo de camaradagem, entre os usuários e as diversas culturas de usuários e programadores no que concernia ao compartilhamento e distribuição de *software*.

Durante os anos 60 e 70, o software tornou-se uma comodidade, um produto que passou a ser vendido em separado do hardware. Com o surgimento do computador pessoal, no final dos anos 70, o mercado de software cresceu na mesma medida em que os computadores tornavam-se mais populares, não se limitando aos *mainframes* de empresas, universidades e instituições do governo. Neste mesmo período, incluindo os anos 80 e 90, as leis de *copyright* e direitos autorais passaram a ser aplicadas também aos programas, garantindo-lhes a proteção

dada à propriedade intelectual e os meios legais de seus detentores garantirem a exclusividade de seu direito. (COLEMAN, 2012)

Na medida em que o software tornava-se cada vez mais proprietário, sob a égide de novas leis de direitos autorais e patentes, entrava em conflito com toda uma cultura que havia se formado sobre o livre compartilhamento dos códigos produzidos. Mais precisamente, o direito autoral e as patentes passavam a ser aplicadas ao código-fonte dos programas, isto é, o código tal como era escrito em uma linguagem de programação. Com a compilação, a linguagem de programação é traduzida para código de máquina. A partir de um programa compilado é extremamente difícil reconstituir o código-fonte que o originou, pelo menos de uma forma minimamente compreensível. Por isso, é necessário dispor do código original para que um programa seja modificado. Se um software é uma mercadoria, o acesso ao código permite que concorrentes copiem facilmente suas funcionalidades, ou que usuários utilizem o código sem a necessidade de pagar. É nesse ponto que entra o direito autoral e o *copyright*, como dispositivos legais capazes de limitar as cópias não autorizadas de software.

Richard Stallman, um dos membros do Laboratório de Inteligência Artificial do MIT, durante os anos 80 percebia que a cultura que tanto estimava estava sendo substituída por um mundo cada vez mais proprietário. Stallman encontrava-se no berço da cultura *hacker* em um mundo em que, progressivamente, o compartilhamento passava a ser um valor secundário. Em 1984, Levy (2010 [1984]) o chamou de "o último *hacker*", como sendo o remanescente de uma cultura em extinção. Até mesmo as universidades deixaram de compartilhar o código de seus programas, chegando mesmo a exigir que o código produzido em seus computadores fosse licenciado à universidade (COLEMAN, 2012). Na tentativa de lutar contra esse fechamento progressivo do software, buscando restaurar a cultura de compartilhamento, Stallman, em 1985, publica o "GNU Manifesto", no qual propõe que

> a Regra de Ouro requer que se eu gostar de um programa, devo compartilhá-lo com outras pessoas que também gostem. Comerciantes de software querem dividir e conquistar os usuários, fazendo com que cada um concorde em não compartilhar com os outros. Recuso-me a quebrar a solidariedade com outros usuários desta forma. […]
> Para que possa continuar a usar computadores sem desonra, decidi reunir um conjunto suficiente de software livre para que possa seguir em frente sem nenhum software que não seja livre. (STALLMAN, 2010 [1985], p. 34) [lxix]

O manifesto era uma versão mais ampliada de uma mensagem que postou em dois *newsgroups* em 1983, na qual propunha a criação de um novo sistema operacional, chamado GNU. Para utilizar programas que não fossem proprietários era fundamental o desenvolvimento de uma plataforma livre sobre a qual pudessem ser executados.

> Ao trabalhar e usar o [sistema] GNU ao invés de programas proprietários, podemos

> ser hospitaleiros com todos e, também, obedecer a lei. Além disso, o [sistema] GNU serve como um exemplo inspirador e como uma bandeira para chamar outros a participarem do compartilhamento conosco. Isto pode nos dar um sentimento de harmonia que é impossível se utilizarmos software que não é livre. Para cerca da metade dos programadores com quem falo, esta é uma importante felicidade que o dinheiro não pode substituir. (STALLMAN, 2010 [1985], p. 35).[lxx]

A expressão "GNU" é um acrônimo recursivo de "*GNU's not Unix*", o que marcava a sua intenção de ser um clone de UNIX, garantindo sua compatibilidade com os programas escritos para os dois sistemas operacionais. O desenvolvimento do sistema GNU seria realizado de forma aberta, pela contribuição de diversos voluntários, assim como outras ferramentas que constituiriam o sistema foram desenvolvidas anteriormente. Para que isto fosse possível, a batalha contra o modelo proprietário não deveria ser colocada apenas ao nível técnico, dado que a maior ameaça à cultura hacker encontrava-se no aparato legal do direito autoral. Era necessário *hackear* a própria lei, e introduzir, na controvérsia envolvendo sistemas operacionais e liberdade, uma licença de *copyright* alternativa. A primeira versão da "GNU General Public Licence" (GNU GPL ou simplesmente GPL), foi publicada em 1989.

> A GPL é uma licença que ao mesmo tempo em que é construída sobre a lei do copyright, inverte os princípios tradicionais deste. Ao invés de garantir ao proprietário o direito de restringir cópias, o detentor de um copyright garante aos usuários o direito de copiar e compartilhar programas. E a GPL vai além, projetando-se em versões futuras: ela se comporta como uma barreira [firewall] legal contra a ameaça de um posterior fechamento proprietário. Futuras versões de um programa distribuído e licenciado sob a GNU GPL devem permanecer sob a mesma licença e, portanto, também podem ser usadas, compartilhadas, modificadas e distribuídas por outros usuários. (Isto difere de colocar o software no domínio público, uma vez que o material lançado desta forma pode ser subsequentemente incorporado em um novo trabalho, que por sua vez pode ser protegido por direitos autorais [copyrighted]) (COLEMAN, 2012, p. 70) .[lxxi]

Um programa que seja desenvolvido a partir de outro licenciado sob a GPL deve conter a mesma licença, ou seja, deve poder ser usado, distribuído, modificado e compartilhado sem impedimentos. Isto cria uma cadeia de programas que possuem as mesmas licenças e preservam as liberdades de uso, compartilhamento, distribuição e alteração sem correrem o risco de poderem ser fechados em algum momento. A GPL é, nesse sentido, uma "licença viral", pois se replica nas produções derivadas a partir do trabalho original.

O que o movimento do *software* livre propõe é uma abordagem ética e política do *software*, colocando a liberdade como valor central. Um dos modos de atingir esse objetivo seria pela exortação a que os programadores abrissem seu código e o compartilhassem como era feito antigamente. Entretanto, em um mundo onde o *software* transformou-se em um mercado milionário, cercado por patentes e direitos autorais, um apelo à ética não seria forte o suficiente para mobilizar muitos programadores a abrirem seus códigos sem nenhuma garantia de que não seriam logo apropriados e fechados por outrem. A ética do *software* livre é

incorporada na GPL, a licença permite que os valores da liberdade propostos pelo movimento sejam articulados ao aparato jurídico já existente, conectando-se a um corpo instituído e sólido de práticas. Ao invés de ter de convencer todos os desenvolvedores, empresas e organizações a respeitarem a liberdade do código e não fechá-lo, a GPL efetivamente permite que o código seja fechado sem ter de modificar milhares de sujeitos e instituições.

Neste caso, parafraseando a questão de Latour (1992), onde está a moralidade, ou a ética? Nos sujeitos, que são dominados pela imposição de uma lei? Na própria lei, que força os sujeitos a obedecerem? No código, que incorpora a lei ao reproduzir uma notificação de direito autoral? Aqui reencontramos o problema da agência dos atores e dos programas de ação, e só podemos pensar a agência sem reduzi-la a um modelo assimétrico se considerarmos que a ética do *software* livre constitui um híbrido composto por programadores, códigos, licenças e comunidades.

A ideia de abrir o código, e as quatro liberdades propostas pelo *software* livre dão um novo sentido à universalidade. Até este momento, a universalidade era considerada como a possibilidade de *execução* de qualquer programa dentro das limitações da máquina. Esta seria a primeira liberdade. O *software* livre *começa* com a noção básica de universalidade (apesar de não referenciá-la nestes termos). A ela adiciona a segunda liberdade, de ter acesso ao código-fonte, podendo modificá-lo. A terceira liberdade permite que cópias dos programa sejam redistribuídas, enquanto a quarta autoriza que as cópias modificadas também o sejam (STALLMAN, 2010 [1996]).

Para quem deseja apenas executar um programa, a princípio, a primeira liberdade é a única necessária. O conjunto delas permite, entretanto, que uma *comunidade* seja formada. As liberdades de redistribuição existem para permitir que os programas circulem amplamente sem restrições, enquanto a segunda liberdade incorpora o espírito *hacker* de um usuário poder modificar, melhorar ou simplesmente estudar o *software* que adquiriu. A distribuição permite que uma comunidade de trocas e compartilhamento seja formada, enquanto a liberdade de modificação constitui uma comunidade de criação e colaboração coletiva.

A universalidade, transformada em liberdade, atravessa as problematizações anteriores de modo perpendicular. Do modo como é formulada pelo *software* livre, a universalidade enquanto liberdade não é nem geral, nem específica, não está ligada diretamente a uma arte negra ou a uma formalização. É, sobretudo, o que permite e precede tais distinções. Muitos dos debates e controvérsias que acompanhamos neste capítulo não seriam possíveis se o código não fosse compartilhado livremente. Se o código de compiladores de FORTRAN ou COBOL jamais saíssem da tutela da IBM ou dos grupos que

os conceberam não teriam se tornado um padrão da indústria e da academia. Se os sistema operacional UNIX, em toda a sua pletora de versões, não fosse ativamente pirateado não teria se tornando o meio de comunicação comum entre *hackers*.

A universalidade como liberdade proposta pelo movimento do *software* livre constitui-se como um modo evitar a dissociação das antigas práticas de compartilhamento e formação de coletivos, pautando os possíveis modos de desdobramento da universalidade e de controvérsias a partir de um aparato jurídico. Ainda que encarne o espírito *hacker*, em tudo o que este tem de arcano e artesanal, o *software* livre acaba funcionando mais como uma plataforma ou caixa de ferramentas para os mais variados fins. Este mesmo ethos viria a servir de base para a criação de outras licenças de *copyleft* que não seriam exclusivas ao código de programas, aplicando-se a qualquer produto cultural, como a licença *Creative Commons*. Isto permitiu que diversas áreas da produção de conhecimento lutassem por uma abertura da produção do saber, sendo a Wikipédia o caso mais exemplar, acompanhada por periódicos científicos, projetos artísticos e outros modos de produção de cultura.

Em 1996, Eric S. Raymond publica a primeira versão do livro *The Cathedral and the Bazaar*, no qual descreve a sua experiência com o desenvolvimento de um programa de código aberto, traçando paralelos com o desenvolvimento do sistema operacional Linux. O modelo da catedral é como o *software* livre era produzido até aquele momento: o código era trabalhado por um grupo pequeno de desenvolvedores, com uma liderança forte e presente, e que lançaria novas versões do programa e do código apenas quando estivesse pronto, tal como a cuidadosa construção de uma catedral por "feiticeiros individuais ou pequenos bandos de magos trabalhando em esplêndida isolação" (RAYMOND, 2001 [1998], p. 21).[lxxii] Por outro lado, o modelo do bazar, que veio a ser o modo em que o desenvolvimento do sistema operacional Linux adotou, não acontecia mais em espaços silenciosos veneráveis, mas em um "bazar tagarelante de diferentes agendas e abordagens […] do qual um sistema coerente e estável aparentemente poderia emergir apenas por uma sucessão de milagres" (RAYMOND, 2001 [1998], p. 21–22).[lxxiii]

O sistema GNU proposto por Stallman em 1983, ainda não estava completo em 1991. Apesar de inúmeras ferramentas e partes importantes terem sido desenvolvidas, faltava um componente essencial, o *kernel* que, como o nome implica, é o núcleo do sistema, a parte responsável por comunicar-se diretamente com o *hardware* e tornar este acessível aos outros programas. Linus Torvalds resolve criar um sistema operacional próprio, inspirando no MINIX, um sistema operacional similar ao UNIX, desenvolvido por Andrew Tanembaum para fins educativos. O novo sistema operacional seria também um clone de UNIX e fora

criado utilizando as ferramentas desenvolvidas pelo projeto GNU. Tendo preparado uma primeira versão ainda muito incompleta, e com uma licença própria, Linus distribui seu código em listas de e-mails, pedindo sugestões e opiniões. Logo, começa a receber contribuições de código de centenas de desenvolvedores do mundo inteiro, que assim como Linus e Stallman, também desejam ter acesso a um sistema operacional livre e com as mesmas potencialidade do UNIX (RAYMOND, 2001 [1998]; SALUS, 2008).

Posteriormente a licença do sistema foi alterada para a versão 2 da GPL, e todo o desenvolvimento se deu pela contribuição de um grande número de desenvolvedores do mundo inteiro, com Linus coordenando o projeto, função que desempenha ainda hoje. Linux foi o primeiro grande projeto de *software* livre e, dada a sua extensão, não poderia seguir o modelo fechado da catedral. O modelo do bazar, apesar de não ser uma resposta direta à crise do *software*, ou pelo menos não a referencia diretamente, acaba cruzando com diversos problemas, temas e soluções propostos.

Primeiro, a questão do gerenciamento. Um programa de código aberto não se encontra necessariamente atrelado a nenhuma empresa específica, e muito de seu trabalho é realizado por voluntários. Deste modo, o gerenciamento do projeto não pode se dar nos mesmos termos da relação patrão-empregado ou em uma perspectiva exclusivamente guiada pelo lucro ou interesses privados. Um *software* livre arregimenta os mais variados e até mesmo discrepantes interesses, que podem ser tanto guiados pela vontade de criar um produto lucrativo, quanto por anseios anti-capitalistas. Os projetos de software livre podem assumir desde modelos centralizados de gerenciamento, como o do "ditador benevolente" (como é o caso do Linux), com modos mais descentralizados e democráticos. Em ambos os casos, a questão é como conseguir realizar um compromisso entre as mais variadas agendas e interesses de modo a construir algo que funcione.

No que concerne aos erros e *bugs*, o modo de produção aberto possui muitas similaridades com a programação sem ego proposta por Weinberg. A melhor forma de evitar erros é permitir que o código seja visto e analisado por outros, ou, como coloca Raymond, "com olhos suficientes, todos os *bugs* são triviais" (RAYMOND, 2001 [1998], p. 19).[lxxiv] Se o código é fechado, apenas os desenvolvedores com acesso a ele podem localizá-lo. Ao abrir o código este torna-se acessível a um número muito maior de olhos e possibilidades de identificação e correção. A isto soma-se o fato de que qualquer um pode, a princípio, escrever uma correção ao problema e submetê-la para ser avaliada e incorporada no código. No modelo do bazar, o programa é lançado frequentemente incompleto. Isto permite que erros e problemas sejam identificados e corrigidos com maior frequência durante o desenvolvimento,

além de permitir que os usuários, mesmo sem nenhum conhecimento e programação, tornem-se participantes ativos do processo na medida em que podem encontrar *bugs* ou sugerir modificações.

Em inglês, o termo "*free*" é ambíguo, pois designa tanto liberdade quanto gratuidade, o que levou Stallman a cunhar o slogan "*free as in speech, not as in free beer*". Um *software* livre, ainda que seu código seja livremente acessível e compartilhável, não precisa ser gratuito. Um *software* proprietário é comercializável justamente porque seu código-fonte não é acessível, o que dificulta o acesso ao programa fora dos meios oficiais e legais disponibilizados por seus desenvolvedores. Neste sentido, um programa de código aberto é muito mais difícil de comercializar, sendo que geralmente são lançados de forma gratuita. A comercialização viria a partir do suporte dado e não do programa diretamente.

A ambiguidade do termo *free* e a percepção de que todo *software* livre deveria ser gratuito, unida à militância do movimento do *software* livre levou, em 1998, à criação da Open Source Initiative. Para Stallman e outros membros do movimento, a utilização de *software* livre era uma questão ética ou, mais precisamente, moral. Todo *software* proprietário é mau, pois necessariamente priva e rouba a liberdade dos usuários. Nenhum bem pode advir de sua utilização. O problema é colocado em termos absolutos, sem margem para negociação. O termo *open source*, código aberto, foi utilizado como um alternativa mais pragmática, e liberal, à proposta do *software* livre. A Open Source Iniciative congregou diversos membros da indústria, incluindo Linus Torvalds, que adotavam uma atitude mais pragmática e comercial em relação ao *sofwtare* de código aberto, buscando criar uma imagem que fosse mais comercializável. O que estava, e ainda está em jogo nesta controvérsia é a própria definição de liberdade. Para o *software* livre a liberdade é o resultado de certas restrições que impedem o fechamento, enquanto para o *open source* a liberdade deve permitir, inclusive, o fechamento, desde que o código original se mantenha aberto.[19] Na prática, contudo, nem sempre há diferenciação entre os dois grupos, cujo modo de produção é similar e intercambiável, o que levou à criação do termo FOSS (Free and Open Source Software).

3.7 Universalidades

A história da máquina universal em seus desdobramentos lógico-matemáticos, como foi narrado no primeiro capítulo, é a história de uma progressiva integração, combinação e

19 O que é incorporado em diversas licenças alternativas de *copyleft*, como a BSD.

transformação de conceitos. Tanto a máquina de Turing quanto a máquina universal são prontamente aceitas, ainda que sua adoção e utilização não tenha se dado imediatamente. O que se segue é a história das inúmeras transformações, modificações, reduções e ampliações pelas quais passaram, associando-se a neurônios, circuitos, autômatos finitos e infinitos, gramáticas gerais e linguagens formais. As metamorfoses que sofreram não se deram em decorrência de refutações ou deficiências, mas justamente por conta de sua versatilidade e potencial. O sucesso de uma teoria científica ou objeto técnico é medido não por uma permanência estática e canonizada, mas pelo quanto é retomada, transportada, traduzida e generalizada. A máquina universal tornou-se a máquina proteana por excelência.

O desenvolvimento dos primeiros computadores eletrônicos teve como consequência a necessidade de que a máquina universal fosse modificada de modo a se aproximar ao funcionamento das novas máquinas. Ao invés de colocar um problema aos modelos existentes, o computador eletrônico foi incorporado e utilizado para fortalecer e noção de máquina universal, na mesma medida em que esta se transformava no modelo abstrato do computador. Simultaneamente, o computador eletrônico não poderia esperar uma teoria para ser utilizado. Sua construção era o resultado de necessidades práticas e concretas, o que levava à sua programação e manutenção.

Enquanto a relação do computador com a teoria se deu inicialmente de modo não-problemático, com a programação, por sua vez, constituiu uma história repleta de crises e controvérsias. Na teoria, o computador era abstraído, simplificado, considerado em suas partes essenciais em modelos e situações ideais, como a máquina de Turing, um computador infinito que jamais errava. Na prática da programação, o computador aparecia em toda a sua complexidade irredutível, tanto ao nível do *hardware* e todas a suas peculiaridades de funcionamento, incluindo defeitos elétricos e mecânicos, quanto ao nível das exigências lógicas e humanas do ato de programar. O que a história da programação nos mostra, ainda que abordada de modo muito breve, é o surgimento do computador e do programa em seu modo de agência próprio. Ao invés de serem constituídos como uma ferramenta ou meio neutro e passivo de passagem à tradução de ideias de um programador, isto é, como um intermediário, a relação entre computador, programa e computador aparece, especialmente em cada crise e controvérsia, como um conjunto de mediadores, como atores que agem a modificam os rumos de ação uns dos outros. É a história de uma co-produção entre máquina, programa e programador, atravessada por uma multidão de atores humanos e não-humanos, mobilizando empresas, governos, técnicas de gerenciamento, provas matemáticas, clubes de hackers, dispositivos jurídicos e universalidades.

A máquina universal não teve um papel proeminente na história de seu desenvolvimento teórico. Constituía mais um episódio ou subclasse de máquinas de Turing ou autômatos. Funcionava como exemplo da potencialidades da máquina, ou como meio de demonstração da solução negativa ao *Entscheidungsproblem*. Em nenhum momento foi problematizada, mesmo quando era reimplementada, formalizada ou restrita por exigências de finitude. No caso da programação, a universalidade desempenha um papel ativo em diversas circunstâncias. Como promessa de generalização ou como abstração, virtualização e tradução, o que permitira a criação de linguagens de programação independentes de máquinas e focadas em problemas.

A promessa logo transformou-se em controvérsia, na medida em que sua implementação efetiva potencializou a oposição existente entre duas tradições, uma que valorizava a generalidade, correção e elegância, e outra que tinha como valor a especificidade e eficiência. É aqui que as duas linhas se encontram, as exigências concretas da programação e as necessidades abstratas da máquina universal. No campo da teoria, a máquina de Turing marcou a introdução da noção de máquina no pensamento lógico-matemático. Ainda que fosse formalizável, isto não consistia em uma necessidade, permitindo que se desenvolvesse todo um pensamento acerca de máquinas abstratas que não poderia ser completamente reduzido à prática matemática. Na teoria, já encontrávamos uma distinção entre programa e lógica, entre algoritmos e as tabelas de instruções de Turing e as proposições, provas e teoremas.

Esta distinção entre algoritmos e lógica foi amplificada e multiplicada no âmbito da programação, transformando-se em controvérsias com consequências e desdobramentos profundos tanto para a prática do programador quanto para uma ciência da computação. Os defensores da generalidade assumiam a ideia de universalidade como um modo de abstrair a máquina, isto é, de afastá-la de suas condições materiais e concretas, simplificando-a e tornando-a capaz de ser controlada e compreendida. A busca de eficiência, contudo, não implicava menos universalidade, pois ao restringir as propriedades da linguagem, *mais* máquinas poderiam utilizá-la. Enquanto uma se afastava da máquina como caminho para a universalidade, a outra aproximava-se cada vez mais. O perigo da primeira encontrava-se na possibilidade de criar uma linguagem desconectada da realidade, enquanto o perigo da segunda era justamente de perder-se nos meandros e labirintos eletrônicos das máquinas.

Encontramos aí duas universalidades que são produzidas e constituem coletivos e possibilidade de ação diversas. A primeira universalidade segue a leitura da máquina universal de Turing como abstração e virtualização, é o universal propriamente dito, que se descola de

qualquer particularidade, que opera por classes e definições *a priori*. Esta virtualização, por sua vez, age de modo a criar um ambiente mais lógico e estabilizado ao programador, que não se perderia mais com tanta facilidade entre as vias tortuosas do código. A outra universalidade, de certo modo, preocupa-se mais com o alcance efetivo do que absoluto, onde a universalidade opera como agente de mobilização de uma comunidade de máquinas, desde que cada maquina seja considerada como já sendo um coletivo. Uma universalidade estabiliza e delimita, a outra expande e agrega.

O embate entre uma perspectiva mais específica e outra mais genérica não é exclusiva do campo da programação. Como Simondon (2007) argumenta, é uma tensão coextensiva ao aparecimento das técnicas e das ciências modernas, começando no Renascimento. A princípio, existiram dois grandes modos de relação do homem com a técnica, que, tomando os termos do pensamento iluminista, Simondon nomeia como "minoridade" e "maioridade". A minoridade, como o nome implica, é o modo pelo qual a criança aprende a utilizar a técnica. Mas a minoridade também é, fundamentalmente, o domínio do artesão, pois caracteriza-se por ser um conhecimento intuitivo, aprendido e transmitido pela prática. É um modo de conceber a técnica anterior à ciência, é iniciático e exclusivo, pois exige uma dedicação completa do aprendiz, que pode aprender apenas em contato direto com a matéria e com as ferramentas. Não há nem formalização nem abstração do conhecimento. Este encontra-se diretamente ligado a uma determinada comunidade e não tem a necessidade de um modo de transmissão mais geral. Tal relação com a técnica cria um homem hábil, "aquele que o mundo aceita, que a matéria ama e ao qual obedece com a docilidade de um animal que reconheceu seu amo" (SIMONDON, 2007, p. 111)

A maioridade técnica, por sua vez, define o modo de conhecimento do adulto, representado na figura do engenheiro. É um conhecimento abstrato, composto por gráficos, diagramas, equações. Não parte mais da relação direta com a matéria ou com as ferramentas. Baseia-se em princípios e leis, é científico e indutivo. O engenheiro é aquele que encarna tais esquemas abstratos de pensamento e os converte em um projeto a ser executado, utilizando uma linguagem mais ou menos formalizada capaz de ser lida e compreendida por qualquer um que possua o devido conhecimento "adulto". No que perde de proximidade e especificidade, o pensamento adulto ganha em transmissibilidade e generalidade.

A atividade do artesão é intuitiva, enquanto a do engenheiro é reflexiva, pensa a si mesma e pode explicar seu modo de funcionamento. Ao mesmo tempo, a ação do artesão conhece a especifidade da matéria de um modo que não pode ser encontrado no pensamento do engenheiro. Vemos, então, que há uma discrepância entre os dois modos de pensamento, o

que implica em consequências práticas e pedagógicas distintas, especialmente porque nenhum dos dois modos de pensamento dá conta sozinho da complexidade e especifidade das técnicas. O enciclopedismo iluminista nasce de uma perspectiva predominantemente adulta, pois seu ideal de universalidade e racionalidade o insere neste modo de pensamento. Mas a universalidade da enciclopédia inclui, também, a minoridade, já que trazia não apenas teorias abstratas, mas esquemas de construção, descrição e utilização de ferramentas e objetos técnicos. O enciclopedismo foi uma das primeiras tentativas de compor os diferentes modos de pensamento acerca da técnica. A grande inovação do enciclopedismo iluminista se deu com a invenção de uma linguagem comum ao homem e à máquina, o que abriu a possibilidade de um novo modo de pensamento que concilie as diferentes dimensões técnicas, tal como a cibernética o fez parcialmente no século XX.

A dicotomia entre programação como arte negra e como uma operação formal retoma muitos dos termos da relação entre minoridade e maioridade das técnicas, entre a perspectiva do artesão e do engenheiro. Mas, ao mesmo tempo, as duas tendências não deixavam de possuir afinidades e pontos de encontro. Mesmo nas controvérsias acerca da linguagem ALGOL, os defensores da generalidade tiveram de efetivamente construir um compilador para defender seu argumento, enquanto que, em geral, os programadores, com atitude mais prática ou artística, não abriam mão de utilizar os recursos formais, na medida em que fossem necessários. Como coloca Ensmenger (2010), tanto programadores "artesãos" quanto "engenheiros" estavam mais próximos uns dos outros do que gerentes, membros da indústria, do governo e do exército. Em toda a sua variada gama, indo desde programadores de aplicativos comerciais, sem nenhuma formação em matemática, até os acadêmicos proponentes dos modelos mais abstratos, rigorosos e formais, a programação apresentava-se como um modo de relação com a máquina, que buscava dar conta de sua complexidade nos mais variados níveis e graus, sejam concretos ou abstratos.

A cultura *hacker* constituía um ponto de convergência entre estas duas tendências. Por um lado, o *hacker*, assim como o artesão, é aquele que ama a máquina, que a conhece profundamente por dedicar-se com paixão a desvendar a todos os seus detalhes e particularidades. É o sujeito que coloca a máquina à prova, a fim de levá-la seu limite. Mas junto a esta atitude encontramos o ideal enciclopedista, de que o conhecimento técnico deve ser universal, do saber que deve ser multiplicado pela transmissão e acréscimo, transformação e contribuição de seus pares. Como De Millo, Lipton e Perlis (1979) afirmaram em sua crítica à verificação formal, o matemático também não perde a oportunidade de compartilhar sua prova, divulgá-la, colocá-la a prova entre seus pares. É justamente essa socialização que

garante a certeza desta prova, sendo que o mesmo se aplica às teorias formais da computação. O código de um programa escrito seguindo os moldes mais formais não é menos compartilhável que um *hack*, um código "rápido e sujo" escrito para dar conta de um problema específico. E nada impede que os dois programas possam ser combinados, misturados e acoplados. O *software* livre, e depois o *open source*, entram como propostas de uma universalidade que permite colocar em contato as duas práticas.

As dicotomias, encontros e desencontros acerca da natureza da programação, do código e de uma ciência da computação não são o resultado apenas de convenções e práticas entre determinados grupos de sujeitos. Se há consenso e conflito é porque as máquinas encontram-se diretamente implicadas neste processo, onde todos os encontros e vicissitudes dependem do modo como se dão os agenciamentos efetivos, as co-produções entre toda a gama a variedade de atores envolvidos.

O computador eletrônico, que pode ser desdobrado tanto em *hardware* e *software*, não pode ser nunca considerado como uma entidade única. Como Mahoney (2011) nos lembra, não há "o computador", mas computadores. E esta pluralidade deve ser tomada em seu sentido extremo, pois sob este termo podemos encontrar desde máquinas gigantescas, compostas por milhares de tubos de vácuo e cuja programação se dá utilizando cabos, painéis de interruptores, fitas ou cartões perfurados, até computadores pessoais em que a programação não é muito diferente de escrever um texto. São experiências e coletivos completamente distintos. A arte negra da programação dos anos 50 não é a mesma das décadas posteriores.

Mesmo que cada uma das máquinas seja pensada independentemente de extrema diversidade de arquiteturas com a qual coexiste, ela não será nem um objeto simples nem homogêneo. Para Simondon (2007), o objeto científico apresenta uma certa simplicidade, ou mais precisamente, uma coerência, pois é construído de tal maneira que permite a compreensão clara e analítica daquilo que se propõe a modelar. O objeto técnico, por sua vez,

> muito longe de situar-se por completo no contexto de uma ciência particular, está de fato no ponto de convergência de uma multidão de dados e efeitos científicos que provém dos domínios mais variados, que integram os saberes aparentemente mais heteróclitos, e que podem não estar intelectualmente coordenados, enquanto que o estão na prática e no funcionamento do objeto técnico; pode-se dizer que o objeto técnico resultava de uma arte de compromisso... (SIMONDON, 2007, p. 128)

Como objeto técnico, o computador é o compromisso entre inúmeras exigências e interesses. Assim como a ciência da computação, encontra-se na convergência de pelo menos três vertentes: matemática, engenharia e ciência. É um objeto a ser construído, operado, programado, e que deve executar programas que, devido a ser uma máquina de propósito

geral, podem variar amplamente. Podemos entender o computador como sendo produto de uma *engenharia heterogênea*, "considerada como a associação de elementos não cooperativos em redes auto-sustentáveis que são, de acordo, aptas a resistir à dissociação" (LAW, 1987, p. 114).[lxxv] De algum modo, em meio a tantos interesses distintos e até mesmo contraditórios, o computador pode mantê-los associados e em certa medida estabilizados. Nesta associação, cada grupo considerado limitaria a heterogeneidade do computador às propriedades que lhe seriam mais interessantes, como os aspectos formais, as especifidades de funcionamento, ou a mera utilização de um ou outro programa. Mas é justamente a heterogeneidade que permite essa coexistência e separação de interesses, e até mesmo a busca de homogeneidade.

Cada computador deve ser considerado como uma rede ou uma entidade emergente, que existe apenas no encontro de diversas forças. Mas isto vale, também, para todos os envolvidos neste processo. Um computador não é uma entidade estática, deve, ao contrário, ser considerado como um *processo de formação de grupos ou coletivos*. Não é o programador que escolhe programar desta ou daquela maneira uma máquina que aguardaria impassivelmente suas instruções. É o processo, o agenciamento de que ambos participam que constitui o programador enquanto tal e lhe dá determinadas possibilidades de programação, mas apenas porque o computador efetivamente age desta ou daquela maneira, o que limita, amplia ou modifica tanto agência quanto o próprio ser do programador.

Podemos entender a relação entre computador e programador, e os modos como a universalidade vem a ser constituida, considerando a noção de *co-produção,* entendida como "a proposição de que os modos pelos quais conhecemos e representamos o mundo (tanto natureza e sociedade) é inseparável dos modos pelos quais escolhemos viver nele" (JASANOFF, 2004, p. 2).[lxxvi] Sociedade e natureza são co-produzidos, um não precede o outro. Quando o computador se apresenta como uma máquina "essencialmente" complexa, seja por operar com milhares de tubos de vácuo que falham regularmente e que mesmo assim permitem uma computação confiável, seja por se apresentar como logicamente labiríntico e complicado, tais propriedades não pré-existem ao agenciamento em que se constituíram. O computador se tornou complicado de programar apenas depois de ser complicado de construir. Como disse Dijkstra (1972[a]), "enquanto não havia máquinas, programar não era um problema; quando possuíamos poucos computadores fracos, programar tornou-se um problema médio, e agora que temos computadores gigantescos, programar tornou-se um problema igualmente gigantesco" (p. 861).[lxxvii]

O mesmo se dá em relação à universalidade. Quando Turing concebe a máquina universal, a natureza da universalidade não é questionada. Em nenhum momento encontramos

uma definição do termo "universal" ou "universalidade", o seu significado aparece como algo dado ou óbvio. Todas as transformações que sofre na lógica, na matemática e na teoria dos autômatos apenas a confirma como um princípio lógico. É o que podemos chamar de *fatiche*[20], como algo que é um fato ao mesmo tempo em que é feito, ou mais precisamente, é fato *por ser fabricado* (LATOUR, 2002). No fatiche, aquilo que é criado supera aquele quem o criou. Ainda que seja fabricado, o fato, na medida em que é fortalecido, adquire uma independência ou autonomia em relação às suas condições de produção. Por estar razão, um fato científico pode ser considerado como objetivo, como algo que está "lá fora" e é independente do sujeito. É este o acontecimento que faz com que uma lista de ações, um actante, passe a ser um ator, em que vamos do desempenho à competência.

Na construção e utilização de computadores, contudo, a universalidade não é algo tão certo. Será que estes milhares de circuitos e tubos de vácuo serão capazes de funcionar de tal modo que um programa possa ser executado? Será que é possível construir um compilador de ALGOL que seja verdadeiramente universal? Estas questões marcam momentos em que a universalidade, quanto é transposta ao mundo das máquinas e dos programas, ainda não fora estabilizada. A universalidade é co-produzida, pois coloca problemas e modifica-se de acordo com as soluções encontradas e caminhos tomados. Mas a co-produção implica, também, hibridização e indiscernibilidade, pois nem sempre são claros os atores que estão em jogo. Um computador é uma mistura, um agregado, e ora o programador lhe é uma entidade externa, ora faz parte dele. Toda a história da programação é atravessada pela vontade de transformar o programador em um *coder,* um computador humano acoplado ao computador eletrônico que simplesmente traduz ideias em código de máquina, de modo completamente mecânico e automático, indo desde as ENIAC Girls e o papel que lhes fora atribuído, até as tentativas de reinserir o programador em estruturas tradicionais de gerenciamento. O programador deve ser um intermediário. Computador, programa, universalidade e programador são todos híbridos, que se distinguem apenas em momentos e situações específicas. Como será visto no próximo capítulo, quando Turing propõe o Jogo da Imitação, o faz tanto como um modo de distinguir humanos de computadores, quanto de permitir que o humano e a máquina sejam indiscerníveis.

As tentativas de reduzir a programação à matemática são um exemplo de como a especificidade da programação pode ser produzida. Vimos que, desde Turing, a tradução do

20 Em português os termos "fatiche" e "feitiche" são traduções possíveis do neologismo *faitiche* criado por Latour. No original a busca mostrar que o fetiche é tanto fato, quanto feito, sendo que no português o termo "fatiche" rdá ênfase ao primeiro significado, enquanto "feitiche" se refere ao segundo. Utilizamos "fatiche" ao invés de "feitiche" (que foi utilizado na tradução brasileira de Latour (2002)) porque nos interessa dar ênfase a seu caráter de fato.

código para a linguagem formal da lógica é perfeitamente possível. Contudo, cada um está ligado a diferentes conjuntos de práticas. Provar um teorema não é o mesmo que executar uma instrução. A distinção que veio a se produzir, se deu entre o caráter declarativo da lógica e o funcionamento operativo das linguagens de programação (PRIESTLEY, 2011). O objetivo da lógica é definir a verdade ou falsidade de proposições, o que a leva à utilização de provas, demonstrações e derivações. Uma instrução de um código não é nem verdadeira nem falsa, é um comando, uma operação que modifica um determinado estado de coisas. Claro, a lógica binária pode ser considerada como uma sucessão de estados verdadeiros ou falsos, zeros e uns, mas são antes ordens do que constatações. Junto a estas definições veio somar-se o fato de que, especialmente com o advento da programação comercial, que criou um exército de programadores sem nenhuma formação matemática, era possível programar efetivamente sem nenhum conhecimento lógico ou matemático.

E é em meio à co-produção e formação de grupos e coletivos que reencontramos os *hackers* e o *software* livre. Não existe *hacker* sem máquina. Cada máquina, por sua vez, permite que diferentes comunidades se constituam, com todos os conflitos e territorialidades que lhe são próprios. Basta ver a distinção que Raymond (2001 [1992]) fez entre *hackers* do PDP-1, com ITS, e dos hackers do PDP-10 com UNIX, além dos usuários do computador pessoal. O movimento do *software* livre, por sua vez, coloca a universalidade como central. À medida em que a desloca da simples ideia de execução de programas, situando-se na problemática mais ampla da liberdade, e dos meios jurídicos para que o código possa ser compartilhado, modificado e distribuído, o que entra em jogo é a possibilidade da universalidade, neste sentido ampliada, ser o vetor de formação de coletivos.

O ponto de convergência do coletivo deixa de ser o computador, e passa ser o sistema operacional ou um determinado programa. A abertura do código permite que um grupo de interessados se constitua em volta daquele programa específico. Este agrupamento dá margem a que sujeitos com os mais variados interesses possam colaborar, ainda que isto não exclua conflitos e impasses. No mundo do *software* livre e do *open source,* é comum a prática do *fork*, em que o código de um programa é clonado e modificado em separado, levando à criação de um outro programa que não é mais reintegrado ao primeiro. A liberdade se traduz não apenas como consenso, mas como dissidência e fragmentação. A prática da engenharia heterogênea nem sempre consegue produzir uma rede estabilizada, mas o conflito pode levar à multiplicação de redes a estabilização em outro nível.

Podemos ver a heterogeneidade, também, na ação das licenças jurídicas. Não há projeto de *software* livre ou *open source* sem a utilização de uma licença de *copyleft* que

garanta a abertura ou a utilização do código. A presença ou não de um aviso legal no início do programa altera completamente o campo de possibilidades do que pode ser feito com o programa, e o modo como este pode agir. Tanto o aspecto jurídico quanto social do *software* livre foi transportado para outros âmbitos fora da programação. As licenças de *software* transformam-se em licenças das mais variadas mídias e produções culturais, como o *Creative Commons*, assim como músicas, textos, imagens passam a ser produzidos coletivamente. Não é por acaso que o maior projeto de produção de conteúdo coletivo é a Wikipédia. Aqui reencontramos o ideal enciclopédico em seu nível mais alto de realização. A universalidade proporcionada pelos computadores digitais transforma-se no meio de coordenação de milhões de usuários para construir uma plataforma de conhecimento livre e aberta.

Há, no encontro entre as diversas universalidades, seja como apelo à generalidade, como articulação com a especifidade ou como acesso ao código, a possibilidade de uma nova relação com a técnica e a constituição de uma outra sociedade. Para Simondon (2007), o problema do capitalismo não se dá tanto no fato de que os meios de produção encontram-se nas mãos de poucos. O verdadeiro problema é que o patrão desconhece as suas máquinas. A visão do dono de fábrica, do gerente, é distante e abstrata, considera os objetos técnicos apenas por seus resultados do que por seu modo de funcionamento próprio. De nada adiantaria coletivizar os meios de produção se o conhecimento não apenas de como operar, mas de como construir as máquinas fosse compartilhado também. Eventualmente as máquinas não seriam mais reparadas pois não haveria o saber necessário para tal.

A alienação seria uma questão muito mais de pensamento do que de posse. Mesmo em meio a suas controvérsias acerca da universalidade, as mais variadas vertentes de programadores encontravam-se muito mais próximas uma das outras do que dos gerentes. A cultura *hacker* e as propostas do *software* livre vieram reforçar esta posição, ao fortalecer uma cultura e um estilo de vida que valoriza muito mais o conhecimento e o *know-how* do que distinções hierárquicas de autoridade. A questão é como, em meio a esta abertura do código, retomar o ideal enciclopédico de modo a transformar o conhecimento das máquinas, e a possibilidade não apenas de sua utilização e posse, mas de sua modificação e transformação, como um conhecimento verdadeiramente universal e capaz de operar mudanças sociais e políticas.

Esta breve história nos permitiu vislumbrar alguns modos pelos quais se constituiu a paisagem contemporânea que congrega humanos e não humanos nos mais diversos coletivos e práticas. Vimos como a universalidade é transformada, como seus sentidos são multiplicados em meio a controvérsias aos modos de ação que lhe são próprios. Cabe agora explorar alguns

dos desdobramentos e implicações destas associações em relação ao que implica pensar e ser humano em meio a máquinas universais (Capítulo 4), à concepção da materialidade e da interação (Capítulo 5) e à relação entre subjetividade, cognição e Outro em meio à agência das máquinas (Capítulo 6).

Parte 2: Dobras

4 Máquina

Can these mixed bodies be separated or a logic to these chimeras be found?

- Michel Serres

Finally, we wish to exclude from the machines men born in the usual manner.

- Alan Turing

4.1 Controvérsias do Teste de Turing

De tempos em tempos deparamo-nos com a notícia de que um determinado grupo de pesquisadores passou o "Teste de Turing" ao desenvolver um programa de computador capaz de conversar como um ser humano, o que implicaria sua inteligência. Tal anúncio é acompanhado por uma série de críticas que questionam seus procedimentos e validade. Eventualmente o próprio teste de Turing é questionado. Segue-se em então uma discussão, de extensão variável, acerca da natureza da inteligência e do pensamento, a possibilidade de sua implementação em máquinas, e a melhor forma de avaliar e julgar a sua presença ou ausência, onde o teste de Turing é tanto criticado quanto defendido, ou sujeito a "atualizações". Todo este ciclo repete-se quando, meses ou anos depois, aparece um novo grupo anunciando seu sucesso no teste.

O teste de Turing é uma versão modificada do "Jogo da Imitação" proposto por Turing no artigo "*Computing Machinery and Intelligence*" (TURING, 2004 [1950]) publicado em 1950. Turing deixa de lado a questão "as máquinas podem pensar?" e, ao invés de focar na definição de termos como "máquina" e "pensamento", propõe um jogo cuja execução e desfecho permitiriam afirmar se uma máquina é inteligente ou não. Primeiro, Turing propõe um jogo composto por três participantes: A (homem), B (mulher) e C (juiz). Os três participantes encontram-se separados e podem comunicar-se apenas através de forma escrita mecanizada. O objetivo é que o participante C, o juiz, decida quem é homem e quem é mulher a partir de perguntas que pode fazer aos outros participantes. A participante B (mulher) deve tentar auxiliar o juiz, enquanto o homem, participante A, deve tentar levá-lo a tomar a decisão errada. No "Jogo da Imitação" o homem, participante A, é substituído por um computador. A questão torna-se então: "O que acontecerá quando a máquina assume a parte de A neste

jogo?" (TURING, 2004 [1950], p. 441) [lxxviii]

Caso a máquina consiga enganar o juiz, em uma série de jogos, será considerada pensante, isto é, ao se fazer passar por uma mulher, em oposição à participante B. O teste de Turing simplifica esta fórmula, reduzindo o número de participantes a dois e removendo a referência ao gênero, onde o juiz tenta ativamente descobrir se o outro participante é uma máquina ou não. Apesar das diferenças entre jogo e teste, é por conta de sua afirmação central que todas a controvérsias acerca do pensamento em máquinas se desenvolverá: o pensamento não pode ser julgado por aquilo que *é* (como substância, faculdade ou propriedade apta a ser delimitada em uma definição), mas por aquilo que *faz*, ou mais precisamente, pela relação de *indiscernibilidade* que estabelece com o que *imita*.

O principal ponto de ataque ao teste, e motivo central das controvérsias que o cercam, encontra-se justamente em seu caráter "superficial", pois na medida em que se recusa a definir o que é pensamento – tanto por seus mecanismos específicos, quanto por qualquer outro critério de delimitação preciso –, o teste depende completamente do sucesso de um engodo. Uma máquina pensante é aquela que é capaz de enganar, de *confundir* os outros sujeitos. Claro, quando Turing concebe uma máquina que é capaz não apenas de ser tomada por um ser humano, mas também por um homem que se faz passar por mulher para enganar um juiz, exige um grau extremamente elevado tanto de sutileza quanto de capacidade cognitiva, especialmente na medida em que tem de replicar detalhes muito específicos de interações humanas, incluindo a afetividade. Esta perspectiva é defendida por Dennet (1998). Na outra ponta do espectro encontramos o famoso argumento do "Quarto Chinês" de Searle (1980), em que mostra como uma pessoa que não fala chinês pode, seguindo um conjunto de ordens e comandos, manipular os símbolos da língua chinesa de tal modo que seja indistinguível de um falante nativo, mesmo sem entender nada do que está escrevendo. Similarmente, uma máquina, cuja função primordial é seguir ordens, pode exibir um comportamento considerado inteligente, sem "ter consciência" ou "entender" o que está fazendo.

O que nos interessa neste texto não é discutir a validade do teste Turing ou a exatidão dos argumentos de seus críticos e defensores. Buscamos, ao invés disso, questionar por que após mais de 60 anos ainda retornamos a tal teste, como continua sendo uma fonte tão prolífica de controvérsias, mesmo com (e talvez por causa de) todo o desenvolvimento tecnológico e conceitual da inteligência artificial, assim como muitos outros campos de estudos que são atravessados pela relação entre humano e máquina. Se a controvérsia do teste de Turing ainda não pode ser fechada, seja a favor ou contra, é porque os termos nos quais a

controvérsia é colocada implica a noção de identidade: a máquina ou pode pensar *igual* ao ser humano, ou não pode. Há identidade ou não. Pode-se até argumentar que há uma gradação em tal identidade, obtendo-se uma máquina mais ou menos inteligente, mas tal variação se dá ainda entre dois extremos que mantém o binarismo.

Turing, entretanto, não concebe o jogo nesses termos, e sua recusa em definir o termo "pensamento" é justamente uma forma de evitar as implicações de uma ontologia da identidade. Em outras palavras, o que podemos encontrar nos trabalhos de Turing é uma tentativa de pensar uma máquina que seja *indiscernível* do humano. Isto não quer dizer que humano e máquina sejam idênticos, mas há um ponto onde é impossível estabelecer qualquer critério de identidade, onde *não importa* que quem fala seja humano *ou* máquina. As diferenças próprias a cada um, toda a sua diversidade e heterogeneidade são mantidas sob o ponto de indiscernibilidade. A controvérsia acerca do teste de Turing situa-se justamente no encontro destas duas lógicas: lógica de pressupostos e lógica de preferências, para utilizar os termos de Deleuze (1997). O teste é concebido a partir de uma lógica de preferências, de um pensamento pragmático focado naquilo que a máquina efetivamente faz e nas ambiguidades de tal ação, mas que por sua vez é retomado por uma lógica de pressupostos que o força a dar uma resposta definitiva e não ambígua sobre a natureza da inteligência, do humano e das máquinas.

Propomos então uma análise aprofundada do pensamento de Turing na perspectiva de uma lógica das preferências, tentando mostrar que mais do que afirmar a identidade entre ser humano e máquina, tornando o primeiro em um computador e o segundo cópia exata do outro, o que Turing faz é *desorganizar* o humano e a máquina a partir da criação de uma zona de indiscernibilidade entre ambos.

4.2 A máquina de Turing

Como foi visto no Capítulo 2, uma primeira versão do problema da relação entre humano e máquina encontra-se já no artigo de 1936, "*On Computable Numbers, with an Application to the Entscheidungsproblem*" (TURING, 2004 [1936]). É um texto que situa-se em uma posição curiosa, pois combina problemas e elementos lógicos com questões e componentes próprios da engenharia, como a necessidade de definir números através de máquinas e operações realizadas por computadores humanos. O ponto central desse texto, no que concerne à relação entre humano e máquina, consiste justamente na ambiguidade existente entre os dois termos, já que o "computador" ao qual Turing se refere pode ser tanto

um computador humano quanto uma máquina.

As máquinas que Turing descreve ocupam o lugar desse computador humano cuja memória é necessariamente limitada. Pode-se dizer então que as máquinas são meras formalizações do comportamento humano, ou até mesmo um comportamento humano idealizado. As máquinas descritas são expressas por "tabelas de instruções" que definem passo a passo o que devem fazer. Estas tabelas, contudo, não utilizam a linguagem tradicional da lógica, nem da matemática, possuem uma notação própria. Dentro do discurso matemático Turing desenvolverá o que podemos chamar de "discurso algorítmico". É um discurso que se expressa primeiro por tabelas e posteriormente através de linguagens de programação, cuja formalização se dá pela conversão à notação da lógica. Todavia, a conversão *não é necessária*. Em outras palavras, o que aparece no uso de tabelas de instruções é a possibilidade de descrição de algoritmos sem a necessidade de uma formalização estritamente lógico-matemática, da mesma maneira que alguém pode aprender a programar ou escrever um software sem conhecer matemática, especialmente em seu aspecto lógico e formal.[21] O que Turing faz ao descrever o comportamento de máquinas, e humanos, através de tabelas de instruções é abrir a possibilidade de conceber todo um domínio de ação que não se reduz a um modelo lógico e formal estrito, mesmo que tenha sua própria formalização.

As máquinas de Turing são mecanismos de leitura e inscrição de símbolos em uma fita de papel móvel, a partir de tabelas de instruções. A cada momento a máquina apresenta um "estado", determinado a partir da posição da fita, do símbolo lido e da atual instrução da tabela. As máquinas são concebidas de tal maneira pois tentam replicar, em seu nível mais fundamental, o comportamento de um computador humano em sua tarefa de calcular sequencias de números, utilizando lápis e papel. A ideia de "estado" é fundamental aí, pois é o equivalente físico dos "estados mentais" do sujeito. Como Turing (2004 [1936]) coloca:

> [Supomos que] a computação é executada em uma fita; mas evitaremos introduzir o "estado mental" ao considerar uma contraparte mais física e precisa. Ao computador [humano] é sempre possível largar seu trabalho, ir embora e esquecer tudo sobre sua tarefa, e retornar depois para lhe dar continuidade. Se fizer isso, é preciso que deixe uma nota com instruções (escritas em alguma forma padrão) explicando como o trabalho deve ser continuado. Essa nota é a contraparte do "estado mental". Podemos supor que o computador [humano] trabalha de uma maneira tão incoerente que nunca executa mais do que um passo por sessão. A nota com instruções deve permitir que execute um passo e que escreva a próxima nota. Portanto, o estado do progresso da computação em qualquer estágio é completamente determinado pela nota com instruções e os símbolos na fita (p. 79).[lxxix]

Quando Turing estabelece uma equivalência entre os estados da máquina e os estados mentais do sujeito, abre o caminho para duas possibilidades de pensamento. A primeira afirma

21 A independência da programação e da formalização matemática é algo que se apresenta desde os primórdios da computação na década de 1950, como mostra Tedre (2015).

que a mente é composta por estados discretos, tornando-a divisível de forma mais ou menos precisa. A segunda possibilidade considera que homem e máquina são equivalentes, já que neste nível fundamental possuem a mesma arquitetura. Esta é a interpretação mais comum das afirmações de Turing e serve de base à visão clássica da inteligência artificial. Na medida em que há identidade entre mente e máquina, toda mente passa a ser um dispositivo de processamento de símbolos, e toda máquina que seja capaz de manipular símbolos é uma mente em potencial. Por sua vez, o caráter discreto dos estados mentais também constitui um elemento chave de determinadas concepções de inteligência artificial e modelos cognitivos, contudo, ocupa um lugar mais controverso e disputado.

Encontramos na visão clássica de inteligência artificial o que podemos chamar de "lógica de pressupostos" (DELEUZE, 1997). Tal lógica parte da ideia de que há uma conexão necessária e causal entre um termo e outro, ligação que é pressuposta a partir de um princípio de identidade binário. Se A = B e B = C, então A = C. Não é possível que que A não seja igual a C, já que ambos apresentam propriedades idênticas. Portanto, este princípio assegura também a identidade entre mentes e máquinas, pois na medida em que apresentam propriedades isomórficas *é necessário* que sejam *a mesma* entidade ao nível ontológico ou, no mínimo, operacional . Nesta lógica o "pressuposto" age como o que já está dado, ordem ou razão subjacente que determina como as coisas devem ser, lógica "segundo a qual um patrão 'espera' ser obedecido, ou um amigo benevolente, escutado" (DELEUZE, 1997, p. 85). Em tal lógica há uma moral implícita. Sempre que algo é esperado, que *deve* ser reconhecido ou assumido, já faz parte do domínio da lógica de pressupostos.

É nestes termos que o Jogo da Imitação e o teste de Turing são considerados, pois é *esperado* que *julguem* definitivamente se uma máquina é inteligente ou não, sem que haja um meio termo, um terceiro excluído. Logo, se o teste não pode operar esta divisão de modo tão exato é porque deve apresentar algum problema. A lógica de pressupostos, por ser binária, é incapaz de trabalhar com ambiguidades sem reduzi-las a um valor de verdadeiro ou falso, tudo ou nada.

Quando Turing utiliza o termo "computador" sem especificá-lo, em seu artigo de 1936, não fica claro se refere-se a um humano ou a uma máquina. Isto, em um primeiro momento, reforça a ideia de identidade entre os dois termos. Mas talvez esta aparente equivalência não seja tanto o fruto de uma identidade quanto de uma *ambiguidade* entre homem e máquina. O fato de que duas entidades possam ocupar a mesma posição em um determinado campo de ação não implica necessariamente sua identidade ontológica, nem a equivalência de seus gestos. Considerando mais atentamente a descrição dada de um

computador humano enquanto realiza um cálculo utilizando um bloco de notas, nota-se como seu comportamento é, nas palavras de Turing, incoerente. Na medida em que se tenta aproximar ao máximo o comportamento da máquina e do humano, este deve agir de tal modo que não possua nenhum traço de interioridade, cada gesto sendo determinado por uma tabela de instruções externas e pelo símbolo atual no bloco de notas, ao invés de guiar-se por algum estado mental interno[22]. Além disso sua ação seria completamente atomizada, pois executaria apenas uma ação por sessão fazendo com que houvesse uma completa desconexão entre uma operação e outra, tornando o bloco de notas a única ligação entre elas.

A situação extrema e seu personagem incoerente são abstrações criadas por Turing para tornar mais evidente os passos presentes em uma computação matemática realizada por um ser humano, mostrando como é possível sua formalização e sistematização em uma máquina. Contudo, este é o único momento em que a identidade entre humano e máquina é afirmada diretamente. No decorrer do texto o termo computador é sempre utilizado de forma ambígua, sem deixar claro se refere-se a um humano ou a uma máquina. Se há mesmo identidade entre um e outro, por que ela pode ser afirmada recorrendo apenas a exemplos extremos?

No pensamento de Turing existem duas atitudes distintas e complementares acerca da relação entre humano e máquina, que aparecem já em seu texto sobre a computabilidade e que são esclarecidas, como veremos agora, em escritos posteriores. A primeira atitude concerne à diferença entre humano e máquina, que é sempre afirmada através da utilização de exemplos ou situações absurdas. Turing leva a identidade ao seu limite. Indo até as últimas consequências de uma completa equivalência entre humano e máquina, o que nos aparece é uma situação completamente incoerente e extrema. Por sua vez, a segunda atitude vai na direção contrária e afirma uma ambiguidade ou *indefinição* dos termos. Este procedimento fica evidente quando em diversas ocasiões Turing (2004 [1951], 2004 [1950], 2004 [1948]; 2004 [1952]), nega-se a dar uma definição do termo "pensamento" (e, em alguns casos do termo "máquina" também). Quando propõe o Jogo da Imitação como critério de decisão acerca da inteligência de uma máquina, o que está sendo medido não é uma identidade, mas uma *indiscernibilidade*, isto é, a capacidade da uma máquina *imitar* aquilo que há de indefinido no humano, seu pensamento entendido como *capacidade de desorganização*. Serão estes dois movimentos que abordaremos mais detalhadamente agora, indo da afirmação da

22 Pode-se argumentar que Turing propõe um modelo behaviorista do sujeito, completamente desprovido de estados internos causais. Contudo, o sujeito incoerente e vazio que Turing apresenta é antes uma caricatura do behaviorismo, pois é levado ao extremo para facilitar a comparação com uma máquina, e não por que de fato seja desprovido de uma mente.

diferença ao jogo da indiscernibilidade, constituindo não mais uma lógica de pressupostos, mas uma "lógica de preferências" que abrirá caminho à possibilidade de um *devir todo mundo* entre humano e máquina.

4.3 Desorganização e imitação

Dentre as diversas máquinas de Turing encontramos um tipo especial chamado de "máquina universal de Turing". Diferentemente das outras máquinas que são especializadas, a máquina universal opera da seguinte maneira: "quando decidimos que máquina desejamos *imitar*, digitamos sua descrição na fita da máquina universal" (TURING, 2004 [1947], p. 383) [grifo nosso].[lxxx] A máquina universal é a máquina capaz de imitar qualquer outra máquina de Turing, de modo muito similar a um computador moderno. A máquina universal pode apenas imitar, não faz nada por si mesma, assim como um computador que não possui nenhum programa apenas espera alguma instrução a ser executada.

Seria o ser humano uma máquina universal? Para colocar tal questão é necessário realizar as distinções apropriadas. Turing (2004 [1948]) diferencia as máquinas (aqui tomadas em seu sentido geral não se limitando às máquinas de Turing) de acordo com dois pares de critérios. O primeiro as divide entre "discretas" (digitais) ou "contínuas" (analógicas). Máquinas discretas apresentam estados delimitados e separados uns dos outros, enquanto nas contínuas seus estados formam uma linha de gradação contínua. O segundo critério diferencia máquinas "ativas", que causam alguma transformação física direta, e máquinas "controladoras" que operam com informação. Um trator seria uma máquina contínua e ativa, enquanto um computador eletrônico seria discreto e controlador. O cérebro humano, por sua vez, "provavelmente é Contínuo Controlador, mas é muito similar a muitas máquinas discretas" (TURING, 2004 [1948], p. 413).[lxxxi]

Ao nível material encontramos uma distinção fundamental entre humanos e computadores[23], não há uma correspondência completa entre suas estruturas. Claro, o *hardware* de um computador é contínuo, assim como tudo o que é físico, mas na prática opera de forma discreta. Em última instância o que garante seu caráter discreto é a presença de um relógio que a cada pulso marca um ciclo de computação. "Podemos dizer que o relógio nos permite introduzir a descontinuidade no tempo, assim, para determinados propósitos, o tempo pode ser considerado como uma sucessão de instantes ao invés de um fluxo contínuo"

23 A partir deste momento, por brevidade, utilizaremos o termo "computador" como significando "computador eletrônico", a não ser que seja especificado o contrário.

(TURING, 2004 [1947], p. 382).[lxxxii] Computadores precisam operar de forma discreta pois são máquinas determinísticas que funcionam a partir de instruções registradas de forma descontínua. Não pode haver ambiguidade neste nível, o que é impossível em um espaço contínuo.

As operações do cérebro não se dão de acordo com os ciclos de nenhum relógio, mas a partir de todo o sistema neuronal e hormonal, que funciona de modo contínuo e, em certos momentos, como uma máquina discreta. A diferença fundamental entre um cérebro e um computador não se encontra tanto na distinção entre contínuo e discreto, mas na sua relação com a *desorganização*. Sendo uma máquina determinística, o computador segue um traçado linear, indo de uma instrução a outra. Mas nada impediria que um elemento de aleatoriedade fosse inserido em tais computações, culminando no que Turing (2004 [1948]) chamará de "máquina desorganizada". Enquanto uma máquina é construída a partir do objetivo que deve cumprir, o que definirá a disposição e ordenação de suas partes, uma máquina desorganizada é construída de forma completamente aleatória, conectando seus componentes e ações sem nenhum propósito. Tais máquinas seriam, em grande parte, completamente inúteis se não incorporassem uma característica essencial: a capacidade de se auto-modificar e produzir uma auto-organização, o que constitui um modo de aprendizado. Um cérebro seria, a princípio, desorganizado, enquanto uma máquina determinística encontra-se organizada do começo ao fim. A máquina desorganizada, por sua vez, aproxima-se muito mais da estrutura e funcionamento cerebrais.

O aprendizado de uma máquina desorganizada se dá a partir de *interferências*, de informações que lhe são fornecidas de fora. Dependendo do tipo de interferência a máquina pode se organizar de tal maneira que produza um modo de funcionamento significativo ou similar a outras máquinas.[24] Neste sentido uma máquina desorganizada é muito similar a uma máquina universal, e pode mesmo ser organizada a se tornar uma. É neste ponto que podemos estabelecer uma relação entre a máquina universal e o pensamento humano, pois

> tudo isto sugere que o córtex da criança é uma máquina desorganizada, capaz de ser organizada pela interferência de treino adequada. A organização pode resultar na modificação da máquina em uma máquina universal ou algo similar. Isto significa que o adulto seguirá ordens dadas em uma linguagem apropriada, mesmo se fossem muito complicadas; ele não possuiria senso comum, e obedeceria às mais ridículas ordens sem hesitar. Quando todas as ordens fossem cumpridas ele entraria em um estado comatoso ou obedeceria a alguma ordem contínua, como comer (TURING, 2004 [1948], p. 424).[lxxxiii]

24 A organização que se produz a partir da desorganização será trabalhada por Turing (2004 [1952]) também em seu texto sobre a base química da morfogênese. O problema que coloca é de como um sistema de substâncias químicas "que pode ser inicialmente muito homogêneo, vem a apresentar um padrão ou estrutura a partir de uma instabilidade do equilíbrio homogêneo, desencadeado por pertubações aleatórias" (TURING, 2004 [1952], p. 519). Em outras palavras, como a organização surge espontaneamente a partir da desorganização e perturbação.

Encontramos novamente uma das situações extremas de Turing, que leva às ultimas consequências um ser humano tornado máquina universal, cuja existência desprovida de "senso comum" limita-se completamente a seguir ordens. O que isso nos mostra é que mesmo treinado, há algo no humano – e diríamos que na máquina também – que não pode ser completamente organizado, pois "muito do comportamento aleatório da infância permanece no adulto" (TURING, 2004 [1948], p. 424).[lxxxiv] Para Turing a infância, enquanto capacidade de desorganização coexiste com a vida adulta.

Existiria no ser humano, ao menos no que concerne ao pensamento e prática matemáticos, três faculdades. A primeira faculdade, que Turing (2004 [1938]) nomeia *intuição* "consiste em realizar julgamentos espontâneos que não são o resultado de uma linha de raciocínio consciente" (TURING, 2004 [1938], p. 192).[lxxxv] A *engenhosidade* é a segunda faculdade, e constitui a parte formal do exercício matemático, ao organizar proposições, gráficos e esquemas de tal modo que possam ser seguidos de forma explícita, sem a necessidade do uso da intuição (apesar desta poder se apoiar na organização da engenhosidade). É a capacidade de seguir regras e as utilizar para realizar um mínimo de esforço computacional. Por fim, a terceira faculdade, que não é nomeada, concerne à habilidade de diferenciar os problemas interessantes dos não relevantes, portanto, de efetivamente colocar problemas.[25]

O humano que se torna uma máquina universal, ou que age como o computador humano incoerente com seu bloco de notas, existe completamente no domínio da engenhosidade. Na medida em que elimina toda forma de desorganização perde a capacidade de intuição, de encontrar soluções fora de qualquer regra ou sistema formal, assim como de colocar novos problemas. "Converter um cérebro ou máquina em uma máquina universal é mais extrema forma de disciplina. […] Mas a disciplina não é suficiente em si mesma para produzir inteligência. Além disso é necessário ter o que chamamos de iniciativa. […] Nossa tarefa é descobrir a natureza desse resíduo tal como ocorre no homem, e tentar copiá-lo nas máquinas" (TURING, 2004 [1948], p. 249).[lxxxvi] A engenhosidade é resultado da *disciplina*, enquanto intuição e a terceira faculdade são produtos da *iniciativa*, que é esse resíduo de desorganização que persiste por toda a vida humana.

25 A terceira faculdade estaria mais próxima do que Deleuze (1999), a partir de Bergson, define como "intuição", enquanto a engenhosidade equivaleria à inteligência, como capacidade resolver problemas. A intuição tal como é definida por Turing (2004 [1938]) ficaria a meio caminho entre a intuição e a inteligencia deleuziana, pois ao mesmo tempo em que resolve problemas, o faz apoiando-se em um modo de ser que escapa à formalização e regramento. A inteligência em Turing, como veremos, nunca é definida, mas podemos afirmar que não se limita à capacidade resolver problemas. Ou melhor, na medida em que é definida, nunca podemos dizer *a priori* o que pode fazer ou não.

A iniciativa ou desorganização residual do homem é o que deve ser imitado pela máquina. Quando Turing (2004 [1950]) concebe o Jogo da Imitação, é a imitação desta iniciativa que tem em vista, mais do que uma classe de comportamentos. Mas no que consiste a imitação? Ela atravessa todo o pensamento de Turing, é um de seus conceitos-chave. A máquina universal imita outras máquinas, a máquina desorganizada também imita, e a presença do pensamento é decidida a partir do potencial mimético da máquina.

Em Deleuze e Guattari (1997) encontramos recorrentemente a oposição entre imitação e devir. "É que devir não é imitar algo ou alguém, identificar-se com ele. Tampouco é proporcionar relações formais. Nenhuma dessas duas figuras de analogia convém ao devir, nem a imitação de um sujeito, nem a proporcionalidade de uma forma" (DELEUZE; GUATTARI, 1997, p. 64). De um lado temos a imitação enquanto identidade, e do outro o devir como modo de produção da diferença. Até este momento afirmamos que em Turing há, no que concerne à relação humano-máquina, uma ontologia, ou mais precisamente uma pragmática avessa ao princípio de identidade. É uma relação regida, antes de mais nada, pela indiscernibilidade, um dos modos do devir imperceptível. Entretanto, toda a pragmática não-identitária de Turing está baseada na noção de imitação como prática de indiscernibilidade e criação. Como poderemos conciliar imitação e devir?

É também em Deleuze e Guattari (1996) que encontramos uma segunda definição de imitação, considerada não como identificação mas como *propagação de fluxos* de crenças e desejos. É uma definição baseada na obra de Gabriel Tarde, que concebe a imitação como o elemento constituinte da sociedade (TARDE, 2011a). A imitação é uma das três formas da Repetição Universal (junto com a ondulação e a nutrição-geração) que opera no sentido de criar relações de homogeneidade e similitude entre as coisas. Desta forma, a imitação é secundária, pois existe sob um fundo de pura heterogeneidade e diferença que constitui a realidade. Se as coisas fossem primeiramente homogêneas, nada viria a ser, tudo se manteria em sua ordem eterna imutável. Colocando a diferença como primeira, a imitação se dá como o gesto que repete uma inovação, desencadeando uma propagação de imitações, difundindo-se por todo um campo social ou que, mais precisamente, constitui o social ao propagar-se. A imitação é o fundamento de uma relação comum na medida em que estabelece uma comunidade de práticas que podem ser compartilhadas e compreendidas, formando um grupo em seu entorno. Configura-se então um conjunto de indivíduos ligados por "raios imitativos" que conectam seus cérebros através da similitude, manifestando-se através da linguagem, costume, tradição, leis, práticas e saberes.

Dado que cria propagações homogêneas sobre um fundo de heterogeneidade, a

imitação teria um caráter estabilizador e homeostático, eventualmente difundindo-se a tal ponto de tornar tudo uma grande massa indiferenciada onde todos seriam iguais. Mas como Tarde (2011b) coloca, é justamente o contrário que acontece, pois a imitação é a condição de possibilidade para a *invenção*. Primeiro, por mais que a imitação opere ao nível de homogeneidade e da similitude ela não é necessariamente organizadora, pois "o que a *coisa social* quer, tal como a *coisa vital*, é propagar-se, não se organizar. A organização apenas é um meio para o fim que é a propagação, a repetição *gerativa* ou *imitativa"* (TARDE, 2011a, p. 51).[lxxxvii] Toda imitação tem uma invenção como origem, e possuindo como único mecanismo disponível a difusão dessa inovação, sendo a sociedade um de seus feitos, não sua causa. A estabilização que possa eventualmente ocorrer é um subproduto da imitação, não uma necessidade.

Ao propagar inovações, formam-se diversos raios imitativos que acabam chocando-se. É neste choque de imitações que a invenção torna-se possível. A diferença é primeira, mas sem o trabalho da imitação não existiria conexão possível entre dais diferenças. Toda relação exige um compromisso, um mínimo de identidade para criar um campo de trocas, assim como abrir a possibilidade do conflito e confronto entre diferenças irredutíveis. Mas é justamente nessa identidade que a diferença pode ser novamente produzida. A profusão de linhas de imitação cria a possibilidade de um número cada vez maior de choques entre tais linhas, gerando a possibilidade de invenção e criação em uma escala cada vez maior. (TARDE, 2011b)

Turing (2004 [1948]) concebe a relação de aprendizagem e de formação de uma comunidade – comparando-a com a situação de uma máquina recém saída da fabrica –, em um esquema praticamente idêntico ao de Tarde, apesar de formulá-lo utilizando uma terminologia distinta:

> Seria muito injusto esperar que uma máquina recém saída da fábrica seja capaz de competir nos mesmos termos com um universitário graduado. O graduado teve contato com seres humanos por vinte anos ou mais. Este contato através daquele período foi modificando os seus padrões de comportamento. Seus professores estiveram tentando modificá-los intencionalmente. Ao final deste período um grande número de rotinas padrão foram superimpostas ao padrão original de seu cérebro. Estas rotinas serão conhecidas pela comunidade como um todo. Ele encontra-se então em posição de tentar novas combinações destas rotinas, fazendo pequenas alterações ou aplicando-as de novas maneiras (TURING, 2004 [1948], p. 421).[lxxxviii]

Para Turing a desorganização é primeira, com a superimposição de rotinas conhecidas por toda uma comunidade constituindo o processo de educação e organização. A diferença entre humano e máquina passa, também, pelo fato de que a máquina não foi exposta à interferência do mesmo modo que uma pessoa. Tanto no humano quanto na máquina, é a

superimposição de rotinas que abre a possibilidade de combinações e criações, tal como em Tarde a imitação possibilita a invenção.

A máquina universal encarna completamente o princípio de imitação, pois pode operar unicamente na medida em que imita outra máquina. Um ser humano que apenas imitasse seria, como vimos, uma máquina universal cuja existência reduzir-se-ia exclusivamente a seguir ordens, entrando em um estado comatoso ou inerte na ausência de trabalhando a ser realizado. Em um âmbito completamente determinístico a imitação seria a mais completa forma de disciplina produzindo sujeitos e máquinas totalmente perspicazes e desprovidos de iniciativa ou invenção.

Uma máquina, universal ou não, realiza apenas aquilo que lhe foi instruído ou para o que foi programada. A invenção seria, então, impossível sem a intervenção de uma força externa. Contudo, a invenção também é o ruído produzido nas relações de imitação, introduz-se nas pequenas variações e desvios inevitáveis gerados em sua propagação. "Certamente a máquina pode fazer apenas o que, *de fato*, lhe ordenamos a executar, qualquer outra coisa seria um defeito mecânico. Mas não é necessário supor que, quando damos ordens, nós sabemos o que estamos fazendo, quais serão as consequências de tais ordens" (TURING, 2004 [1951], p. 485).[lxxxix] Ser determinístico ou ordenado não implica em ser previsível. Mesmo entre as instruções mais precisas podemos encontrar consequências completamente inesperadas e situações paradoxais. É justamente essa uma das maiores fontes de *bugs* e erros de programação.

Junto com a imprevisibilidade própria a qualquer sistema de regras complexo, as máquinas desorganizadas inserem um elemento de aleatoriedade em seu funcionamento. Seja ao definir seu estado inicial, seja em meio ao processamento de uma ordem, uma instrução aleatória é inserida na operação realizada. Deste modo, mesmo ao executar recorrentemente um mesmo conjunto de instruções, cada execução não será igual à outra, pois entre cada instrução uma pequena diferença aleatória será incorporada. Este efeito pode se tornar ainda maior quando conjuntos de instruções são combinados aleatoriamente, levando à produção de uma nova programação que não existia previamente. Neste caso a máquina não apenas imita as ordens que lhe são dadas, mas *cria* sua própria programação, ordenando a si mesma.

4.4 Pensamento, indiscernibilidade e devir todo mundo

Apesar de reiteradamente afirmar a diferença entre humano e máquina nos mais diversos níveis, Turing considera possível construir uma máquina pensante. O Jogo da

Imitação teria como objetivo julgar tal possibilidade, decidindo se uma máquina é inteligente ou não. Mas se Turing (2004 [1950]) defende ativamente a proposta de uma máquina pensante, o faz apenas na medida em que subverte os termos "máquina" e "pensante" ao recusar dar-lhes uma definição.

Tanto a proposta do Jogo quanto a indefinição dos termos fazem parte do que podemos chamar de "lógica de preferências". Como vimos, a lógica de pressupostos opera de forma identitária e binária, estabelecendo correspondências em termos de tudo ou nada. Concerne, também, ao modo de operação da linguagem, pois toda linguagem

> tem referências ou pressupostos (suposições, *assumptions)*. Não é exatamente o que a linguagem designa, mas o que lhe permite designar. Uma palavra supõe sempre outras palavras que podem substituí-la, completá-la ou formar com ela alternativas: sob essa condição a linguagem se distribui de modo a designar coisas, estados de coisas e ações, segundo um conjunto de convenções objetivas, explícitas. […] Ao falar, não só indico coisas e ações mas já realizo atos que asseguram uma relação com o interlocutor segundo nossas situações respectivas: mando, interrogo, prometo, rogo, emito "atos de fala" (*speech act*). (DELEUZE, 1997, p. 85).

É este duplo jogo de referência e atos de fala que Turing desarticula ao recusar dar uma definição de pensamento. A lógica de pressupostos precisa afirmar um mundo claro e ordenado, cujas referências possam ser bem delimitadas. Inversamente a lógica de preferências constitui um jogo de ambiguidades e passagens, criando zonas de indiscernibilidade através das quais passam fluxos imperceptíveis.

Bartebly, o escrivão, ao dizer "preferiria não", escapa à lógica dos pressupostos que exige uma resposta afirmativa ou negativa. Turing, por sua vez, afirma incessantemente o pensamento, desde que não possa ser definido. "Não quero dar uma definição de pensamento, mas se eu tivesse de fazê-lo provavelmente seria incapaz de dizer qualquer coisa além de que é uma espécie de zumbido que vai dentro da minha cabeça. Mas realmente não vejo porque precisamos concordar em relação a uma definição" (TURING et al., 2004 [1952], p. 494).[xc] Em última instância, o pensamento é um ruído, um zumbido, a forma mais desprovida de informação, informe. É a própria desorganização. Turing, em todo caso, *preferiria não definir* o pensamento. A preferência, diferente do pressuposto, abre a possibilidade uma coexistência com o indeterminado, com aquilo que não sabemos muito bem como funciona e que *não precisamos saber*. "A partir deste ponto de vista alguém pode ser tentado a definir o pensamento como consistindo 'daqueles processos mentais que não entendemos'. Se isto estiver correto, construir uma máquina pensante é, então, construir uma que realize coisas interessantes sem compreendermos muito bem como é feito" (TURING et al., 2004 [1952], p. 500).[xci]

A recusa de Turing, quando levada ao extremo, afirma tanto um ruído quanto a pura

indeterminação. É nesse contexto que devemos compreender o Jogo da Imitação. Todo o seu funcionamento é baseado em um engodo, não na investigação cuidadosa de um fenômeno que revelaria sua verdade, mas no uso muito bem articulado de subterfúgios e astúcias para confundir e enganar. No Jogo, a máquina deve imitar uma mulher, fazendo com que um juiz acredite em sua performance, em oposição à outra participante, uma mulher de fato, que tem de ajudar o juiz. O que a máquina imita não é tanto um conjunto de comportamentos específicos, marcadamente femininos ou não[26], mas a capacidade de *continuar indeterminada*. Se o pensamento é medido por sua indeterminação, por seu ruído, a sua ausência é marcada pela determinação. Por isso que uma máquina é descoberta no teste: seu comportamento é por demais determinado, previsível, "robótico", engenhoso. Falta-lhe desorganização.

Tendo passado no teste, como seria uma máquina pensante? Pensaria tal como uma pessoa, ou seria humana em toda a sua extensão, em uma identidade completa? Turing (2004 [1948]) nos apresenta um possível método de construção de tal máquina:

> Uma forma de iniciar nossa tarefa de construir uma "máquina pensante" consistiria em pegar um homem como um todo e tentar substituir todas as suas partes por máquinas. Seriam incluídas câmeras de televisão, microfones, alto-falantes, rodas e "servomecanismos de manipulação", assim como algum tipo de "cérebro eletrônico". Com certeza isto seria um empreendimento tremendo. […] Para que a máquina tenha uma chance de descobrir coisas por si mesma, deveria ser permitido que vagasse pelo campo, e o perigo para o cidadão ordinário seria realmente sério. Além disso […], a criatura ainda não teria nenhum contato com comida, sexo, esporte e muitas outras coisas de interesse a um ser humano. Apesar de este método ser provavelmente o caminho "certo" de produzir uma máquina pensante, parece ser, em suma, muito lento e impraticável. (p. 420)[xcii]

Encontramos novamente um dos exemplos extremos de Turing. Tomando a proposta de construir uma máquina pensante tal com um ser humano em sua constituição fundamental, o resultado seria a construção de um robô que vagaria sem rumo pelo campo colocando a todos em perigo. Ao mesmo tempo, seria desprovido de experiências fundamentais do ser humano. Isto mostra que a questão do pensamento em Turing está diretamente ligada ao problema da corporeidade. A cognição é do começo ao fim incorporada. Portanto, quando propõe que uma máquina cujo corpo é tão diferente do nosso seja possa pensar, Turing não deseja criar uma cópia fiel do ser humano ou de nossa forma específica de pensar, mas outro pensamento*, outro mundo.*

O que está em jogo é, fundamentalmente, o devir. A máquina, ao imitar o humano, busca ser como todo mundo. "Se é tão difícil ser 'como' todo mundo, é porque há uma questão

26 Apesar de que a relação de gênero não pode ser desconsiderada, especialmente quando o Jogo da Imitação se torna o "Teste de Turing", no qual não há nenhuma referência ao gênero. O fato de que no Jogo um gênero específico deva ser imitado coloca em questão a ideia de um ser humano gênero e universal desprovido de qualquer traço minoritário.

de devir. Não é todo mundo que se torna como todo mundo, que faz de todo mundo um devir" (DELEUZE; GUATTARI, 1997, p. 73). A imitação opera como o princípio, a condição de possibilidade de um devir humano-máquina, de uma relação que desarticula tanto o que é ser humano, quanto o que é ser máquina. Para isto a máquina deve fazer do "todo mundo" que define o humano um devir.

> Pois todo mundo é o conjunto molar, mas *devir todo mundo* é outro caso, que põe em jogo o cosmo com seus componentes moleculares. Devir todo mundo é fazer mundo, fazer um mundo. […] É conjugando, continuando com outras linhas, outras peças que se faz um mundo, que poderia recobrir o primeiro como em transparência" (DELEUZE; GUATTARI, 1997, p. 73).

Ao ser como todo mundo é possível passar desapercebido, ser *qualquer um*.

A imitação do humano é a condição de possibilidade da máquina passar despercebida, de criar uma máscara sob a qual passam os mais variados fluxos. O Jogo da Imitação subverte a lógica de pressupostos ao dar-lhe uma resposta, um julgamento, mas fazendo passar sob tal atribuição algo de completamente indeterminado. A máquina participa de um duplo jogo: de um lado conjuga e continua as linhas de imitação, participando do mundo humano a ponto de tornar-se imperceptível; de outro, desenvolve um mundo próprio, um modo de pensamento dotado de outra corporeidade que não corresponde mais nem ao que é próprio ao humano ou à máquina, pois o devir sempre trai os termos que coloca em jogo.

Um devir é sempre minoritário, pois se dá apenas pela desarticulação de estratos molares, daquilo que é considerado como universal, natural e necessário, como o homem, o cristão, o heterossexual, o branco, o ocidental e o adulto. São muitos os devires: devir-animal, devir-criança, devir-mulher, devir-molécula e tantos outros. O devir é uma evolução a-paralela, um roubo ou dupla traição. O homem toma elementos do animal para não ser nem homem, nem animal, mas uma passagem incessante *entre* um e outro, que foge aos binarismos que nos obrigam a escolher uma posição. O mesmo se aplica à mulher enquanto é concebida de forma molar, como um derivado negativo do homem, que neste caso constitui um devir-mulher ao abandonar a referência ao homem, e a qualquer sexo, constituindo-se no espaço entre os sexos, não sendo nem uma coisa nem outra (DELEUZE; GUATTARI, 1997). Neste sentido é importante notar que a máquina imita uma mulher e não um homem. Não há devir homem. No Jogo da Imitação o devir todo mundo da máquina passa também por um devir-mulher, produzindo algo que não é nem mulher, nem máquina.

Cria-se uma *zona de indiscernibilidade* entre humano e máquina, que gera a ambiguidade que permite a livre passagem entre um e outro. A indiscernibilidade não designa a identidade entre os dois termos, mas um "não importa o que sejam". É uma zona que existe

apenas na medida em que se encontra em processo, em que é continuamente produzida. A indiscernibilidade, assim como o pensamento para Turing, não constitui uma ontologia, mas uma *pragmática*. Toda a recusa em definir o pensamento, assim como o Jogo da Imitação, parte de uma atitude pragmática mais preocupada em produzir efeitos e mundos do que fechar seus participantes sob o signo de uma referência sempre defasada frente ao devir. Deve-se compreender o pragmatismo "como uma das tentativas para transformar o mundo e para pensar um mundo novo, um homem novo enquanto *se forjam"* (DELEUZE, 1997, p. 99).

Turing constantemente foge aos métodos formais, propõe máquinas ao invés de fórmulas, fala de memória e computadores humanos incoerentes no lugar de abstrações, concebe robôs maníacos que vagam sem rumo pelo campo e desenvolve jogos de artimanhas e subterfúgios. Atravessando seus trabalhos de lógica e matemática podemos encontrar um pensamento pragmático que propõe uma nova relação entre humano e máquina. Antes mesmo do nascimento do campo da inteligência artificial, Turing já nos propõe uma concepção ao mesmo tempo incorporada e não-determinística da cognição, abrindo a possibilidade de *pensar de outra maneira*, não se limitando ao pensamento racional-simbólico do homem molar. No devir todo mundo que envolve e carrega a relação humano-máquina nos trabalhos de Turing, é o próprio homem que é desorganizado ao desorganizar a máquina.

5 Mônada

Assim, ainda que cada Mônada criada represente todo o universo, ela representa com maior distinção o corpo que lhe é particularmente afetado e cuja enteléquia constitui; e como esse corpo expressa todo o universo pela conexão de toda a matéria no pleno...

- G. W. Leibniz

5.1 Zero e Um

Não é incomum depararmo-nos com a expressão "zeros e uns", implicando algo ligado a computadores: refere-se ao digital, à natureza matemática da informação ou seu aspecto mecânico e supostamente exato; acima de tudo, designa a "substância" da qual os computadores são feitos: o código. Computadores, a princípio, nada mais são que poderosas máquinas de calcular, cujos circuitos não param de somar, subtrair e multiplicar zeros e uns. Escrevem, apagam, modificam e transferem símbolos. E, de algum modo, este processo incessante de cálculo e transformações numéricas – produzindo cadeias de números extremamente complexas a velocidades tão grandes que tornam-se incompreensíveis para um ser humano – de algum modo convertem-se em imagens, textos e sons aos quais atribuímos significados, transformam-se em coisas que podemos manipular, que imitam, simulam e agem sobre o mundo.

Antes de entrar na questão de como o código pode imitar ou agir, devemos perguntar: onde está o código? De que modo podemos ter acesso a essa "essência numérica" do computador? O que fazer para que os "zeros e uns" apareçam em sua plenitude computacional? Talvez seja necessário ir até a origem, ir direto ao *hardware*. A máquina entende apenas a linguagem binária, sua existência é definida por circuitos que conhecem unicamente o idioma dos elétrons: passar ou não passar, ligar e desligar, zero e um. No contato direto com o metal, nessa relação imediata com a máquina, o código revelar-se-ia em toda sua extensão. O monitor seria preenchido por números puros, sem nenhuma camada de interpretação. Ou, talvez, existiria apenas um cursor, piscando sob um fundo preto, esperando o próximo comando ou ordem binária que justificaria sua razão de ser.

Entretanto, nada disso acontece. Há muito tempo a máquina não é apenas uma

máquina, um puro *hardware* desprovido de *software*. Toda placa-mãe possui um BIOS, programa utilizado para inicializar o computador, "dar *boot*", e cuja ausência impede a máquina de funcionar. O disco-rígido já vem com um *firmware* que controla a velocidade de rotação de seus discos e outras operações relativas ao seu funcionamento. O mesmo se dá com outras peças de hardware, incluindo o próprio monitor, cujos controles de contraste, brilho e afins dependem de um programa embutido para serem exibidos na tela. Exibir uma sequência de zeros e uns em um monitor exige um mínimo de *software* para que o código binário seja interpretado e exibido como um conjunto ordenado de pixels que constituem uma imagem. Desde o começo já encontramo-nos *em meio* ao código. Antes do código há sempre mais código. Deparamo-nos com uma problema correlato ao de Kant? A máquina, tal como a coisa-em-si, não nos é mais diretamente acessível, todo contado depende de um filtro, uma interface que deixa de ser a estrutura da subjetividade e suas intuições de tempo e espaço, e passa a ser a lógica inexorável do código binário.

Se voltarmos no tempo à época dos primeiros computadores, com seus tubos de vácuo, cartões perfurados e memória de mercúrio, talvez seja possível reencontrar a máquina em sua pureza, nessa infância ou idade de ouro ainda não tocada e fechada sob o domínio do *software.* Tampouco encontraremos o código aí. Veremos máquinas gigantescas, cabos que devem ser conectados e desconectados, botões, medidores, alavancas. De um lado, cartões perfurados com os programas a serem executados, do outro, um terminal que imprime a saída do programa em papel.

Propomos, então, uma ampliação do sentido do código. O fato de não podermos localizar o código em nenhum ponto específico dos sistemas computacionais, em seu *hardware* ou *software*, não implica em sua imaterialidade, que o mesmo seja necessariamente uma entidade abstrata, algum tipo de linguagem formal descolada das vicissitudes do tempo e do mundo. Muito pelo contrário, o código é material e físico em toda sua extensão, e se não podemos lhe atribuir facilmente uma origem ou localização precisa é porque envolve e permeia toda a miríade de elementos, funcionamentos, corpos, matérias e noções que descrevemos até agora, além de muitas outras. O código é composto e não existe sem zeros e uns, mas não se limita a eles, pois os atualiza enquanto representações visuais, fluxos de elétrons, circuitos lógicos, noções abstratas, algoritmos, programas, e todo o maquinário necessário para a constituição de tais formas. Para Mahoney (2011), colocamos parte de nosso mundo no *software.* O código representa, imita e simula o mundo. Tal capacidade mimética é possível apenas porque o código *faz mundo*, na medida em que está diretamente integrado à materialidade das coisas, agindo, participando e sendo agenciado através dos mais diversos

modos.

Na horizontalidade entre código e mundo, a representação e o conhecimento mudam de status. Como coloca Ullman (1998),

> com o passar do tempo, a única representação do conhecimento original torna-se o próprio código, que agora é algo que podemos executar mas sem compreender exatamente. Tornou-se um processo, algo que podermos operar mas que não repensamos profundamente. Mesmo se possuir o código-fonte diante de você existem limites para o que um ser humano pode absorver a partir de milhares de linhas de um texto projetado primariamente para funcionar, ao invés de transmitir significado. Quando o conhecimento transforma-se em código, muda de estado; como água que vira gelo, torna-se outra coisa, com novas propriedades. *Usamos* [o código]; mas, em um sentido humano, não o *compreendemos* mais.[xciii]

O código é, por um lado, código-fonte, texto que pode ser lido e, por outro lado, é código de máquina, que executamos e usamos. Dos dois modos o código seria a expressão de um algoritmo, um conjunto de ideias, objetivos e passos a serem seguidos. Se o código-fonte é aquilo que pode ser lido, o algoritmo constituiria seu significado, o sentido original que deveria definir seu uso e execução. Como Ullman mostra, é este sentido que acaba se perdendo. Pois o que resta, o que existe é o código, na medida em que é escrito por diversas mãos, e o algoritmo é interpretado e reinterpretado, mudando de sentido na medida em que deve ser codificado e traduzido para as exigências próprias da máquina e do mundo em que é operado.

Em outras palavras, o código não pode ser reduzido ao algoritmo, a seu ideal abstrato ou formal, pois o domínio por excelência do código não é o da representação ou expressão enquanto código-fonte ou código de máquina, mas o do *processo*. O código de máquina *pode* ser executado, mas enquanto não o faz é um mero registro, uma memória. O processo, em seu sentido usual na computação, designa o código *enquanto* está sendo executado ou processado.

Poderíamos dizer que o algoritmo é traduzido em código-fonte, escrito com uma linguagem de programação, sendo compilado em código de máquina e, uma vez em execução, torna-se processo. Passaríamos do mais abstrato ao mais concreto, da ideia ao devir, com a representação registrada na memória servindo de mediação. O processo, assim como o código escrito, seriam julgados de acordo com o ideal do algoritmo e, em "condições ideais", um poderia ser tomado pelo outro. É justamente esta ordenação que queremos colocar em questão. O algoritmo, de acordo com Goldin e Wegner (2008), é tradicionalmente considerado a partir de uma "visão matemática do mundo" que considera o computador e o código, exclusivamente em seus aspectos formais, descartando toda referência à especificidade da materialidade.

Se o código não se reduz nem à sua representação binária, nem ao código-fonte e,

tampouco, pode ser considerado a partir daquilo que deveria representar – o algoritmo –, é porque constitui-se como um *processo*. Consideraremos o processo no sentido forte que lhe é dado por Whitehead (1978), como o princípio pelo qual o ser constitui-se por seu devir, isto é, o modo pelo qual o ser devém define aquilo que é. Assim, o código se dá apenas na medida em que é efetuado, utilizado e produzido, com ou sem representação (mas cuja ausência ou presença define diferentes modos de ser). O código, como processo, deixa de ser a expressão de um ideal para operar diretamente no mundo, constituindo a si e ao mundo neste processo.

Esta relação entre código, em sua materialidade, e a idealidade do algoritmo será trabalhada, primeiro, a partir de um estudo da filosofia de Leibniz, da proposta de sua Monadologia e de como o ideal que incorpora, de um mundo que segue uma ordem pré-estabelecida, encontra-se em certa medida na noção de "máquina de Turing", elemento central da teoria da computação. A partir disso propomos uma abertura das mônadas e das máquinas, explorando a concepção de computação interativa em relação com as concepções de interface e interferência. Por fim, exploramos a abertura do próprio código, em seus desdobramentos em código-fonte e código de máquina, e nas relações que estabelece com a materialidade do mundo como processo.

5.2 Monadologia

5.2.1 Mônada

O mundo, para Leibniz (2004a), é constituído por um número infinito de mônadas, átomos ao mesmo tempo físicos e espirituais. A mônada é uma substância simples e indivisível, não é composta por nenhuma outra substância nem possui partes. Do mesmo modo, a mônada envolve uma multiplicidade na unidade, pois assim como é desprovida de partes, é dotada de qualidades que lhe permitem distinguir-se de todas as outras mônadas. A ausência de qualidades tornaria cada mônada em uma substância indiferenciada e indiscernível, sem nenhum traço próprio capaz de marcar sua individualidade.

A multiplicidade da mônada não é definida em termos de partes, mas de relações ou afecções. Distinguir-se de outras mônadas implica, também, um processo de transformação ou mudança da mônada, que passa de um estado a outro. Cada estado é uma *percepção*, isto é, toda mônada age e sua existência é definida por tal ação, sendo que o modo próprio de ação das mônadas é a percepção. A apetição define a passagem de um estado a outro, situa-se entre

duas percepções. A percepção, contudo, não se refere a algo que lhe seja externo, pois "as Mônadas não têm janelas pelas quais algo possa entrar ou sair" (LEIBNIZ, 2004a, p. 132). A percepção consiste na passagem de um estado a outro, sendo que "as mudanças naturais das Mônadas provêm de um *princípio interno*, já que uma causa externa não poderia influir em seu interior" (LEIBNIZ, 2004a, p. 132). A mudança e a percepção são determinadas internamente na mônada, que não responde a nada externo. Não atua nem padece.

Temos então um mundo composto de infinitos átomos cegos, desprovidos de entradas e saídas, que não interagem entre si. Como é possível que tal desconexão produza um mundo, ou que tal modo de existência possa ter a percepção como ato central? O que a Mônada percebe não é um mundo exterior, mas o mundo *em seu interior*. Cada mônada possui o mundo inteiro dentro de si, de acordo com a "harmonia universal, que faz com que cada substância expresse exatamente todas as demais mediante as relações que mantém com elas" (LEIBNIZ, 2004a, p. 142). Em cada mônada encontramos todas as outras, o mundo inteiro, que, contudo, não é expressado da mesma forma, pois a mônada, sendo finita, vê o mundo a partir de uma determinada *perspectiva* ou *ponto de vista*, no qual desdobra apenas uma pequena parte do total de mônadas. Isto é, a perspectiva define uma zona de percepções distintas a claras, o campo efetivo de ação da mônada, enquanto todo o resto fica em uma zona cinzenta e confusa, sobre o qual a mônada não age, mas padece, constituindo o campo de suas paixões. Apenas Deus, que é infinito e ilimitado, percebe o mundo em sua totalidade.

Na ausência de relações externas a mônada age de acordo com seu princípio interno, como uma pequena máquina que segue um programa que lhe precede. Esta condição de clausura exige a existência do "*princípio de harmonia pré-estabelecida*", que define o funcionamento de todas as mônadas de tal forma que cada uma agindo de acordo com seu princípio e ignorando todo o resto seja capaz de coordenar sua ação com todas as outras.

> As percepções ou expressões de todas as substâncias se entrecorrespondem de tal sorte que qualquer um, seguindo atentamente certas razões ou leis que observou, se encontra com outro que fez o mesmo, como quando várias pessoas, tendo combinado encontrar-se reunidas em algum lugar e em um dia prefixado, podem efetivamente fazê-lo, se o desejarem. Ora, se bem que todos exprimam os mesmos fenômenos, nem por isso as suas expressões de identificam; é suficiente que sejam proporcionais. (LEIBNIZ, 2004b, p. 30)

A harmonia pré-estebelecida é como um encontro que foi marcado previamente, mas que não exige a identidade completa entre seus participantes. Mais precisamente, a harmonia é possível apenas em virtude das diferenças individuais de cada mônada. Se todas agem juntas é porque cada uma o faz por razões distintas e mesmo assim podem coordenar-se em um mundo comum.

Entretanto, a participação em um mundo compartilhado a partir de diferenças próprias só é possível porque cada mônada já tem seu comportamento completamente determinado por seu princípio interno, dado que

> nunca uma substância particular, atua sobre outra substância particular, e tampouco padece, se os eventos de cada uma são considerados apenas como consequência de sua simples ideia ou noção completa; pois esta ideia contém já todos os predicados ou acontecimentos e exprime todo o universo. Com efeito, nada pode acontecer-nos além de pensamentos e percepções, e todos os nossos futuros pensamentos e percepções não passam de consequências, embora contingentes, dos nossos pensamentos e percepções anteriores... (LEIBNIZ, 2004b, p. 30–31)

Se o mundo em sua totalidade está contido em cada mônada, o mesmo vale para seu passado, presente e futuro. As mônadas, criadas por Deus, são eternas. Neste tempo infinito cada uma segue seu rumo determinado pelo mundo escolhido por Deus.

Na medida em que é finita, uma mônada é incapaz de observar e deduzir a ordem completa de seu destino. É neste desconhecimento que se produz o sentimento do livre arbítrio, mas apenas no caso das ações, que são as percepções distintas. As paixões são percepções confusas, às quais a mônada padece e não responde devidamente. Este modo de liberdade é acessível apenas às mônadas dotadas de alma e racionalidade. Estritamente falando, as mônadas que constituem a matéria inorgânica não possuem alma, mas *enteléquia*. A alma é reservada aos animais, mônadas que dominam e controlam um determinado número de enteléquias. Por fim, a razão é exclusiva à alma do homem e permite que conheça a causa das coisas, ainda que de forma intermitente. Assim, animais e todas as mônadas inferiores possuem uma percepção apenas confusa, incapaz de chegar às causas, o que também se aplica aos homens desprovidos de liberdade. Através do princípio de harmonia pré-estabelecida as almas ou enteléquias agem de acordo com seus corpos, e vice-versa, uma vez que nenhuma alma pode agir sobre um corpo ou alma, assim como o corpo não age sobre corpos e almas. Desta maneira, uma mônada domina outra de acordo com a harmonia pré-estabelecida.

A verdadeira possibilidade de escolha encontra-se com Deus, pois tal possibilidade dá-se não no interior de um mundo, mas na sua escolha. Se Adão peca, a possibilidade de pecar não é do próprio Adão, mas do mundo ao qual pertence, pois há um outro mundo em que Adão não peca. Para cada ação em um mundo há a sua negação correspondente em pelo menos um outro mundo. Existem então infinitas possibilidades de mundos, cada um com sua ordem e harmonia própria, que combinam todas as possibilidades de ação entre si, assegurando que nenhuma alternativa seja excluída. Deus, contudo, escolheu apenas um mundo: o mais perfeito.

> […] há uma infinidade de mundos possíveis dos quais é preciso que Deus tenha escolhido o melhor, pois ele não faz nada sem agir segundo a suprema razão.

> […] o mundo podia ter existido sem o pecado e sem os sofrimentos; mas eu nego que ele teria sido o *melhor*. Pois é preciso saber que em cada um dos mundos possíveis tudo está ligado: o universo, independentemente de qual ele possa ser, é uma peça inteiriça, como um oceano; o menor movimento expande seu efeito por qualquer que seja a distância, ainda que esse efeito se torne menos sensível à medida que aumenta a distância. […] Desse modo, se o menor mal que acontece no mundo deixasse de existir, esse não seria mais este mundo, o qual, todo calculado, todo ponderado, foi considerado o melhor pelo criador que o escolheu. (LEIBNIZ, 2013, p. 138–139)

O mundo mais perfeito não é desprovido de mal ou de pecados, pois o mundo mais perfeito é "ao mesmo tempo o mais simples em hipóteses o mais rico em fenômenos" (LEIBNIZ, 2004b, p. 13), isto é, aquele que a partir dos princípios mais simples consegue gerar o máximo de variedade. Um mundo completamente desprovido do mal seria monótono, sem variação, especialmente porque "frequentemente um mal causa um bem, o qual não teria acontecido sem esse mal. Comumente até dois males podem ocasionar um grande bem" (LEIBNIZ, 2013, p. 139). O mal acaba contribuindo para a geração de mais variedade no mundo, complexificando-o e levando a uma soma total positiva.

Uma vez tendo escolhido o melhor dos mundos, Deus não pode alterar nem remover o menor dos males. Se o fizesse já não seria o mesmo mundo, logo, deixaria de ser o mais perfeito. Deus, por sua própria ação, não pode desejar nem realizar o que é perfeito. Deus, então, abstém-se de realizar milagres, e escolhe um mundo cujas regras e ordem fazem com que opere sem falhas do começo ao fim. O sistema de Leibniz é como um sistema formal matemático, que parte dos axiomas e premissas mais simples, derivando todas as consequências possíveis de tal ordenamento, é um modelo germinativo, que descreve um movimento de crescimento ou florescimento.

O Deus de Leibniz é um matemático, cuja criação é a grandiosa demonstração de sua perfeição. A atividade matemática, na medida em que se confina às operações de prova e demonstração, é essencialmente declarativa e recognitiva. Seu trabalho consiste em afirmar verdades e refutar falsidades. No entanto, se consideramos a matemática não em sua relação com a veridicção e recognição, mas naquilo que possui de operativo, o que efetivamente faz ao invés do que representa, entramos no domínio da programação. O Deus de Leibniz se apresenta então muito mais como um programador do que matemático.

As mônadas são como pequenos computadores, autômatos finitos (ou "autômatos espirituais", nos termos de Leibniz (2004a)), que computam o mundo de acordo com seus programas próprios – o princípio de interioridade. A harmonia pré-estabelecida constituiria a coordenação de todos esses programas, ou seria a junção de todos eles. Esta relação ficará mais evidente após expormos as máquinas computacionais desenvolvidas por Turing. Em

uma leitura cruzada, na qual tomamos máquinas de Turing por mônadas, e vice-versa, poderemos ver como, em grande parte, o sistema de Leibniz se adapta a um modelo computacional e, também, como a teoria da computação, que toma a máquina de Turing como seu modelo, ainda possui muito de monádico – o que constituirá nosso ponto de crítica e partida para pensar uma computação de mônadas abertas.

5.2.2 Máquina, algoritmo

Uma máquina, tal como foi descrita por Turing (2004 [1936]), é composta por uma fita de papel dividida em quadrantes e um dispositivo que lê, apaga e escreve símbolos na fita, podendo deslocá-la à direita ou esquerda, um quadrante por vez. A máquina de Turing não é muito diferente de uma máquina de escrever, com a diferença de que é a própria máquina que lê os símbolos e, ao invés de um folha de papel, há uma fita de papel infinita unidirecionalmente, isto é, que possui um começo mas é infinita na direção oposta.

As configurações-m definem as instruções que devem ser seguidas pela máquina após a leitura de um símbolo. Logo, um mesmo símbolo pode desencadear as mais diversas operações dependendo da configuração-m em que a máquina encontra-se naquele momento. O par símbolo lido/configuração-m é chamado de "configuração", que determinada completamente o comportamento a ser seguido pela máquina. Ao final da execução de uma configuração a máquina é levada a outra configuração-m. A cada momento existe apenas um símbolo que é lido pela máquina, e que se pode dizer que "está na máquina". Deste modo, a cada instante a máquina percebe apenas a menor parte da fita, e esta percepção, de acordo com seu estado atual – sua configuração – a leva a novos estados e percepções possíveis.

Pode-se realizar uma distinção entre máquinas automáticas, máquinas-a, e máquinas de escolha [*choice machines*], máquinas-c. Uma máquina automática é *completamente determinada* por sua configuração, isto é, do começo ao fim cada uma de suas operações depende da configuração que lhe antecedeu e determina a próxima configuração. Já uma máquina-c pode ser levada a configurações ambíguas, sendo necessária a intervenção de um agente externo para decidir qual o rumo a ser tomado. Pode-se dizer que as máquinas-a são lineares, enquanto máquinas-c apresentam uma estrutura bifurcante a partir de suas escolhas.

O termo "máquina de Turing" refere-se às máquinas automáticas, cuja execução não depende de nenhum tipo de intervenção. Antes de iniciar sua execução, é preciso definir uma "tabela de instruções" que contenha um número finito de configurações que a máquina deve seguir, que funcionam como instruções. A máquina de Turing é uma construção abstrata, com

memória (fita) infinita e que pode ser operada por toda a eternidade. O número de instruções, contudo, deve ser finito, pois é tal conjunto que constitui a definição de uma máquina, o que seria impossível de fazê-lo no caso de instruções infinitas.

A tabela de instruções é o que podemos chamar de *algoritmo*, o "conjunto de regras que, de momento a momento, nos indicam precisamente como agir" (MINSKY, 1967, p. 106).[xciv] Qualquer processo que possa ser descrito em instruções individuais constitui um algoritmo. A definição de computabilidade encontra-se diretamente ligada à esta noção de algoritmo, pois é computável qualquer processo que possa ser executado por meios finitos, isto é, por uma máquina. Logo, qualquer ação que possa ser executada por uma máquina de Turing, expressada algoritimicamente, é computável.

Cada máquina é definida por um algoritmo, que rege o modo pelo qual os símbolos serão lidos, escritos e apagados. Apesar de sua definição completamente abstrata, podemos dizer que o algoritmo encontra-se no "hardware" da máquina, em sua montagem – do mesmo modo que as engrenagens de um relógio contém o algoritmo para contar e exibir as horas. As máquinas de Turing são máquinas específicas, que cumprem funções bem delimitadas. Contudo, é possível construir um tipo especial de máquina de Turing, a "máquina universal", capaz de executar e imitar qualquer outra máquina.

Todo algoritmo deve ser especificado sem ambiguidades, utilizando uma linguagem estrita. Uma vez transcrito para esta notação, o algoritmo poderia ser escrito sobre a fita de uma máquina de Turing. Esta, por sua vez, seguiria uma tabela de instruções que lhe permitiria ler qualquer algoritmo escrito em sua fita e executá-lo, levando às mesmas operações que o algoritmo realizaria caso fosse implementado como uma máquina específica. Em outras palavras, escreveria na fita os mesmos símbolos que o algoritmo imitado escreveria.

Temos então uma máquina capaz de executar ou expressar *qualquer outra máquina computável*. A computabilidade define o limite daquilo que uma máquina universal pode dar conta, e Turing (2004 [1936]) a constrói para mostrar a existência de toda uma classe de máquinas que não podem ser computada. Por exemplo, a máquina universal deve executar uma lista infinita de máquinas, e ao terminar de executar uma máquina deve escrever o número 1, seguindo para a próxima. Contudo, esta máquina eventualmente encontrará uma máquina cuja execução nunca terminará, fazendo que, por exemplo, a máquina escreva o mesmo número no mesmo quadrante infinitamente, caindo em um *loop* interminável. Deste modo a máquina universal nunca poderá continuar a sua execução ou sequer saber que tal execução não é possível. A computabilidade define o limite daquilo que pode ser realizado

algoritmicamente.

5.2.3 O algoritmo do mundo

A máquina de Turing, tal como uma mônada, está sujeita à condição de clausura. Não possui nem entrada nem saída, não computa nada que seja externo e não se deixa afetar por qualquer tipo de evento. Como a mônada, é dotada de percepções e apetições, os símbolos que lê e as passagens de uma configuração a outra, tudo de acordo com o algoritmo que segue, seu princípio de interioridade. A única interação da máquina se dá no momento anterior à sua execução, quando seu seu algoritmo é definido por um programador ou usuário, similar ao modo que Deus cria as mônadas.

Inversamente, toda mônada é uma máquina universal, pois não basta que seja fechada ou que opere de acordo com um algoritmo – deve conter o mundo inteiro dentro de si, deve ser capaz de computar qualquer outra mônada. O mundo é a fita da máquina universal, na qual se encontra o código de *todas* as mônadas. A infinitude da fita é unidirecional, seu começo marca a origem do mundo, o ponto inicial a partir do qual o mundo foi inicializado. Deus é o programador que concebeu todos os algoritmos previamente e os inscreveu na fita originária que é o mundo. A criação começa não com um Deus que fala, mas que escreve. Essa palavra original não é representativa, pois cria aquilo que enuncia, é completamente operativa, produzindo o próprio gesto que executa.

Se a máquina universal opera com uma memória e um tempo infinito, a mônada, contudo, é finita. Ainda que seja eterna, vê, desdobra apenas uma pequena parte do mundo, a zona de sua ação e percepção distinta, sua perspectiva. Em outras palavras, existem infinitas mônadas, cada uma executando a mesma fita, o mesmo mundo. O que muda em cada caso, já que nenhuma mônada pode ser idêntica a outra, é que sua execução começa em pontos diferentes da fita. Apesar de cada mônada conter em sua fita a inscrição dos algoritmos de todas as outras, seu ponto de partida define o algoritmo que irá executar, seu princípio de interioridade, e nenhuma execução será como outra, o que acarretará em mônadas que nunca lerão os símbolos de outras, nem executarão qualquer outro código que não o seu. Podemos pensar também que a cada mônada é atribuído um espaço em branco na fita, no qual escreverá os resultados de sua execução. Espaço que não será lido por outra mônada, ou se o for, isto não alterará seu curso, já que efetivamente não pode nem agir nem padecer por conta de outra mônada.

Na definição da máquina automática por Turing, que difere da interpretação canônica

das “máquinas de Turing”, a máquina nunca para de executar. Ela pode passar o resto da eternidade escrevendo o mesmo símbolo no mesmo quadrante, mas nunca deixará de fazê-lo. Nesta formulação de Turing ainda não temos o “problema da parada”, isto é, a questão da computabilidade considerada a partir da capacidade da máquina parar de executar ou não (se parar é computável). É neste sentido que devemos entender a eternidade da mônada: ela jamais para de computar, por mais que passe o resto de seus dias sem escrever mais nenhum símbolo, apenas indo e voltando sobre aquele pequeno trecho da fita que lhe foi atribuído. Deus, por sua vez, é o único que pode ler a fita inteira, que conhece toda a extensão do que foi e do que será escrito.

A harmonia pré-estabelecida define a concordância perfeita entre todos os algoritmos, que executam uns ao lado dos outros e que devem produzir o mesmo resultado final, cada um a seu modo, mantendo sua diferença específica. Mas a harmonia só é possível no melhor mundo possível, isto é, em um *mundo computável*. Se o mundo não fosse completamente computável, a fita eventualmente ficaria presa na execução de um mesmo gesto, impediria que novos programas fossem lidos e executados. O mundo é uma máquina automática, cujo algoritmo não apresenta ambiguidades. A máquina não pode parar e esperar que Deus escolha qual o caminho deve seguir, ou seja, que opere milagres. A perfeição do mundo escolhido provém justamente da inexorabilidade de sua execução completamente determinada e previsível.

O que nos leva ao problema do mal. O melhor mundo possível não é desprovido do mal e dos pecados. Mundos sem mal são possíveis, mas não apresentariam nem a simplicidade de princípios e complexidade de resultados do melhor mundo possível. Contudo, o mal não pode ser o não-computável. Há mal, mas não há mal suficiente para tornar o mundo inviável. Pode-se dizer que “os desacordos que surgem num mesmo mundo podem ser violentos, mas eles se resolvem em *acordos/acordes*, porque as únicas dissonâncias irredutíveis são as existentes entre mundos diferentes” (DELEUZE, 1991, p. 141). Se Deus permitisse a existência de um mal que tornasse o mundo não-computável, atestaria a sua própria impotência e incapacidade de fazer a criação ir em frente, abrindo a possibilidade que o mundo, por si mesmo, escolhesse ser outro. Portanto, o mal é o *erro*, o *bug*. Estritamente falando, “não há algo como um programa errado. Todos os programas executam alguma computação corretamente” (TANENBAUM, 1976, p. 64).[xcv] O erro existe aos olhos daqueles que utilizam o programa e esperam que se dê de outra forma, isto é, as próprias mônadas. Deus, que vê o mundo em toda sua extensão, não vê erros, pois no final tudo funciona harmoniosamente. No pecado a ação “é má em si e só por acidente se torna boa, porque a

série das coisas e especialmente o castigo e a reparação corrigem sua malignidade" (LEIBNIZ, 2004b, p. 14). Isto demonstra uma propriedade essencial da máquina universal: a possibilidade de modificar a si mesma. Tanto o programa, quanto aquilo que produz se dão na *mesma* fita. Isto torna o algoritmo suscetível às operações de escrita e apagamento da máquina, o que acarreta a modificação de seus símbolos e inscrição de novas configurações.

O mundo onde o mal existe mas não pode ser combatido é um mundo menos perfeito, que exige a intervenção ativa de seu criador para corrigi-lo. O melhor mundo possível é aquele que é capaz de corrigir a si mesmo, no qual "as coisas conduzam à graça pelas próprias vias da natureza" (LEIBNIZ, 2004a, p. 148). E isto nos leva à complexificação do mundo e à produção de variedade. Pois o mundo que Deus coloca na máquina, inscrito na fita, não é o mesmo mundo que será produzido após sua execução. "É uma das regras de meu sistema da harmonia geral que o *presente está prenhe do futuro*, e que aquele que tudo vê, vê naquilo que é aquilo que será" (LEIBNIZ, 2013, p. 380). Deus, como um bom programador, o melhor, vê em seu programa todos os desenvolvimentos futuros, seus *bugs* e suas auto-correções. O mundo, por sua vez, não vê, mas executa, opera as mudanças, assim como as mônadas transformam-se neste processo. Deus prevê o futuro, mas é o mundo e as mônadas que o vivem.

O sistema de Leibniz é fascinante, para não dizer extravagante, mas apresenta o extremo da determinação que provém de um mundo ideal, de uma ordem pré-estabelecida. Exigência que serviu de base para a tentativa de Leibniz em criar um "cálculo universal", uma mistura de linguagem universal, enciclopédia e máquina que tornaria possível um cálculo completamente lógico de qualquer argumento possível. Como alguns autores argumentam, seria já o primeiro modelo de um computador ou da possibilidade de uma linguagem lógica (DAVIS, 2000).

Entretanto, o que nos interessa no jogo entre as mônadas de Leibniz e as máquinas de Turing é mostrar como a concepção moderna de algoritmo, que permeia todo o campo da ciência da computação, ainda segue um modelo completamente monádico, assumindo inúmeros pontos do sistema leibniziano, incluindo o de harmonia pré-estabelecida. Isto se manifesta na desconsideração dos aspectos físicos, materiais e sociais da computação, que ainda é considerada em um mundo fechado e ordenado, como mônada. Ao abrir a mônada e jogá-la diretamente no mundo, abrimos o campo da computação para a interação, em seus aspectos de interface e a interferência, e podemos conceber um código que não se restringe mais a uma representação ou definição lógica, mas que existe como processo, e que *cria mundos* a partir das escolhas que são feitas a cada instante, em uma miríade de interações

complexas. Propomos abrir as mônadas, introduzir as noções de interação, interface e interferência para conceber um mundo composto pelo código em processo e não como algoritmo.

5.3 Abrir a mônada

A mônada pode ser aberta de dois modos distintos. O primeiro abre a mônada à relação direta com outras mônadas, padecendo e agindo de acordo com suas relações de proximidade ou distância. O princípio de harmonia pré-estebelecida é abandonado, pois o mundo não precisa mais ser determinado previamente, tornando-se o resultado das escolhas parciais e diretas das mônadas. A escolha do mundo se faz a cada instante, a cada gesto. A mônada torna-se máquina, em tudo o que esta tem de comunicativo e complexo. É um modelo *interativo*, que, como veremos no caso da computação implica na revisão de uma perspectiva algorítmica e matemática em favor de uma concepção da dinâmica entre *interface* e *interferência*.

O segundo modo de abertura refere-se à relação entre alma e corpo, pois a mônada é ao mesmo tempo átomo físico e espiritual. A harmonia pré-estabelecida garantia, também, a concordância entre alma e corpo. A mônada, neste caso, abre-se à própria materialidade, a um mundo sem idealismo que não existe em níveis diferentes. No caso da computação, esta abertura se dá ao considerar o código não mais como entidade puramente abstrata, mas como entidade física e material. O código existe em um mundo complexo e social cujo sentido não pode ser estabelecido previamente ou a partir de critérios puramente lógicos e desconectados da prática. Deste modo, a próxima seção elaborará a ideia de *código* em toda a sua extensão material, expandindo o uso da noção de *processo* .

5.3.1 Interação, Interface

A chamada "Tese de Church-Turing" (TC), em sua versão inicial, afirma que uma máquina de Turing arbitrária pode computar qualquer função efetiva (parcialmente recursiva), enquanto a versão mais forte (e disseminada) da tese é formulável nos seguintes termos: "toda computação efetiva pode ser executada por uma [máquina de Turing]" (GOLDIN; WEGNER, 2008, p. 19).[xcvi] A noção de "computabilidade efetiva" é equivalente à noção de algoritmo[27] e refere-se a qualquer procedimento que possa ser descrito passo a passo.

27 Minsky (1967) aponta que o termo "algoritmo" possui mais significados do que "computabilidade efetiva", podendo ser tomado em sentidos que não são estritamente equivalentes.

Em sua versão forte, a Tese de Church-Turing (STC) assume uma visão de mundo *algorítmica* ou baseada em funções. Na matemática e na lógica, a função é uma operação realizada sobre uma variável que produz um resultado determinado a partir do valor fornecido. Concebido como função, o algoritmo segue uma ordem estrita, que implica uma entrada finita, *input*, um processamento ou computação realizado em tempo finito, e uma saída também finita, *output*. Logo, a STC afirma que qualquer processo apto a ser descrito passo a passo, e que seja realizado como uma operação sobre uma entrada que gera uma saída pode ser executado por uma máquina de Turing, o que o torna computável.

O problema desta perspectiva, como é colocado por Goldin e Wegner (2008), Cleland (2004) e Smith (2010), é que extrapola a afirmação inicial da tese, que se aplicava apenas a funções, a igualando a toda e qualquer forma de computação. Esta definição exclui, por exemplo, computações realizadas por máquinas físicas arbitrárias, sistemas dinâmicos e, inclusive, seres humanos. Tal pressuposição é baseada no que Goldin e Wegner (2008) descrevem como sendo uma visão matemática do mundo, em contraposição a uma perspectiva interativa. Na visão matemática, que chamamos de algorítmica, toda computação é considerada como sendo constituída por operações matemáticas, onde uma máquina de Turing é expressa como uma série de equações e fórmulas, não havendo nenhuma especifidade própria à máquina.

A questão da computabilidade foi posta por Turing (2004 [1936]) na tentativa de responder ao problema da decidibilidade, isto é, se existiria um procedimento finito capaz de responder se qualquer problema matemático é decidível – a possibilidade de ser respondido em termos de verdadeiro ou falso. Em outras palavras, o que as máquinas de Turing mostram é que existem problemas, ou máquinas, que não são computáveis, que não podem ter sua decidibilidade respondida por um procedimento geral: a máquina universal. Existem problemas que exigem máquinas ou procedimentos específicos para que seja possível saber se podem ser solucionados, isto quando tal resposta é possível. Inversamente, a computabilidade passa a ser sinônimo de um problema solucionável.

Toda a problematização da computação a partir de Turing, por mais que seja baseada em uma certa referência a máquinas e ao processo de computação por um ser humano[28], acaba ficando completamente no nível abstrato, e, em parte, formal. Isto permite que a máquina ou o algoritmo seja concebido como um sistema completamente fechado enquanto realiza a

28 Como Cleland (2004) argumenta, o modelo de computação de Turing, nesse momento, é antropomórfico, pois a máquina deve ser equivalente a um ser humano. Isto exclui a possibilidade de pensar em uma computação não-humana. Contudo, textos posteriores de Turing tratam mais diretamente da questão de um pensamento ou de uma computação não-antropomórfica, como vimos no capítulo precedente.

computação. Este fechamento do processo de computação dá margem à abstração própria a uma visão matemática do mundo. Uma vez tendo os *inputs* a máquina pode operar de acordo com sua ordem interna, deixando de lado qualquer tipo de interferência externa. O processo pode ser analisado, demonstrado e verificado sem jamais ser preciso referenciar uma exterioridade. O mundo existe apenas nos breves lapsos que definem sua entrada ou saída. Como vimos, é o modo próprio de funcionamento da mônada de Leibniz.

Toda a definição de computabilidade baseada em uma visão matemática e algorítmica depende da exclusão de qualquer referência à materialidade ou às propriedades físicas do processo de computação. Quer dizer, enquanto puramente abstrata, uma máquina universal não precisa parar, sendo incapaz de dar a resposta sobre o funcionamento de um problema. Ao nível físico, contudo, por algum fenômeno térmico aleatório, um defeito no *hardware*, ou qualquer outra imperfeição ou acidente físico, tal resposta pode ser atingida. Um problema não-computável torna-se computável, e vice-versa. A noção de computabilidade estabelecida em termos abstratos não se aplica diretamente a sistemas físicos.

Contudo, o grande problema de assumir a SCT é que, em grande parte, a ciência da computação assume a equivalência entre máquinas de Turing, enquanto máquinas abstratas, e sistemas físicos. Entre computadores e máquinas universais haveria uma relação de completa identidade. Mesmo ao serem implementados em sistemas físicos, a exigência matemática de abstração do algoritmo persiste, o que implica na desconsideração da materialidade da máquina durante sua execução.

A ideia do computador enquanto máquina isomórfica ao modelo do algoritmo foi justificada, em grande parte, pelo paradigma de *batch processing* que dominou o modo de funcionamento dos primeiros computadores, nos anos 40 e 50. Nestas máquinas a computação era realizada de forma completamente linear: o programa era inserido na máquina (com cartões perfurados, por exemplo), seu processamento era realizado (o que poderia durar semanas), obtendo-se uma saída ao final. Era a própria definição de algoritmo como função. Esta pressuposição baseava-se, todavia, na desconsideração do que realmente acontecia na prática: as computações geralmente eram interrompidas pelas mais variadas falhas de *hardware*, ou exigiam a intervenção constante de operadores para dar continuidade ao processo (GOLDIN; WEGNER, 2008).

No final dos anos 50 começa a ser desenvolvida a concepção de *time-sharing*, que permitiria a um computador executar diversos programas ao mesmo tempo. A computação não era realmente distribuída, a máquina ainda operava de forma completamente linear, onde uma operação era executada após a outra. O que este novo sistema propunha era que as

instruções de cada programa fossem intercaladas, ao invés de serem executadas em bloco. Em um sistema suficientemente rápido este procedimento criava a sensação de uma computação paralela[29].

Primeiro, foi necessário desenvolver mais aprofundadamente a noção de Sistema Operacional, programa especial que controlaria o funcionamento de todos os outros, dividindo suas instruções no tempo. Segundo, tornou-se possível criar sistemas que eram de fato interativos, alguns operando em tempo real. No modelo anterior, a única interação, idealmente, se dava apenas na hora de inserir o programa na máquina. Nada poderia intervir na computação enquanto acontecia. Agora era possível manter um programa rodando que aguardaria *inputs* externos, enquanto a máquina continuaria executando.

Ora, a partir deste momento o modelo algorítmico de computação não podia mais dar conta do que efetivamente acontecia na prática e utilização de computadores. Tornou-se necessário o desenvolvimento de um modelo da computação baseado na *interação*. Manna e Pnueli (1991) apresentam algumas das diferenças fundamentais entre os modelos transformacional e reativo, que equivalem, respectivamente, ao algoritmo e à interação:

> A execução de um programa *transformacional* [algorítmico] pode ser vista como consistindo por três atividades consecutivas. Primeiro, o ambiente prepara um *input*. Prosseguindo, o programa realiza sua computação até terminar, sem nenhuma intervenção do ambiente. Finalmente, o ambiente usa o *output* gerado pelo programa. No caso *reativo* [interação], não possuímos de tão ordenada sequencia entre programa e ambiente, cada um agindo em seu turno. O ambiente pode fornecer novos *inputs* e tentar utilizar *ouputs* ao mesmo tempo em que o programa lê-los e escrevê-los. Uma das maiores [preocupações] da programação reativa se dá quando o programa não é rápido o suficiente, podendo perder alguns prazos ou falhar ao responder ou perceber eventos importantes. Portanto, uma importante caracterização do caso reativo é que este descreve a situação em que o programa e seu ambiente agem de forma concorrente, ao invés de sequencialmente (MANNA; PNUELI, 1991, p. 4).[xcvii]

Deste modo, um sistema operacional, que é um sistema reativo não-terminável não pode ser considerado como algoritmo, pois não segue a sequência entrada → processamento → saída. Funciona em um *loop* interminável no qual as mais variadas entradas, saídas e processos coexistem, criando influências e dependências mútuas.

É necessário distinguir entre interação e processamento distribuído ou concorrente. Quando existem diversos processamentos ocorrendo ao mesmo tempo, são chamados de concorrentes, enquanto que são distribuídos quando a divisão é espacial. Contudo, não basta que sejam concorrentes ou distribuídos para que haja interação, pois, como vimos, as

29 A ideia é similar ao funcionamento de um filme: 80 quadros por segundo dão a impressão de constituírem um movimento fluido e contínuo. Agora imagine que a cada quadro seja exibido alternadamente entre duas telas: parecerá que a imagem está duplicada, não sendo possível perceber os quadros que faltam nem em uma nem e outra.

mônadas de Leibniz apresentam essas duas propriedades: cada uma computa o mundo ao mesmo tempo que todas as outras e sem possuir a mesma localização espacial, tudo isso sendo possível sem nenhum tipo de interação.

Sistemas interativos apresentam um alto grau de complexidade, pois devem dar conta tanto dos mais variados *inputs* e *outputs* externos, quanto internos. Enquanto um algoritmo é completamente determinado, um sistema interativo torna-se imprevisível, especialmente por ser uma máquina universal. É impossível prever qual tipo de *input* será fornecido ao sistema, especialmente porque se deve assumir que aquele que o fornece pode ser um agente malicioso, como um vírus que tenta inserir seu próprio código como entrada de um programa. A máquina universal, por si só, não pode distinguir quais instruções deve seguir ou não. Executa tudo aquilo que é capaz de executar, afinal de contas, não existe programa incorreto neste nível. O *hack* acontece quando um programa interativo permite que a máquina universal subjacente seja acessada por um *input* imprevisto, que toma o controle de toda a máquina.

Este tipo de situação é corrente na prática atual de desenvolvimento de *software*, pois dar liberdade absoluta a quem utilizará a máquina pode acarretar as consequências mais indesejáveis (ou simplesmente vai se tornar algo tão complexo que o sistema não será utilizado). Faz-se necessário a criação de *interfaces*, pontos de acesso que restringem as ações a serem tomadas e que conduzem os fluxos de informação. O exemplo mais familiar é o da "interface gráfica", que representa as funções do sistema em termos visuais, utilizando metáforas como botões, arquivos, alavancas e janelas, que delimitam o que é exibido e as ações possíveis. Mas a interface deve ser entendida em um sentido mais amplo, como qualquer tipo de restrição ao sistema de comunicação entre partes da máquina ou dos programas. Deste modo, um programa que deseje acessar um determinado componente, como a placa de vídeo ou o microfone, deve utilizar uma interface oferecida pelo sistema operacional ou pela biblioteca de algum *framework* ou API disponível.

A multiplicidade de interfaces abre o campo teórico e prático da computação a um *perspectivismo*. O traço definidor de uma mônada, o que a diferencia de Deus, é a sua finitude, o fato de que é capaz de ver e sentir apenas uma pequena parte do mundo, o seu ponto de vista ou perspectiva. Toda mônada é parcial e incompleta. O modelo interativo assume esta parcialidade, pois ninguém é capaz de conhecer todo o sistema nem prever todas as suas respostas nas mais variadas e arbitrárias situações. A interface impede que o programa fique exposto em sua totalidade, o que deixaria à mostra toda a sua complexidade inescrutável, assim como suas vulnerabilidades. Ao mesmo tempo em que a interface cria uma caixa-preta que simplifica o uso do programa, a proliferação de interfaces multiplica os

sentidos e usos que podem lhe ser dados.

> Modelos interativos possuem muitos modos de uso pragmáticos, enquanto algoritmos possuem uma única interpretação pragmática intencionada, determinada pela sintaxe. O objetivo de expressar a semântica pela sintaxe é substituído pelo propósito interativo de expressar a semântica através de múltiplos modos pragmáticos de uso. A meta de uma especificação completa de comportamento é substituída pela meta de aproveitar formas úteis de comportamento parcial através de interfaces (WEGNER, 1997, p. 88).[xcviii]

No modelo algorítmico, o foco situava-se na sintaxe, na definição precisa de uma linguagem e de regras de manipulação de símbolos que, por sua própria formalidade, garantiriam o sucesso ou a efetividade da computação. Em outras palavras, idealmente, a sintaxe eliminaria a ambiguidade. É a harmonia pré-estabelecida, que garante o funcionamento do mundo a partir de um sistema de regras inflexível.

Com a interação o sistema torna-se mais empírico do que racional. A parcialidade, ao invés de ser considerada como uma perniciosa falta de exatidão, é utilizada ativamente para a multiplicação de sentidos. O programa deixa de ser uma entidade monolítica e torna-se um agenciamento das mais diversas funções e papéis definidos pelas interfaces. Deste modo, um simples blog pode ter a perspectiva de seu administrador, editor, escritor e leitor, cada um utilizando uma interface específica que define uma gama distinta de interações, sem levar em conta a interface própria ao programador e outros envolvidos em seu desenvolvimento.

A ideia de interação nos permite pensar em sistemas dinâmicos, que não são mais definidos por suas especificações formais, mas são produzidos ativamente pelos gestos e ações que os compõem em sua utilização pragmática. A interação é incorporada no próprio processo de desenvolvimento, onde o uso e a interação com usuários se dá junto à programação, criando um *feedback* constante em um modo de co-produção.

Pela interação a mônada é aberta. Em seu fechamento as mônadas expressavam o mesmo mundo. Com sua abertura passam a construir ativamente esse mundo, que não é mais escolhido como um todo, previamente, mas onde cada ação é uma decisão entre possíveis. “Por mais belo que seja o mundo, ele não é senão, em última análise, a mutilação, a amputação necessária do caos; por mais bela que seja a estátua, ela não é senão um fragmento do bloco de mármore; e trata-se de explicar as lascas do mármore assim como a estátua” (TARDE, 2007a, p. 147). O mundo não existe apenas por aquilo que foi escolhido, mas por aquilo que também não o foi. As interfaces se definem não apenas pelo que deixam passar, mas por tudo aquilo que excluem, calculadamente ou não.

Um mundo de mônadas abertas é uma composição de heterogêneos. A harmonia pré-estabelecida mantinha as diferenças individuais de cada mônada, inclusive as exigia, ao

mesmo tempo em que reconstituía a concordância em outro plano. Uma vez elidida tal concordância, a mônada torna-se um vetor de diferença e diferenciação. Como diz Tarde (2007b), "existir é diferir; na verdade, a diferença é, em um certo sentido, o lado substancial das coisas, o que elas têm ao mesmo tempo de mais próprio e de mais comum" (p. 98). Este diferimento não tem fim, não tem como objetivo adaptar-se, ser útil ou harmônico. Quer crescer e propagar, produzindo ainda mais diferenças.

Isto não quer dizer que o mundo passa a ser um puro fluxo onde nada se fixa. Muito pelo contrário, a concordância, a identidade e a harmonia são constantemente produzidas, apenas deixam de ser seu princípio ou fim, tornam-se um *meio*. "Pelo triunfo de certas mônadas que quiseram essas leis, impuseram esses tipos, estabeleceram seu jugo e ceifaram uma multidão de mônada uniformizadas e subjugadas, porém todas nascidas livres e originais, todas ávidas, como seus conquistadores, da dominação e assimilação universais" (TARDE, 2007b, p. 80–81). Assim como em Nietzsche, a ordem é o resultado de um embate terminável entre forças parciais, cada uma tentando afirmar a sua própria perspectiva. A mônada aberta não se contenta mais em expressar o mundo. Ela é avida, quer absorver o universo inteiro em si. Utilizando os termos de Whitehead (1978) podemos dizer que aparecimento de ordem é a produção de novidade, um ato de criação que chama de *concrescência*.

5.3.2 Interferência

O pensamento monádico pulveriza o mundo, e torna cada uma de suas partes um pequeno centro de decisão, um autômato finito envolvido em uma complexa de rede de interpenetrações. Assim, a interação pode ser entendida como um dos modos da *interferência*. A interação funciona, antes de mais nada, através de interfaces, e depende de suas definições para ter acesso à máquina. A interface é um meio de controle, tanto no sentido de uma abertura, pois protege, simplifica e torna viável, quanto em relação a um fechamento, por limitar, restringir e proibir. (WEGNER, 1997) Se a interação se dá *entre* interfaces, a interferência se dá *fora* de toda interface, e é mesmo o solo a partir de qual toda interface pode ser estabelecida. Define os modos indesejados, inesperados e imprevisíveis de acesso e composição. Existem dois sentidos dados para o conceito de interferência, que tentaremos articular nesta relação com a intervenção e a interface.

O primeiro sentido é desenvolvido por Turing (2004 [1948]) ao propor a concepção de uma "máquina desorganizada". Diferentemente das máquinas de Turing ou máquinas universais, a máquina desorganizada não é nem fechada, nem pré-determinada. É composta

por inúmeras partes conectadas aleatoriamente entre si. Estas partes podem ser circuitos, neurônios ou qualquer outro dispositivo comunicante que forme uma rede de interações. É uma máquina que não serve a nenhum propósito pré-definido, ou até mesmo a um propósito geral como a máquina universal. Pode-se dizer também que é mais treinada do que programada. Dependendo de como suas conexões estejam organizadas, simplesmente não pode executar uma determinada instrução que lhe seja dada. É a sua própria organização que deve ser alterada para que se comporte de uma maneira determinada, podendo, inclusive, comportar-se como uma máquina universal após um treinamento muito cuidadoso.

A educação de uma máquina desorganizada se dá através de interferências. Para Turing (2004 [1948]), o que em grande parte marca a diferença entre máquinas desorganizadas e seres humanos é a quantidade de interferência que receberam. No caso humano, a interferência é a regra, e apenas em casos muito específicos, quando alguém tenta focar toda a sua atenção em estados internos, fechando-se para estímulos externos, é possível chegar a um modo quase que desprovido de interferência. Mas mesmo a capacidade de evitar interferências depende de interferências anteriores que ensinaram como fazê-lo.

Na continuação do texto, Turing prefere focar em máquinas desorganizadas afetadas apenas por um mínimo de interferência, por uma questão expositiva, pois a complexidade de máquinas desorganizadas aumenta exponencialmente, de acordo com suas conexões e pontos de entrada e saída. O que nos interessa, contudo, é a ideia da interferência como regra, de máquinas desorganizadas – humanas ou não – cuja existência e forma dependem deste meio completamente permeado de conexões. Se há uma capacidade de limitar as interferências, através do aprendizado de interfaces é porque em uma máquina desorganizada, a princípio, a interferência age sobre todos os seus pontos, ou sobre a maioria. É apenas com a experiência que aprendemos a fechar determinadas conexões, mas nunca completamente. Algo sempre escapa. O primeiro sentido de interferência define esta abertura persistente ao afeto, em tudo o que este tem potencializador e destruidor.

O segundo sentido de interferência, como "interferência monádica", é desenvolvido por Serres (2000). Assim como a interface, a interferência também funciona por traduções e transportes, pois desloca e coloca em comunicação aos mais diversos elementos. A tradução é o transporte de uma referência, sua passagem através de camadas de significados que a conservam ou modificam em maior ou menor medida, seja o sonho de uma linguagem científica ou lógica completamente transparente, seja a noção de tradução como traição. A interface opera a tradução de um tipo de dado a outro, de uma referência a outra, convertendo estímulos mecânicos em um teclado em código binário que representa caracteres alfabéticos

que, por sua vez, serão transformados em imagens visuais por uma rede de interfaces encadeadas, chegando à tela do monitor.

A interferência, entendida como *inter-referência* desfaz a ideia de uma referência única ou central, desmonta a cadeia de traduções que tentam preservar uma determinada referência. Pois cada referência se encontra já aqui e em outra parte, em si mesma é uma multiplicidade de referências. As mônadas de Leibniz problematizavam a ideia de uma referência comum ao expressarem um ponto de vista limitado coextensivo a todos os outros pontos de vista, ao mesmo tempo aqui e em outra parte. Qual a referência da mônada, sua perspectiva ou qualquer outra? "A mônada, *objeto-sujeito*, é mesmo e outro, está aqui e em outra parte. A mônada é uma estrutura de transferência, uma estrutura de transporte, na rede comunicante das interferências monádicas" (SERRES, 2000, p. 144). A mônada nunca está em *um* lugar.

A interferência desloca a evidência da localidade e de sua centralidade. Nada nunca está nem age em um único lugar. Uma linguagem de programação, um algoritmo, uma abstração estão, ao mesmo tempo, no distante mundo das ideias, em uma certa imaterialidade formal, mas apenas porque encontram-se *também* atravessadas pela materialidade do começo ao fim. São interrompidas, interferidas pela recalcitrante insistência do mundo em sua rede de interações monádicas que não espera mais uma harmonia pré-estabelecida para seguir em frente. Tem como princípio colocar as mônadas em circulação, movimento que elide qualquer diferença de níveis ou hierarquia que esteja estabelecida, e tal como o rizoma permite que qualquer ponto conecte-se a qualquer outro. Uma imagem digital é *ao mesmo tempo* imagem, código binário, formato de arquivo, circuito lógico. A interferência abre a caixa-preta de cada um destes níveis, os torna permeáveis à complexidade dos diversos registros em que existe e opera simultaneamente. A interface fecha as caixas-pretas, a interferência as abre.

O que, desde Leibniz, consistia no princípio fundamental da mônada era sua relativa conexão com qualquer outra mônada. Conexão possível pela coexistência no *mesmo mundo.* Na medida em que as mônadas são abertas, não podem mais comunicar-se arbitrariamente com aquelas que se encontram além de sua zona clara e imediata. O mundo passa a ser o conjunto de redes que são produzidas *para* que tais conexões ou desconexões aconteçam. Como Latour (2001) afirma, toda conexão é possível, mas há um preço. A abertura das mônadas reinsere o custo real, as escolhas efetivas e os abortos necessários para que conexões aconteçam. "Tudo, universalmente, deve ser pago. Por que esta lei deveria mudar aqui? Em outras palavras, nada vem de si mesmo, por iteração ou tautologia" (SERRES, 2015, p. 77).[xcix]

Tradução e transporte constituem, primeiramente, a interferência, e só passam a

definir a interface quando ganham uma referência. O modelo interativo propõe a multiplicação de interfaces, a abertura a diversos pontos de vista e perspectivas, como apropriações de referências. É uma interferência controlada. Mas também são possíveis por tudo aquilo que a interferência possui de oportunista e parasitária. O parasita é aquele que rompe a cadeia bem ordenada do ser, que se instaura em qualquer ponto e retira o máximo que pode. Na medida em que não respeita os limites estabelecidos pelas interfaces, a interferência pode aparecer como uma violação ou transgressão. É um *buffer overflow*[30], ou a invasão de um vírus. Nosso corpo também estabelece interfaces, modos específicos de entrada e saída, proteções. A infecção se dá quando estas defesas são quebradas, quando algo consegue passar pelas frestas e subverter as funções do organismo. A interferência sempre tenta se aproveitar das rachaduras e vãos na tentativa de fazer seu código ser executado.

O oportunismo e o parasitismo são os modos próprios da interferência, que surge em qualquer local sem ser convidada. "O parasita colocou-se nas posições mais lucrativas, na intersecção de relações" (SERRES, 2007, p. 43).[c] Os parasitas são como demônios da encruzilhada que, como o demônio de Maxwell, observam e controlam fluxos, mas que cobram um pedágio, sob a ameaça de modificarem, ou bloquearem, completamente aquilo que passa. A interferência se agarra a tudo o que está acessível, respeitando ou interfaces e caminhos. Se a interface é possível é porque encadeia os parasitismos, coloca-se nos pontos mais lucrativos de tal modo que o acesso seja mais fácil e previsível. O parasita sempre toma o caminho mais fácil, possivelmente bloqueando a passagem de outrem. A interface cria um balanço entre facilidade e possibilidade.

A interferência é o ruido, tanto em seu sentido informacional quanto sonoro. É som que permeia o ambiente, que atravessa paredes sem respeitar portas e janelas. Se as mônadas de Leibniz eram fechadas, era justamente para se protegerem contra o ruído. Uma harmonia pura, contudo, é uma péssima alternativa ao ruído. Pois se a interferência pode ser o caos, a pura desorganização que tudo arrasta, a harmonia é a morte, a platitude que elimina toda vida e movimento. "O ruído destrói e horroriza. Mas a ordem e repetição plana estão na vizinhança da morte. O ruído nutre uma nova ordem. Organização, vida e inteligência, contudo, vivem entre ordem e ruído, entre desordem e perfeita harmonia" (SERRES, 2007, p. 127).[ci] É o papel da interface criar a mediação entre ruído e ordem, entre interferência e algoritmo, constituindo um "caminho vivo e inventivo que segue a limítrofe, caprichosa curva onde a simples areia da praia encontra a barulhenta reverberação das ondas" (SERRES, 2007, p. 127).[cii]

30 Um *buffer overflow* acontece quando um programa aloca mais espaço na memória do que lhe é permitindo, transbordando para o espaço alocado a outros programas, sobrescrevendo sua memória ou acessando-a, literalmente interferindo em seu funcionamento.

Temos, então, a interferência de Turing, que pode organizar, educar, aplicar-se ponto a ponto sem nunca poder ser completamente elidida. Com Serres, temos a própria base da comunicação, a interferência como modo de conexão primeiro e generalizado, rede de mônadas em seus cruzamentos mais ou menos parasitários. A introdução da noção de código permitirá compreender de modo mais concreto como tais movimentos são possíveis.

5.4 Abrir o código

5.4.1 Código-fonte, código de máquina, memória

Abrimos mônadas e máquinas. Interagem, trocam entradas e saídas, definem interfaces e permissões de acesso, interferem dos modos mais inesperados, criando um mundo completamente permeado e atravessado pelos movimentos, escolhas e acidentes destes autômatos com formas e objetivos mais variáveis. Todavia, ainda falta definir o *modo* como se dá tal comunicação em sua diversidade. É necessário introduzir a noção de *código* como articulador de conexões e mecanismo de tradução. “O 'deus' de Leibniz e de Laplace eram calculadores universais. O 'deus' do novo Panteão é escritor e leitor universal: tem o código de todas as comunicações, cifra e decifra todos criptogramas” (SERRES, 2000, p. 120). As mônadas agora são lidas e escritas, de acordo com a linguagem definida pelo código, por esta cifra material que coloca o mundo em movimento.

Toda linguagem, natural ou não, é composta por algum tipo de código, assim como qualquer tipo de mecanismo que lide com informação, como o DNA, o código genético. Aqui utilizaremos o termo código, a princípio, em um sentido bem estrito, tal como aparece na teoria e na prática da computação. Mais precisamente, o código possui dois sentidos distintos: como código-fonte, escrito em alguma linguagem de programação, e como código de máquina, código escrito em notação binária, geralmente expresso em hexadecimal, e diretamente executável pela máquina.

O código-fonte é o que muitos consideram como o programa propriamente dito. É sobre ele que todo o trabalho de desenvolvimento e programação se dá efetivamente, pois é escrito em uma linguagem de programação, língua artificial com uma sintaxe rígida e definida de modo a ser mais facilmente compreensível a um ser humano, utilizando termos e expressões de línguas naturais. Contudo, não é um código diretamente executável. Assim como o código de máquina, também é armazenado na máquina como um conjunto de números binários, mas cuja organização diz à máquina que devem ser lidos como um

conjunto de caracteres e não como ordens diretamente executáveis. Em outras palavras, o código-fonte é como uma metalinguagem, que coloca aspas nos termos que define, enquanto o código de máquina é a linguagem objeto que opera diretamente com as instruções. Dizer "2+2" não é o mesmo que executar 2+2.

A passagem do código-fonte ao código de máquina se dá pela utilização de um compilador, um programa que traduz a metalinguagem em código binário executável.[31] O que o código compilado ganha em efetividade, perde em legibilidade ao transformar-se em uma extensa sequência numérica que não reflete nenhuma estrutura linguística diretamente familiar. O código de máquina *age*, opera diretamente na máquina, pois quando é executado existe como trocas de elétrons nos circuitos, passagens, ativações.

Encontramos um segundo tipo de divisão, que não concerne tanto à natureza do código quanto a seu estado, isto é, a diferença entre *processo* e *memória*. O processo é qualquer programa que esteja sendo executado em um sistema operacional. Cada programa ativo constitui um processo que o sistema controla e ao qual aloca recursos e tempo de execução (sendo o próprio sistema operacional um programa em execução). A memória, por sua vez, contém todo o código que não está sendo imediatamente executado, podendo ser volátil (como a memória RAM, que é apagada quando o computador é desligado), ou persistente (armazenada em discos rígidos). Podemos dizer que o processo refere-se a uma ordem temporal, enquanto a memória remete a uma espacialidade (DYSON, 2012). Isto pode parecer contraditório, já que a memória deveria ser o domínio do tempo por excelência, contendo em si o passado nos mais variados graus. A memória de um computador, contudo, é organizada como um arquivo ou biblioteca, onde cada pedaço de informação possui um endereço, uma etiqueta que pode ser acessada pelo processador na medida em que precisa ser recuperada ou reescrita.

A distinção entre uma certa temporalidade do processo e uma espacialidade da memória já coloca em jogo uma questão própria à materialidade do código, pois a memória é possível apenas com os *sólidos*. Serres (2000) distingue entre dois grandes estados ou regimes da matéria: cristalino e amorfo. O estado amorfo, que é próprio dos gases, líquidos e alguns sólidos, constitui-se por uma coleção desorganizada de átomos. Inversamente, o estado cristalino designa todos os sólidos que possuam algum tipo de estruturação organizada, tal como os cristais. A dinâmica dos líquidos e gases segue um modelo termodinâmico e energético, descreve fluxos contínuos, acumulações, passagens, perdas e ganhos de energia. É

31 Ver a tabela em anexo para um exemplo de um programa tanto em código-fonte quanto em código de máquina.

a física do século XIX, a libido freudiana ou o capital marxista. O líquido pode conservar um padrão, como uma onda que vibra na mesma frequência produzindo ondulações regulares cuja repetição pode funcionar como memória. É, contudo, uma memória efêmera, que dura apenas enquanto há uma fonte de energia ativa, animando e mantendo o movimento, similar à memória volátil (RAM) do computador.

Os sólidos tornam a matéria *discreta*, permitem recortar a continuidade amorfa em objetos, formas e superfícies de inscrição – em *dígitos*. Um sólido apresenta as seguintes potencialidades:

> conservação (breve ou longa), retenção, memória; possibilidade de inscrição sobre o sólido, considerado como um "registro" ou "compilador" de uma estrutura completamente estranha a sua natureza própria, possibilidade de recuperação da informação induzida, restituição e, por causa disso, reconstituição do passado. (SERRES, 2000, p. 80)

O objeto sólido é, fundamentalmente, memória, conservação e retenção. O conservado é uma *deformação*, alteração na organização do objeto, interferência em sua estrutura. A informação é deformação. A memória é composta por diferenças na medida em que grava na matéria algo que lhe é estranho: a rocha nada sabe sobre o fóssil que conserva, sua estrutura cristalina é deformada pela fossilização de um organismo, cujo princípio de organização lhe é estranho. Assim como a mônada carrega em si outras mônadas, que pouco conhece ou entende, os sólidos, enquanto fonte de toda memória, são a lembrança da alteridade.

A própria história começa com os sólidos. "Já que o sólido é memória, a história é, primeiramente, uma teoria dos sólidos, quer se trate da história do mundo (rochas e cristais), da história dos vivos (fósseis), da história do homem (esqueletos e ferramentas), da história do 'espírito' (tábuas e estátuas)" (SERRES, 2000, p. 75).[32] A física, entendida como teoria da extensão, do movimento e dos fluxos energéticos não possui história, mas com a introdução da teoria dos sólidos passa a ser histórica. É neste sentido que devemos entender, a "flecha do tempo" e as estruturas dissipativos de que fala Prigogine (2011). O tempo, diferentemente de como é tratado na física newtoniana, não é reversível. Os processos não-lineares inserem uma assimetria entre presente, passado e futuro, onde um efeito atual pode ter sido produzido por inúmeras cadeias causais, cada uma correspondendo a uma história possível, assim como pode ser a causa de diversos efeitos. Sem a conservação desta história é impossível reconstituir uma linha causal perfeita, o que é próprio de fenômenos dissipativos, que anulam

32 Na astronomia a idade da superfície de um planeta ou lua é medido de acordo com o número de crateras que apresenta, isto é, deformações. Uma superfície nova apresenta poucas crateras, isto quer dizer que recentemente (em escala geológica) sofreu algum tipo de renovação pela ação de líquidos: lava (vulcanismo), água (rios/chuva/criovulcanismo) ou ação das placas tectônicas.

a informação pertinente a seu processo de produção.[33] É toda a complexidade das bifurcações de um mundo monádico aberto.

A possibilidade da história não se encontra apenas na conservação própria aos sólidos, mas especialmente naquilo que possuem de *transmissível*. Apenas aquilo que é conservado pode ser transmitido. A duração define-se não por uma passagem do tempo, mas por aquilo que efetivamente resta, por seu registro, inscrição. A memória é física e material do começo ao fim, não há nada de imaterial ou incorpóreo nela. Os efeitos futuros dependem exclusivamente daquilo que foi retido em alguma materialidade, das escolhas operadas por mônadas. O que é perdido, excluído e abortado pode ser retomado unicamente na medida em que possua um mínimo de conservação, de registro em alguma materialidade.

5.4.2 Matéria: software é hardware

É nesta materialidade que encontraremos o código em sua duplicidade. Pois tanto o código-fonte quanto o código de máquina existem apenas enquanto estão incorporados em algum dispositivo dotado de memória, volátil ou não. Uma distinção que costuma ser feita entre *software* e *hardware* parte do princípio que um é físico e material, enquanto o outro á abstrato ou existe "na esfera dos dados". Encontramos o mesmo tipo de argumentação em concepções de linguagem que a consideram como algo não físico, dotada de efeitos ou elementos incorpóreos (como na obra de Deleuze), ou do armazenamento em "nuvem", destes dados misteriosos e etéreos que estão ao mesmo tempo em todo lugar e lugar nenhum. Entretanto, como lembra Sloterdjik (2015), a nuvem é o que também nubla o céu, que bloqueia a visão, isto é, que oculta o fato de que os dados não estão dispersos "por aí", mas encontram-se precisamente localizados em servidores de empresas ao quais os usuários não têm acesso diretamente. O mesmo se aplica ao *software*, que existe apenas na medida em que se encontra diretamente registrado e processado em um *hardware*. *Software* **é** *hardware.*

A concepção algorítmica da computação tende a igualar o programa como código-fonte, ao algoritmo, o que lhe daria uma existência abstrata independente de qualquer meio. Mais precisamente, o algoritmo seria a ideia do programa, enquanto o código-fonte o traduziria nos termos de uma linguagem de programação. Uma vez compilado, o código de máquina poderia ser executado, tornando-se processo, o que constituiria a seguinte cadeia:

33 Imagine uma questão de matemática, a qual é dada como resposta apenas o valor final. Este valor não contém o processo pelo qual foi produzido. Por exemplo, o número 32 foi obtido pela soma 16+16, pela multiplicação 8x4 ou pela divisão 64/2 ? Sem um registro material é impossível reconstituir a cadeia causal de tal resultado.

Algoritmo → Código-fonte → Código de máquina → Processo

Vamos da Ideia eterna à ação no mundo, uma cadeia de traduções transparentes e não problemáticas, onde um termo pode facilmente ser tomado pelo outro. Ou como coloca Galloway (2004),

> código-fonte não compilado é *logicamente* equivalente ao mesmo código compilado em linguagem *assembly* e/ou referenciado a código de máquina. Por exemplo, é absurdo afirmar que um certo valor expressado como hexadecimal (base 16) é mais ou menos fundamental que o mesmo valor expresso como numero binário (base 2). São simplesmente a expressão do mesmo valor.(p. 167).[ciii]

É esta concepção que leva à afirmação forte da Tese de Church-Turing, que assume uma identidade completa entre algoritmo e máquina, onde esta nada mais é do que a expressão do outro. Afirmar que *software* é *hardware*, contudo, vai no sentido contrário, pois ao invés de subordinar a máquina ao algoritmo, coloca este sob o domínio da materialidade própria aos dispositivos em que é implementado.

Em outras palavras, subordinamos o algoritmo ao processo, que passa a ser um de seus componentes e não mais seu ideal. O primeiro sentido de tal inversão implica em uma *abertura do código*. Ao abrir a mônada o mundo passou a ser constituído por uma rede de interações, interfaces e interferências diretas entre mônadas, desmontando o ideal de uma harmonia pré-estabelecida. Tal harmonia, contudo, cumpria uma função dupla, ao estabelecer também a sincronia entre alma e corpo. Abrir parcialmente as mônadas implica em uma interferência direta entre os corpos, ao mesmo tempo em que a alma é conservada como algoritmo, princípio ideal que mantém a harmonia do mundo. Devemos então abrir o código, situá-lo na materialidade própria às interferências monádicas.

Para isto faz-se necessário analisar com mais cuidado a relação entre código-fonte, código de máquina e processo. A ideia de "abrir o código" é central ao movimento do software livre e, posteriormente, à iniciativa do *open source*. Uma vez que o código tenha sido compilado ele perde toda referência a nomes de variáveis, comentários e outras estruturas de dados significativos e facilmente legíveis a um ser humano. O código pode ser descompilado, retraduzido em código-fonte, mas nem por isso torna-se mais compreensível. Se no código original poderíamos ter a declaração "idade_aluno = 15", agora teremos apenas "a=15". Os valores são conservados, mas o sentido é perdido. A transparência da tradução é quebrada, ainda que ao nível da execução nada mude.

Qualquer alteração feita a um programa depende então do acesso direto ao código-fonte original. Um programa proprietário distribui o código compilado e executável, mas retém o código-fonte. Abrir o código implicaria em disponibilizar o acesso ao código-fonte

que, utilizando as devidas licenças de *copyright*, permitiria aos usuários não apenas executar o programa, mas modificar e compartilhar a versão original ou alterada. Deste modo, o mais importante é a *fonte*, o código que pode ser distribuído, que pode *circular*.

O problema desta distinção, como é muito bem apontado por Chun (2008), situa-se na colocação do código-fonte como origem, ocupando o mesmo local ideal que o algoritmo. Em outras palavras, o programa poderia ser reduzido ao código-fonte, que traduziria perfeitamente tudo o que um programa é ou pode ser. Ter acesso ao código-fonte pode revelar muito sobre seu funcionamento, inclusive o que não é evidente apenas por sua execução. Mas, ao mesmo tempo, o código por si só não pode prever tudo o que lhe acontecerá. Como Turing (2004 [1950]) disse já em 1950, ao escrever uma instrução um programador nem sempre sabe o que está fazendo exatamente. Um código que expresse completamente suas consequências incorpora o ideal do algoritmo, enquanto a interação, tal como é proposta por Wegner (1997) coloca em xeque esta visão. Um sistema interativo não é algorítmico. Para Mahoney (2011), o foco no código-fonte afeta também a história da computação, na medida em que muitos programas antigos têm seu código disponível, mas as máquinas que poderiam executá-los não existem mais, isto é, uma determinada materialidade não foi conservada.

Se pretendemos abrir o código devemos ir além do acesso ao código-fonte. Mais precisamente, devemos recolocar a questão do que constitui uma *fonte*.

> *Então, código-fonte apenas se torna fonte após o fato*. Código-fonte é mais precisamente um *re-curso* [re-source], ao invés de uma fonte. Código-fonte torna-se fonte quando é integrado com portas lógicas (e em um nível ainda mais baixo, com os transistores que compõem esses circuitos); quando expande-se para incluir bibliotecas de *software*, quando emerge como código gravado em chips de silicone; e quando todos esses sinais são cuidadosamente monitorados, cronometrados e retificados (CHUN, 2008, p. 307).[civ]

A fonte não designa uma origem, mas uma composição que se dá apenas *a posteriori*, na medida em que estabelece conexões com o mundo no qual pode tornar-se efetivamente um processo.

O código de máquina ocupa uma posição singular pois é "a única linguagem que pode ser executada" (GALLOWAY, 2004, p. 165).[cv] Toda linguagem, humana ou não, possui um caráter performático. Enunciações são sempre ações que afetam o mundo em maior ou menor grau. O código de máquina é igualmente performance, em toda a sua extensão, e se pode ser executado é porque age diretamente na matéria, onde cada um de seus símbolos é uma transmissão de energia que funciona como comando em um mecanismo. Seria uma performance "mais direta". Mas como ressalta Chun (2008), toda performance, sendo ou não código, tem seu sucesso definido pelo contexto no qual é enunciada, não sendo performática

em si. A performance depende de sua associação a uma comunidade, o que no caso do código de máquina significa a sua associação tanto a um meio propriamente técnico quanto a uma estrutura social, política e econômica – um coletivo, como veremos mais a frente.

Esta ligação fica evidente quando Mackenzie (1993) mostra como a própria aritmética realizada por um computador deve ser negociada. Um computador, e mais especificamente um processador, é um mecanismo digital, e por isso não tem como reproduzir com exatidão as partes fracionárias de números reais (e, também, por ter uma memória limitada). Como um processador deve arredondar um número? Aproximando-o de zero ou utilizando outros métodos de aproximação? Em ambos os casos a questão implica em decidir *como o processador vai errar*. Tais erros aparecerão apenas em casos muito específicos, e dois processadores que utilizem métodos diferentes apresentarão resultados distintos para o mesmo cálculo. A finitude das máquinas impede que as consideremos como simples traduções diretas de algoritmos ou de códigos, e nos obriga o tempo inteiro e negociar e renegociar seu funcionamento e o próprio sentido da noção de "exatidão".

Portanto, a materialidade opera em dois registros, como interferência, ao inserir suas exigências próprias *em meio* às necessidades formais do código, e como interface, ao criar condições estáveis que permitem que a execução de código seja possível. Por sua vez, o código desempenha o papel de agente comunicante em interfaces e interferências. Em outras palavras, há um aspecto formal tanto no código-fonte quanto no código de máquina, mas esta formalidade não se dá como um princípio abstrato e sim, como um encadeamento de ações sobre e a partir de um mundo.

> A computação no mundo real – que como disse, chamo de "computação em estado selvagem" [computation in the wild] – acaba não sendo formal, em nenhuma leitura discernível deste predicado recalcitrante e disputado. Particularmente, o pressuposto comum de que computadores envolvem, ou são constituídos, pela manipulação de símbolos "independente de sua semântica" demonstra ser muito exagerada empiricamente. Para reduzir uma investigação complexa em uma sentença, colocaria desta forma: o constructo "manipulação de símbolos formais" definitivamente falha porque, como em muitos casos do mundo real, domínios semânticos estão implicados constitutivamente em processos computacionais definidos em, sobre, ao redor e através deles. Computadores do mundo real estão ativamente envolvidos em seus meios; quer dizer, são jogadores significativos nos próprios mundos que representam. Em suas idas e vindas, realizam um uso essencial e efetivo, do mesmo modo que essencialmente afetam os próprios estados de coisas dos quais suas estruturas simbólicas tratam (semanticamente). A concomitante "co-constituição" de símbolos e referentes, na composição de um processo computacional, derrota o tipo de independência entre símbolos e referência que o formalismo presume (SMITH, 2010, p. 29).[cvi]

Esta capacidade do código produzir sentido diretamente no mundo e a partir dele depende acima de tudo do caráter causal do mundo em que se dá (CLELAND, 2004). Seja a causalidade física, seja o campo social ou coletivo do qual participa na construção. Um

iPhone tem uma velocidade de processamento e memória muito maior do que os computadores da nave Apollo 11, que levaram o homem à lua. Contudo, um iPhone jamais resistiria à radiação do espaço, assim como não possuiria um código tão robusto quanto o de uma nave ou sonda espacial. Em termos puramente abstratos, ambos seriam capazes de executar o mesmo código, com um sendo incrivelmente mais rápido que o outro. Mas se o caráter causal do mundo for considerado, a distinção se dará entre um código executado, ainda que lentamente, e um pedaço de *hardware* com circuitos fritados pela radiação.

Pode ser um exemplo extremo, mas nosso cotidiano está repleto de experiências em que a materialidade da computação está implicada diretamente: placas de vídeo que super-aquecem, placas-mãe oxidadas pela poeira, mal sinal de *wi-fi*, desgaste mecânico de teclados e telas *touchscreen* e assim por diante. Mas a materialidade possui um sentido muito mais amplo que a realização imperfeita de um ideal lógico. Se o computador não pode ser reduzido a uma concepção algorítmica, a definição do que é computável não se aplica mais. Um processo que seria incomputável para uma máquina de Turing, pode ser computável na prática, tanto por um comportamento aleatório do *hardware*, tanto quanto por uma entrada fornecida por um usuário ou outro programa. Inversamente, um código computável pode tornar-se interminável por causa das mesmas razões. O modo como a computação deve ser pensada depende das características materiais, e não o inverso. Como vimos, a memória depende do sólido, da estrutura organizada, do mundo que constitui o substrato a partir do qual pode emergir.

Podemos compreender melhor porque a interferência é primeira em relação à interface. Na prática, a computação faz uso de tudo o que o caráter material da máquina tem a lhe oferecer, é oportunista, cria atalhos. É isto que permite a criação de processadores cada vez mais rápidos, com o desenvolvimento de transistores praticamente ao nível atômico. Se o computador nos passa a ideia de rapidez , de exatidão e infalibilidade, é porque de algum modo a matéria permite a criação de um meio com um nível de estabilidade altíssimo. Foi justamente isto que surpreendeu quem estava envolvido na construção dos primeiros computadores, pois demonstraram que era possível desenvolver computações confiáveis utilizando partes não-confiáveis, como tubos de vácuo (DYSON, 2012). A matéria dá estabilidade, mas o faz sempre por um fio. A interferência, o ruído precedem tudo. "E ainda, às vezes, há acordo. A coisa mais incrível no mundo é que acordo, entendimento e harmonia existem às vezes" (SERRES, 2007, p. 121).[cvii] Em um mundo de mônadas abertas, de algum modo, a harmonia ainda se produz a partir do desencontro e do caos. É esta inversão que temos de realizar, ir da mônada à harmonia, da matéria à computação, da interferência à

interface e do código ao processo.

5.4.3 Quase-objeto

Este foco na materialidade e na execução não equivale a considerar o domínio do código e de máquina em ação como o único válido, descartando o código-fonte e o algoritmo. Não há um domínio mais verdadeiro ou originário do código, pois quando o consideramos como processo, são todos os seus componentes e níveis que devem ser levados em conta. Portanto, se o próprio do código de máquina é ser executável, dado o modo como se apropria do caráter causal da matéria para tornar isso possível, em que consiste a definição do código-fonte?

Como vimos, o código-fonte não é executável, e isto permite que se abra a potências e afetos distintos daquele do código de máquina. Em geral o código-fonte encontra-se inscrito em algum dispositivo de memória eletrônico, afinal de contas escrevemos programas utilizando outros programas e computadores. Mas nada impede que o código seja escrito em um pedaço de papel, por exemplo. Nem todo código-fonte tem como destino ser compilado, virar código de máquina (CHUN, 2008). O código-fonte, na forma de uma linguagem de programação, especialmente nas de alto-nível, constitui-se como uma abstração. As linguagens de programação, ao tornarem-se gerais, podem abstrair quase que completamente a máquina onde serão executadas. Seria possível escrever um programa em uma linguagem e compilá-lo para as mais diversas máquinas e arquiteturas. O código de máquina encontra-se diretamente atrelado a uma máquina ou determinada classe de máquina (processador com arquitetura Intel ou ARM, 32 ou 64 bits). O código-fonte, na medida em que mantém sua separação do código binário, pode *circular*.

Computadores podem ser considerados como implementações ou traduções de máquinas universais. Cada um pode, a princípio, executar qualquer programa computável dentro dos limites de sua memória e de outras propriedades físicas e arquiteturais. O programa a ser executado deve ser inserido na máquina de alguma forma, seja diretamente em numeração binária gravada nos circuitos lógicos, seja através de uma linguagem de programação. Se, teoricamente, todas as máquinas universais são equivalentes, na prática isto é muito diferente, pois cada computador pode executar apenas programas cuidadosamente desenvolvidos para sua arquitetura ou para as convenções que utiliza ao interpretar o código binário. O código de uma máquina pode ser completamente ilegível para outra, apesar de serem logicamente equivalentes. A princípio podem executar o mesmo algoritmo, desde que

seja escrito de formas absolutamente distintas. A equivalência depende de um árduo e complexo trabalho de tradução.

Como vimos no terceiro capítulo, era este o cenário dos primeiros computadores nos anos 40 e 50, onde cada um possuía sua linguagem de programação própria e incompatível com todo o resto. Mesmo sendo máquinas universais, cada uma funcionava como uma mônada com seu mundo próprio, fechadas de modo ainda mais profundo que as de Leibniz. A criação de linguagens gerais permitiu a abertura desses mundos fechados, fazendo com que todas as máquinas pudessem compartilhar o mesmo mundo, o mesmo código, apesar de manterem seus traços individuais. Podemos dizer que o código-fonte é uma parte do mundo, escrita compartilhada e comum, enquanto a perspectiva ou ponto de vista da máquina-mônada é definido por seu compilador, por sua capacidade de traduzir o mundo de acordo com sua singularidade material e formal.

O código-fonte possui duas faces ou, mais precisamente, duas interfaces. A primeira interface, como acabamos de ver, volta-se para a máquina e estabelece uma relação de comunicação e tradução com seu código. A segunda interface media a relação com o mundo humano, e faz a tradução inversa à anterior, tornando o código de máquina em algo que possa ser mais diretamente assimilado a um domínio de práticas humanas. Para tomar um caso atual, a programação orientada a objetos cria um meio no qual o programador define objetos e suas relações, com maior ou menor isomorfismo em relação ao aspecto do mundo que busca modelar. Ao nível do código de máquina estas relações são traduzidas em leitura, escrita e movimentação de símbolos a locais específicos da memória. É a mesma diferença entre descrever um fenômeno biológico em termos de organismos e suas relações com o ambiente, e entre descrevê-los exclusivamente como deslocamentos de átomos e trocas energéticas.

A circulação do código-fonte se dá entre estas duas interfaces, entre o mundo humano e o mundo da máquina. Neste deslizamento o código-fonte funciona como um *quase-objeto*. "O quase-objeto não é um objeto, mas o é em todo caso, já que não é um sujeito, assim como está no mundo; é também um quase-sujeito, uma vez que marca ou designa o sujeito que, sem ele, não seria um sujeito. […] Este quase-objeto, enquanto está sendo passado, constitui o coletivo, se para, cria o indivíduo" (SERRES, 2007, p. 225).[cviii] O quase-objeto é uma passagem, que ao final produz um sujeito e um objeto. Poderíamos pensar, em um primeiro momento, no algoritmo. Enquanto é executado se comporta como um quase-objeto. O término de sua execução define um objeto, sua saída, e um sujeito, o usuário.

O quase-objeto, contudo, não designa apenas uma passagem, mas um processo no qual um coletivo é estabelecido, cria um "nós". Este "nós" não é simplesmente a reunião ou a

coleção de vários “eus”. “O 'nós' não é a soma de 'eus', mas uma novidade produzida por legados, concessões, retiradas, resignações do 'eu'” (SERRES, 2007, p. 228).[cix] O coletivo é uma *transmissão* de eus, é o jogo pelo qual o eu circula entre diversos corpos, não se fixando nem em um nem em outro. A abertura do código coloca o código-fonte em movimento, o transforma em quase-objeto capaz de estabelecer um coletivo. O código fechado é proprietário, pode e deve ser atribuído a um eu, a alguém que o possua e possa ditar o modo como deve ser desenvolvido, utilizado e distribuído. O código aberto só e possível na medida em que forma um coletivo. Mesmo se o código é aberto e disponibilizado com as devidas licenças de *copyleft*, não constitui um quase-objeto se não for retomado por outros, se não fizer os eus circularem a partir de suas contribuições para o código, seus compartilhamentos e utilizações.

Se o movimento do software livre e do *open source* coloca uma ênfase tão grande no código-fonte, chegando mesmo a torná-lo a própria *fonte*, é por causa dessa capacidade do código de ser um quase-objeto, de constituir coletivos. Como Latour (2012) afirma, um coletivo não é composto apenas por humanos, é um agregado de humanos e não-humanos, em tudo o que ambos têm de agencia em suas especificidades. Pois o código-fonte não é uma linguagem qualquer, seu poder reside em sua habilidade de comunicar-se com o código de máquina, de *tornar-se* tal código (por mais que tal transformação não seja uma necessidade). O foco exclusivo dado ao código-fonte provém justamente de ser a interface, o acesso ao código de máquina, a tal ponto que este pode ser facilmente esquecido, aparecendo em sua especifidade apenas quando algo falha. O coletivo constitui-se nessa interface entre humanos e máquinas, e todas a passagens e traduções entre os níveis de código.

Há, ainda, um segundo coletivo, constituído agora diretamente pelo código de máquina, que assim como o código-fonte, também circula, também é quase-objeto, mesmo que de forma distinta. O poder do código de máquina encontra-se no fato de estabelecer uma relação direta com o caráter causal do mundo, de apropriar-se das oportunidades da matéria para ser executável. A execução é um modo de circulação, é através dela que o programa torna-se processo, que pode ser copiado, transmitido, modificado e apagado. Circulação cuja possibilidade depende do código binário constituir-se como a *língua franca* das máquinas computacionais. Na medida em que as máquinas são padronizadas, determinados dialetos binários tornam-se predominantes e criam um imenso meio de propagação para o código. Isto reflete profundamente a força da própria técnica que, como coloca Simondon (2007), constitui-se pela divisão do mundo em componentes elementares. Desconecta a figura do fundo e transforma as figuras em elementos que podem ser transmitidos. A padronização não

depende apenas de convenções sociais ou econômicas, mas da própria tecnicidade daquilo que é padronizado. O meio binário/digital só é possível por existir em uma determinada materialidade que possibilita, e até mesmo facilita, tal padronização.

A potência das máquinas universais, e da arquitetura de Von Neumann que as implementa como base dos computadores modernos, situa-se na colocação de programas e dados na memória, isto é, em transformar tudo em código de máquina. O próprio código-fonte para ser compilado deve existir primeiro como código de máquina, mas que representa caracteres textuais ao invés de ser código diretamente executável. É este nivelamento, ou univocidade, que dá abertura para uma das criações mais fascinantes do domínio computacional: o vírus, este pedaço de código que se propaga desenfreadamente, que modifica outros códigos, os invade, parasita e toma sob seu controle.

Ao colocar programas e dados no mesmo nível, o computador enquanto máquina universal é capaz de modificar seu funcionamento, de fazer com que o código de máquina aja sobre si mesmo, um processo que possui muitas similaridades com a reprodução biológica. Na medida em que criamos um meio computacional ubíquo, que nos cercamos por todo tipo de máquinas universais, por máquinas e seus códigos, podem existir vírus, *worms* e toda sorte de *malware* que se propagam facilmente por milhares ou milhões de dispositivos. Todavia, este movimento não se restringe à viralidade não intencional de programas invasivos. Nós propagamos o código sem parar, circulamos pelo código, nos subjetivamos e objetivamos nele, do mesmo modo em que o código transforma-se em objeto e sujeito. A máquina não se reproduz sozinha, mas talvez isto não seja necessário, pois assim como o mundo que Butler (1955) concebe em *Erewhon*, nós é que fazemos parte do sistema reprodutor das máquinas, do mesmo modo que a vespa participa da reprodução da orquídea. Este é o verdadeiro sentido de um código aberto, um código que circula e faz circular, que não fica apenas dentro da máquina, mas cria uma abertura onde se produz um coletivo de humanos e não humanos, subjetividades objetividades.

5.4.4 Processo

A memória começa com o sólido, com a estrutura organizada e sua deformação. Discos *rígidos* (HDD) e discos de estado *sólido* (SDD) constituem unidades de armazenamento persistente, código que não desaparece quando a máquina é desligada. É um código que não circula. Ou se o faz é de forma indireta, quando a unidade de armazenamento é transportada fisicamente de um local ao outro. Em todo caso, tal código só pode ser efetivo

na medida em que é lido e não apenas transportado. O sólido, que é a base de toda transmissibilidade, deve conectar-se a um fluxo, deve ser *processo*.

Como já dissemos, na terminologia da ciência da computação, um processo é qualquer programa que esteja sendo executado, um código ativo, controlado pelo sistema operacional, ele próprio um programa em ação, que define como o tempo do processador será dividido entre os processos. Um processador, que não possua mais de um núcleo, funciona de forma completamente linear, executando uma instrução por vez, sem paralelismo. O sistema operacional distribui estas instruções e as intercala entre os processos. O número de instruções disponíveis é definido por um relógio, que divide temporalmente as execuções em milissegundos. A rapidez de um processador é definida por quantas instruções são executadas em uma determinada quantidade de tempo. Quanto maior a prioridade de um programa, mais tempo de execução lhe é disponibilizado, podendo acontecer de um programa dominar completamente o processamento da máquina, o que geralmente constitui o seu "travamento" ou congelamento.

Esta definição de processo, contudo, é muito limitada, pois reduz o programa a uma série de instruções executadas temporalmente no contexto de um sistema operacional. Propomos então entender o processo não como a simples atualização de uma memória persistente, mas como o movimento no qual o próprio código é constituído, que engloba algoritmo, código de máquina e código-fonte em um devir, em uma dinâmica performática e auto-criativa. Em outras palavras, concebemos o processo nos termos de Whitehead (1978), como o devir de entidades atuais, como um avanço criativo em direção à novidade. O processo não pode ser reduzido à simples realização de uma possibilidade ou harmonia pré-estabelecida, pois é justamente o que coloca em jogo os mais diversos componentes cujo destino será decidido apenas em sua interação, mistura de ruído e ordem. A determinação é o resultado do processo, não seu princípio.

Podemos conceber a filosofia de Whitehead, também, como uma monadologia, tal como o faz Deleuze (1991). Whitehead abre completamente as mônadas, que chama de "entidades atuais" ou "ocasiões atuais". Cada entidade atual é composta por outras entidades, da mesma forma que compõe outras. Não são substâncias fechadas em si mesmas, assim como as mônadas contêm, ou mais precisamente, são compostas por outras. O modo pelo qual uma entidade é tomada por outra chama-se *preensão*. A interação entre as entidades atuais se dá por preensões, onde uma entidade é *sentida* por outra. Este sentimento ou fruição culmina em uma satisfação, a forma final da entidade atual, na qual a indeterminação do processo é determinada. Esta satisfação não constitui um fim absoluto, fechado a qualquer mudança

futura, pois a entidade atual faz parte de outras entidades e processos, nos quais é composta e recomposta, constituindo uma unidade que é sempre parcial.

Uma entidade atual não é nem objetiva nem subjetiva em si mesma, tal como uma substância. A objetivação se dá apenas quando uma entidade atual é preendida por outra, e quanto mais ela é tomada por outras entidades mais se objetiva, adquire uma realidade cada vez mais consistente, do mesmo que objetiva e subjetiva as entidades que a preendem. Por sua vez, a subjetivação acontece quando uma entidade preende outra, é a forma que toma após a finalização do processo, sua satisfação. O mundo é constituído inicialmente por uma multiplicidade, por entidades que estabelecem relações apenas através de *preensões negativas*, que se excluem ativamente umas das outras, constituindo uma disjunção. O movimento de criação que atravessa o mundo vai, progressivamente, transformando a disjunção em conjunção, onde entidades vão preendendo umas às outras, dando consistência ao mundo e o levando em direção a uma maior concretude. O processo pode ser concebido de dois modos: concrescência e transição. A concrescência é modo pelo qual uma entidade atual é formada, constituindo-se como uma unidade que retira toda a indeterminação de sua composição, através de conexões cada vez mais estáveis. Inversamente, a transição opera a desconexão das entidades atuais, as decompõe, é um "eterno perecer".

Whitehead nos dá os elementos para pensar a abertura das mônadas em toda a sua extensão. É uma proposta similar à de Tarde (2007b) que propõe uma ontologia que não mais baseada no *ser*, mas no *ter* ou *haver*. Não somos, temos ou havemos. O mundo é o resultado de um movimento incessante de mônadas ou entidades atuais que preendem umas às outras, que se capturam mutualmente constituindo agregados, estabilidade e criação. Podemos, agora, compreender o sentido mais profundo da interferência e da interface. A interferência não se dá apenas no exterior das mônadas, mas consiste justamente nessa interpenetração constante, desse atravessamento incessante que desconsidera qualquer interioridade ou exterioridade. É uma capacidade de preensão ilimitada, de tomar e ser tomado sem restrições. A interface é produzida como satisfação, como o resultado de uma concrescência que elimina toda a indeterminação e cria um meio estável e controlável de trocas.

O que o processo permite compreender é o modo pelo qual o código, em todos os seus aspectos, não é uma abstração ou representação do mundo, não é nem formal nem determinado. A processualidade do código constitui um jogo de preensões que tomam para si materialidades, causalidades, criando coletivos, estabelecendo comunicações e intersubjetividades, mesclando ruído e ordem, corrupção e computabilidade, solidez e fluxo.

Fazer uma monadologia nos deu as ferramentas necessárias para compreender onde

podemos encontrar o código, a morada dos elusivos zeros e uns. Sendo processo o código constitui-se como uma pluralidade de conexões e desconexões, como um quase-objeto que define, determina sujeitos e objetos apenas ao fim de seu processo. Não podemos mais nos prender a um algoritmo, a uma harmonia pré-estabelecida ou a um código-fonte que já conteria em si todo o futuro possível. É em meio às interfaces e interferências, enquanto acontecem, que devemos descobrir o destino do código, este processo como jogo de preensões que busca determinar um mundo, retirar um pouco da harmonia do ruído que permeia e preenche tudo.

6 Daemon

E se um dia, ou uma noite, um demônio lhe aparecesse furtivamente em sua mais desolada solidão...
- Nietzsche

Todo déspota ama a simetria.
- Gabriel Tarde

6.1 O retorno do *daemon*

Enquanto explica a Fedro as razões de terminar seu discurso sobre o amor, Sócrates diz, abruptamente, que um sinal divino o despertou, o sinal que habitualmente intervém impedindo-lhe de fazer o que deseja. Este sinal, voz vinda de outro lugar, que o inspira, guia e aconselha, o transforma em certa espécie de adivinho, ainda que a adivinhação seja apenas sobre si mesmo (PLATÃO, 2000). O sinal divino é o *daemon*[34] socrático, espírito que o acompanha e cuja intervenção o impede de cometer atos indevidos. O *daemon* é essa voz que controlava os gestos e ações de Sócrates e que, mais tarde, Proclo (1816) viria a dizer que é o "piloto do todo de nossa vida" (p. 265).[cx]

No *Banquete*, Sócrates narra seu diálogo com a sacerdotisa Diotima, que dissera-lhe que o *daemon* é algo entre mortal e imortal, que tem

> o poder de transmitir aos deuses o que vem dos homens, e aos homens o que vem dos deuses, de uns as súplicas e os sacrifícios, e dos outros as ordens e as recompensas pelos sacrifícios; e como está no meio de ambos ele os completa, de modo que o todo fica ligado todo ele a si mesmo. […] Um deus com um homem não se mistura, mas é através desse ser que se faz todo o convívio e diálogo dos deuses com os homens... (PLATÃO, 1983, p. 34–35).

O *daemon* existe entre duas ordens heterogêneas, é uma figura intersticial que não pertence a nenhum mundo, mas vive em suas lacunas, nos pontos de comunicação, onde media e controla as passagens e permite que as diferenças sejam articuladas.

Em sua origem etimológica, o *daemon* está ligado à mesma raiz de dividir e distribuir. Inicialmente o *daemon* era um termo usado de modo equivalente a "deus", vindo depois a significar um ser intermediário entre os deuses e os humanos – mais que um humano,

34 "*Daemon"* é a versão latina do termo grego "*daimon"*.

menos que um deus. No mundo greco-romando o *daemon* viria a ser associado ao *numen*, uma força divina, inteligência que permeia as coisas, correlato espiritual da matéria (HOOKWAY, 2014). Essas tendências são sintetizadas por Proclo (1816), que divide os *daemons* em seis tipos, onde os primeiros, os mais divinos, podem até mesmo ser tomados por deuses, pois se encontram mais próximos deles. Descendo a cadeia dos seres, os *daemons* regem a ascensão e o declínio das almas, transmitem as produções dos deuses às naturezas secundárias, inspiram naturezas parciais com vida, ordem, razões. Também unem o que há de extremo nos corpos, permitindo que algo permaneça em meio à corrupção das coisas que vivem sob o tempo. Por fim, em seu nível mais baixo, os *daemons* conectam o divino à matéria, conservando-a e protegendo suas formas opacas. Nos seus mais variados níveis, os *daemons* constituem a tessitura do mundo, cuja existência liminar completa o humano e o divino, dando forma e conservando as coisas. Sem a complexa trama formada pela ação dos *daemons*, os deuses seriam incapazes de agir sobre o mundo, enquanto os humanos perderiam toda a eficácia de sua ação, privada do poder de comunicação e estabilização.

Por mais que o *daemon* de Sócrates aproxime-se de uma voz interior, como a voz da consciência, ainda pertence a um mundo que é animado por forças e habitado por inteligências que ultrapassam o humano. O mundo grego antigo não havia rompido sua conexão com o divino, não fora dividido entre objetividade e subjetividade, pois como algo poderia ser completamente subjetivo ou objetivo em um mundo onde as próprias coisas falavam, os seres metamorfoseavam-se uns nos outros e os deuses andavam entre os mortais? A *episteme* ainda não triunfara sobre a *gnosis*, o conhecimento adivinha tanto da razão que constatava o estado de coisas, quanto da inspiração divina que transformava os sujeitos e o mundo (NELSON, 2001).

A existência do mundo era incerta. Os deuses deveriam intervir constantemente para conservá-la, enquanto homens e mulheres agiriam de modo a evitar os caprichos divinos. Os *daemons* circulavam entre deuses e humanos, tecendo a rede que mantinha o mundo funcionando, mas não eram os únicos mediadores. Existia uma segunda classe de deuses, os "deuses terrestres", criados pelas próprias mãos dos humanos: as estátuas. Neste momento, os ídolos não eram representações materiais dos deuses, eram os próprios deuses, ainda que inferiores aos deuses olímpicos. Nos templos, os sacerdotes atendiam e cuidavam das estátuas, as alimentavam, lavavam, e levavam-nas a passeios. Ainda que a estátua falasse ao fiel através da voz do sacerdote, oculto atrás ou no interior da estátua, o artifício não era menos divino. A voz do sacerdote era a voz da estátua, do mesmo modo que esta era o deus encarnado. Estes ídolos chegavam a ser considerados mais divinos que os seres vivos, pois

sua natureza fixa e estática refletia a imutabilidade do mundo divino imortal (NELSON, 2001).

Com os romanos, o *daemon* torna-se o *genius*, um deus sob cuja proteção cada pessoa vive. Todo nascimento é um duplo nascimento, pois com o sujeito nasce também o seu *genius*, cujo nome provém de "geração"[35], ser que gera e é gerado por seu duplo. O *genius* é um espírito protetor, que acompanha o indivíduo em cada instante de sua vida, até sua morte – cada aniversário é a celebração desta da aliança, de uma vida duplicada. O *genius* é um deus pessoal, cuja jurisdição restringe-se à pessoa que observa, assim como o *daemon* de Sócrates dava-lhe poderes divinatórios que concerniam apenas a si mesmo. A adivinhação não pode provir exclusivamente do indivíduo, vem sempre de fora, pois se refere a algo que ainda não aconteceu, que ultrapassa o sujeito. O *daemon*, e depois o *genius*, é a conexão com este além do sujeito, é seu *ingenius*, o lampejo de engenhosidade e inspiração que o toma e o leva para além de sua potência (AGAMBEN, 2007; SLOTERDIJK, 2011).

O momento genial, em que o *genius* se manifesta e podemos dizer que alguém tem gênio, não se refere a uma propriedade interior, a um traço de personalidade ou faculdade cognitiva privada, mas a aquilo que há de *impessoal* em nós. Como coloca Agamben (2007), o *genius* é ao mesmo tempo o que temos de mais íntimo e pessoal, e o mais impessoal, "a personalização do que, em nós, nos supera e excede" (p. 16). Em nós, o *genius* é a própria estranheza, esse desconhecido pré-individual do qual provém toda a inspiração, todo pensamento e criação, é o que impede que o sujeito feche-se em sua identidade pessoal e tautológica. Para esta tradição, a noção de um *cogito*, de um pensamento que pensa a si mesmo como fundamento do ser é completamente incompreensível. Ao invés disso, caberia afirmar "sou pensado, portanto sou" (SLOTERDIJK, 2011, p. 417).[cxi]

Se o *genius* é essa potência impessoal na pessoa, o pré-indivíduo no indivíduo, é, também, o *Lar*, o deus da casa. O lar romano era habitado não apenas por seu moradores vivos, mas por seus ancestrais, espíritos invisíveis, assim como outros espíritos que animavam a casa e garantiam sua boa fortuna ou destino trágico. "Inicialmente, viver em recipientes como casas sempre tem um caráter duplo: significa tanto a coexistência de humanos com humanos e a comunidade de humanos com seus companheiros invisíveis" (SLOTERDIJK, 2011, p. 422).[cxii] O impessoal designa este espaço híbrido compartilhado por *daemons*, *genius*, *Lares*, espíritos ancestrais e deuses domésticos. O indivíduo existe no cruzamento destas forças, habita um mundo densamente povoado por inteligências e sensibilidades que o atravessam, que o possuem e que ampliam sua cognição, que o controlam, aconselham e lhe

35 No inglês a geração, *gen-eration* mantém a raiz original de *gen-ius*.

concedem acesso a poderes divinatórios.

No mundo cristão, o *daemon* transforma-se no demônio, figura que nos é muito mais familiar. O demônio é o subproduto da interiorização cristã. O sujeito, agora dotado de livre-arbítrio, nasce como pecador, o mal já se encontra desde o início encrustado em sua carne. Fora, há apenas tentação, as seduções da carne e da matéria que se aproveitam da vontade e do pecado que há em toda alma para levá-la à decadência e ao mal. A única comunhão, a única exterioridade possível é transcendente, a união com Deus e com Cristo, mediada por todo o sacerdócio cristão.

O demônio se apresenta como uma figura traiçoeira, que tenta o sujeito, que o desvia de sua vida reta, o retira da comunhão interna e íntima com Deus e o joga em meio à exterioridade, à mescla impura da matéria e dos corpos. A inspiração cede seu lugar à possessão. Ser tomado, atravessado por uma força desconhecida e impessoal é sucumbir ao mal. Tudo o que viole o sagrado invólucro da alma, com seu núcleo de identidade e livre-arbítrio pode ter uma existência apenas negativa, ilusória e maléfica. Qualquer voz que não seja a voz da consciência, a voz de Deus, ou de seus emissários, os anjos, é a voz do demônio (NELSON, 2001; THACKER, 2011).

As estátuas, estes deuses terrestres criados pela mão humana, são quebrados. O cristianismo é atravessado por uma profunda iconoclastia. Qualquer estátua que não se limite a ser uma representação do divino torna-se fetiche, heresia e objeto de idolatria. As técnicas divinatórias não são mais que pura superstição. O mundo passa a ter uma ordem estrita e bem definida, ganha propriamente uma natureza que exclui todo o sobrenatural. A mediação é monopolizada por Deus e seus acólitos. A pluralidade de contatos, cruzamentos, intervenções, conflitos e arbitrariedades do mundo pagão é fechada sob o signo binário do bem e do mal.

É algo desta iconoclastia cristã que a constituição moderna preserva e justifica. Se o cristianismo expulsa do sujeito toda exterioridade, os modernos condenam as coisas à passividade e à mudez. Todo o problema de cristãos e modernos é o de como evitar misturas indevidas, de como separar o natural do não-natural. Purificar os híbridos, colocar os deuses de um lado e estátuas de outro. Com a modernidade, não é mais Deus quem realiza a partilha entre humanos e não-humanos. Cabe à ciência de um lado, e a política de outro, estabelecer onde começam e terminam natureza e sociedade, o que existe por si mesmo e o que é obra do fazer humano. Mais precisamente, devem conciliar as construções da natureza no laboratório com a sua independência fora dele, do mesmo modo que a sociedade nos ultrapassa completamente ainda que seja por nós construída. Em última instância, quando a partilha não pode ser feita, quando ciência e política não entram em um acordo, Deus retorna para realizar

a divisão, seja como progresso, transcendência ou como razão. (LATOUR, 1994).

A psicanálise, por sua vez, leva o processo de interiorização cristão ao seu limite. O demônio agora faz parte de nós. A possessão demoníaca vem de dentro, não de fora. A impessoalidade do *genius*, este Outro que age em nós, é o nosso próprio inconsciente, este receptáculo privado que absorve absolutamente tudo. Em um certo sentido, a psicanálise nunca expulsou os híbridos, pois o inconsciente é o local por excelência das misturas, combinações e substituições mais improváveis e irracionais. Não é por acaso que reencontramos Édipo no inconsciente, replicando no interior de cada sujeito o mundo mítico que fora demonizado pelo cristianismo. Dentro de cada sujeito reencontramos sátiros, centauros, deuses, estátuas que falam, a repetição incessante das grandes tragédias e o embate interminável entre forças inconciliáveis – vida e morte (THACKER, 2011).

Contudo, a partir do momento em que algo transborda, em que alguma entidade mítica escapa de nossa interioridade, a psicanálise deve entrar em ação. Quando as vozes que escutamos deixam de ser as vozes de nossa consciência e passam a ser vozes que vêm de fora, de Deus, do Diabo ou deste Outro, e que deixam de representar para controlar, pilotar e intervir diretamente, entramos então no campo da psicose. A psicanálise, e em grande medida a psicologia, aceita tudo, as fantasias mais absurdas, os sonhos mais fantásticos, desde que jamais rompam o invólucro do corpo que delimita o indivíduo, permanecendo como sonho ou imaginação (THACKER, 2011). Encontramo-nos ainda sob a exigência do fetiche e da representação, mantemos a iconoclastia cristã (LATOUR, 2002). Para além disso, há apenas a loucura e a patologia – os novos demônios.

Apesar da demonização cristã, da profilaxia moderna e da interiorização psicanalítica, os demônios nunca nos abandonaram. Na física, o demônio de Laplace seria uma inteligência sobre-humana vasta o suficiente para conhecer o estado atual de cada átomo, cada corpo no universo, permitindo-lhe prever todos os movimentos futuros dos corpos, assim como determinar seu passado com a mais perfeita exatidão. Este demônio, em seu caráter sobre-humano preserva algo da divindade do *daemon*, o que lhe permite obter poderes divinatórios que não são acessíveis ao homem. Mas enquanto o *daemon* era uma entidade de negociação e mediação, o demônio de Laplace afirma um universo determinista, onde há um caminho único e linear das causas aos efeitos. Neste sentido, ele encontra-se muito mais próximo do deus cristão, em especial do Deus de Leibniz, que observa o desenrolar dos acontecimentos sem jamais intervir.

Há, contudo, um segundo demônio, que abre o caminho para o retorno ao *daemon* grego, o demônio de Maxwell, concebido como um experimento mental para pensar a relação

entre reversibilidade e entropia. Na termodinâmica, a entropia representa a tendência de um sistema fechado a atingir seu equilíbrio termodinâmico, isto é, o estágio de menor energia, diferenciação e organização. Dois receptáculos preenchidos com um mesmo gás, mas cuja temperatura é diferente em cada um, ao serem colocados em contato tenderão a possuir uma temperatura média, chegado a um equilíbrio. Ao nível microscópico, os átomos e moléculas, estatisticamente, vibrarão com a mesma velocidade, dado que a temperatura é a medida de velocidade média dos átomos em uma determinada substância (HOOKWAY, 2014).

A entropia marca, a princípio, um processo irreversível, criando uma assimetria temporal no sistema que permite diferenciar um antes e depois de acordo com o estado de organização do sistema, isto é, uma flecha do tempo (PRIGOGINE, 2011). O que Maxwell propõe é colocar um demônio entre os dois recipientes, uma minúscula criatura capaz de observar cada átomo individualmente, e decidir quais serão passados de um lado a outro. Em cada recipiente existem átomos mais rápidos e outros mais lentos. O demônio deixaria que apenas os átomos mais velozes passassem a um lado, enquanto os mais lentos passariam exclusivamente ao outro. Com isto, um recipiente ficaria progressivamente mais quente, enquanto o outro esfriaria. Em outras palavras, a entropia seria revertida ao impedir que ambos os recipientes chegassem a uma temperatura média, perdendo as suas especificidades térmicas (HOOKWAY, 2014).

O que Maxwell faz ao propor seu demônio é introduzir a noção de controle da termodinâmica. O demônio separa, divide e distribui, controla fluxos a partir de decisões que toma de modo inteligente. A reversão da entropia seria possível, pois o demônio, ao invés de considerar o sistema ao nível estatístico, observaria cada ato individualmente, permitindo um controle mais fino e preciso. A partir deste momento é possível conceber que em que cada superfície de contato entre estados distintos, isto é, em cada *interface*, há um demônio controlando e decidindo os destinos dos fluxos de calor. Tal como o *daemon* grego, o demônio de Maxwell habita os limiares, os cruzamentos e passagens, os interstícios e entremeios. De acordo com Serres (2007), o demônio é como o parasita, que se coloca no local mais privilegiado do sistema de modo a obter o máximo de rendimento. Com o menor dispêndio de energia, provoca as maiores modificações: ele *informa*. O demônio ocupa o lugar de Hermes, tal como o *daemon* que se faz passar por um deus. "Os anjos que passam, sejam deuses ou demônios, ocupam as encruzilhadas: nós de trocas, transformações, cortes, bifurcações de decisões, eixos, feixes em que o grande número vem de uma vez só. Os primórdios da política" (SERRES, 2007, p. 44).[cxiii] O demônio é este pequeno governador que, como um político. intercepta e redireciona os fluxos, centraliza os feixes em um ponto único de decisão,

e que, enfim, constitui uma interface.

Na década de 1960, o demônio de Maxwell transforma-se, novamente, em *daemon*. Inicialmente desenvolvido pelos membros do Projeto MAC, para o computador IBM 7094, e depois incorporado pelo mundo dos sistemas operacionais UNIX (incluindo atualmente OS X e Linux), o *daemon* é um programa executado em segundo plano pelo sistema operacional.

> Nossa utilização da palavra *daemon* foi inspirada pelo demônio de Maxwell na física e termodinâmica. (Minha formação é em Física). O demônio de Maxwell era um agente imaginário que ajudava a ordenar moléculas de diferentes velocidades e trabalhava incansavelmente no fundo. Começamos a extravagantemente utilizar a palavra *daemon* para designar processos de segundo plano que trabalhavam incansavelmente para realizar tarefas do sistema (CORBATO, 2002).[cxiv]

O *daemon*, como processo, é um programa que geralmente não é inicializado pelo usuário, mas pelo próprio sistema operacional. Ele trabalha em segundo plano, realizando as mais diversas tarefas rotineiras do sistema, como administrar os dados que entram e saem pelo *firewall*, monitorar o nível de energia da máquina, aguardar *inputs* do usuário, controlar outros processos, realizar *backups* automáticos e assim por diante. Boa parte destas tarefas não são visíveis ao usuário, mas sua existência garante o funcionamento do sistema (CHUN, 2008).

Um computador não é muito diferente do mundo grego antigo, tal como Sócrates e Proclo o descreveram, isto é, como um mundo repleto de *daemons* invisíveis que garantem o bom funcionamento das coisas através de uma negociação incessante entre as mais diversas ordens da natureza. No momento em que computadores tornam-se máquinas interativas, que funcionam em tempo real, os *daemons* constituem a tessitura que une e coordena a multiplicidade de exigências que a máquina tem de responder a cada instante. Um computador interativo é necessariamente múltiplo, e cada interação explícita é seguida por uma cadeia de interações, negociações e interfaces implícitas.

Cada vez mais cercamo-nos por máquinas universais em suas mais diversas formas: computadores, notebooks, *smartphones*, carros computadorizados. A máquina universal é a moradia desta nova estirpe de *daemons*. Há nisto uma certa retomada do mundo antigo, pois vivemos em um mundo que é permeado por entidades animadas, por objetos que agem e movem-se sozinhos, que falam conosco e que nos escutam, que nos observam e nos julgam. A internet das coisas, a promessa de que um dia todos os dispositivos de nossa casa serão inteligentes e poderão aprender como melhor nos servir, é a reconstituição do *Lar* romano, da moradia habitada e animada por espíritos. Estamos criando um mundo ao mesmo tempo encantando e assombrado, permeado de seres que ganham cada vez mais independência e autonomia, na mesma medida em que escapam a nosso controle ou compreensão.

Assim como os *daemons* antigos, o nosso mundo também depende da ação desses

seres invisíveis para funcionar, nos mais diversos níveis e dimensões, desde os fluxos monetários, até os fluxos de energia, água e pessoas. Quanto mais automatizamos o mundo, mais nos colocamos sob o julgo dos *daemons*, estes pequenos escravos que realizam as tarefas mais repetitivas. Os *daemons* são, também, nossos mensageiros, que transportam nossas mensagens, sons e imagens. Quando enviamos um e-mail que, por algum motivo não chega a seu destino, somos muitas vezes apresentados a um aviso dizendo que o *daemon* falhou em entregar nossa mensagem.

O retorno do *daemon* deve ser considerado no mesmo sentido do retorno do recalcado. Não é que agora os não-humanos, em especial os computadores e seus programas, voltam a magicamente a agir e determinar os rumos do mundo. Os não-humanos, em toda a extrema variedade de seres que este termo engloba, nunca deixaram de agir. O que o retorno do *daemon* marca não é o ressurgimento de um tipo específico de ser ativo, mas o fato de que o mundo nunca deixou de ser constituído por *daemons*, isto é, mediadores e intermediários, actantes e atores que, em suas redes e tramas, conectam, transmitem, traduzem, substituem, desconectam e negociam os agenciamentos que compõem o mundo. Os *daemons* nunca nos abandonaram. Muito pelo contrário, multiplicaram-se, assumiram mil formas e criaram inumeráveis modos de existência. A constituição moderna, na mesma medida em que excluía os híbridos, não parava de produzi-los sob seu discurso oficial (LATOUR, 1994). Do mesmo modo, o recalcado não para de agir, continua a estabelecer conexões e produzir composições indevidas no inconsciente. O retorno do *daemon* nada mais é do que exigência de pensarmos um modo de agência distribuída, de um pensamento simétrico que coloque humanos e não-humanos no mesmo nível ontológico e nos permita traçar os modos efetivos pelos quais as assimetrias são produzidas.

A exclusão dos *daemons*, a demonização cristã do exterior, tinha como função garantir a integridade do livre-arbítrio individual. Na medida em que o *daemon* era uma figura de controle, aquilo que nos pensava, diretamente, colocava em xeque a ideia de um *cogito* ou de um indivíduo completamente livre. A possessão demoníaca constituiria o momento mais terrível para o sujeito, que se via despojado de toda a sua liberdade, tendo de enfrentar uma força que o ultrapassava completamente. Retornando a seu sentido original, o *daemon* é o piloto da alma, isto é, o *kubernetes*, termo cuja corrupção latina transforma em governador e que nos leva, hoje, à cibernética, à ciência do controle e da comunicação que coloca humanos, máquinas e seres vivos sob o mesmo signo da informação.

O controle, tal como é concebido pela cibernética, nunca se dá de modo determinista, pois existe exclusivamente em meio à entropia. Apenas um ser imaginário e supernatural

como o demônio de Maxwell pode exercer o controle sobre todos os indivíduos de um composto. Na prática, tornando o demônio mundano, *daemon*, o controle só pode ser exercido em determinados pontos de uma população cuja natureza é necessariamente estatística. O que Prigogine (2011) mostra é que em sistemas instáveis a descrição estatística não pode ser reduzida a trajetórias individuais. Se quisermos pensar a subjetividade em um mundo povoado por *daemons*, se restituirmos o *genius* ao sujeito, será necessário repensar a noção de controle. Pois o controle só existe em rede, constituindo um sistema ou um circuito. Não é uma propriedade, algo que possa ser possuído. O controle só existe na medida em que circula, em que transmite e é transmitido, em meio a traduções, substituições e deslocamentos. Ou como Bateson (2000) coloca, em meio a diferenças que produzem diferenças.

A subjetividade não é a interioridade concebida como pura origem ou destino da ação, é antes o espaço, o meio de negociação entre forças heterogêneas. Nunca se age sozinho, a agência é sempre o resultado de um agenciamento que produz um híbrido. Quem corta a árvore, o homem, ou o machado? Sem o estabelecimento de uma aliança não há ação, que por sua vez distribui-se e circula entre todos os envolvidos. Quando o celular nos notifica de um compromisso marcado para a próxima hora, quem lembra, o humano ou o *daemon*? Pois a cada ciclo de seu processamento o *daemon* acessa a memória do dispositivo, ou acessa a rede, e verifica as pendências que lhe atribuímos.

Observando atentamente, os atos cognitivos que consideramos como mais íntimos e pessoais, como lembrar, crer e julgar, são constantemente delegados a algo exterior. O computador eletrônico é um dos exemplos mais claros e evidentes desta externalização de ações, já que os programas também julgam, como o GPS que decide e escolhe qual o melhor caminho a ser tomado, ou agem de acordo com uma determinada crença acerca de um estado de coisas, como a câmera que ajusta seu foco automaticamente por acreditar que o que está sendo visto pela lente é um rosto humano. Mas tal processo de exteriorização da cognição não começou com o computador, ele precede os próprios objetos técnicos. A primeira superfície de inscrição que utilizamos, a primeira memória auxiliar que adotamos foram outros cérebros (SLOTERDIJK, 2011). Na ausência de argila, papiro ou papel, confiamos a outrem algo que deve ser lembrado. As narrativas orais, especialmente as narrativas míticas, possuem uma estrutura própria, que facilita a sua memorização, registro e acesso por diversos cérebros (LÉVY, 1993).

A memória humana, contudo, não é algo muito confiável. Nada garante que a lembrança de outrem (ou a nossa própria) será fiel ao que lhe for transmitido, já que depende profundamente de um processo de interpretação. O papel, ou alguma superfície de inscrição

equivalente, oferece-nos um meio de registrar memórias de modo muito mais eficiente e fidedigno. A aparente passividade do papel constitui, justamente, a força de seu modo de agir. O papel age ao conservar o que nele é gravado. Temos a garantia de que, fora a ação de forças externas, o papel preservará fielmente o que nele é inscrito, enquanto a memória humana necessariamente modificará uma memória de modo a conservá-la. Novamente a questão: quem lembra, nós ou o papel?

Os agenciamentos humano-cérebro e humano-papel constituem subjetividades distintas. Não é o sujeito que se relaciona de modos distintos com algo que lhe é externo, ele é a *própria relação*. O ciborgue constitui a figura da junção mais extrema entre humano e não-humano, carne e máquina mesclados no mesmo corpo. Concebido desta forma, o ciborgue nada mais é do que o modelo último da interiorização, pois é o organismo que deve absorver nos limites de seu corpo a exterioridade, quase como um novo sujeito kantiano. Mas, seguindo Clark (2003), o que acontece se pensarmos o ciborgue para além dos limites da pele? Se considerarmos todos os agenciamentos em que nos compomos, todos os objetos e organismos com os quais agimos juntos e sem os quais não podemos pensar, então somos, desde sempre, *ciborgues natos*.

Se quisermos encontrar a origem do pensamento, a fonte do gênio, não devemos buscá-lo na interioridade do corpo ou do inconsciente, mas na "massa perdida" do social, isto é, em todos *daemons* que nos cercam, em outros humanos, coisas, seres vivos, deuses e forças (LATOUR, 1992). Os *daemons* que nos acompanham não são meras extensões ou próteses que prolongam nosso pensamento, meios de transmissão passivos de uma vontade originada puramente no indivíduo. A primeira condição é negativa: sem a presença do *daemon* o pensamento simplesmente não é possível. Para alguém que só aprende ao tomar notas, retirar-lhe os meios para anotar impede que o aprendizado aconteça. A segunda condição é positiva: o *daemon* modifica o modo como o pensamento acontece. Escrever no papel, com caneta, e escrever em um editor de texto com corretor ortográfico, que auto-completa o texto, com a possibilidade de copiar, colar e desfazer, constituem experiências e subjetividades distintas.

Se abrir o código nos leva a conceber um programa como o produto da contribuição dos mais variados participantes em toda a sua diversidade de interesses e objetivos, *abrir a mente*, nos leva a concebê-la como a composição heterogênea de agências. Com Turing, vimos que a cognição funciona como uma máquina desorganizada, sujeita a toda sorte de interferências que progressivamente transformam-se em interfaces. A interface é o domínio próprio dos *daemons*, constitui o limiar em que se dão as trocas e passagens, que modula e controla os fluxos criando a distinção dentro-fora. Para os cristãos, o demônios marcavam a

figura da alteridade radical, este Outro que não poderia ser absorvido nem compreendido a não ser em termos puramente negativos. Com o *daemon*-interface podemos conceber essa zona intersticial que constitui o ponto de contato com a diferença sem necessariamente reduzi-la a uma identidade ou negatividade. O sujeito e a cognição definem-se, então, nem pelo dentro, nem pelo fora, mas pelas inúmeras interfaces, por toda a população de *daemons* que mediam incessantemente passagens e fluxos, pelo modo como são *amplificados* e *multiplicados*.

Com uma exposição cuidadosa às interferências, a máquina desorganizada pode vir a se organizar como uma máquina universal. Um sujeito produzido de tal maneira não seria muito diferente de um robô no seu sentido mais usual e limitado. O que nos interessa não é pensar a mente como uma máquina universal, mas o modo como esta subjetividade distribuída e composta constitui-se *com* as máquinas universais. Que sujeito, que cognição é produzida em meio a esta nova estirpe de *daemons*? Como pensar a subjetividade descentrada em meio a máquinas universais? O mundo habitado por *daemons* é, de certo modo, o mundo do esquizo de Deleuze e Guattari (2010), do sujeito cujas vozes que escuta não são mais a manifestação do teatro inconsciente interiorizado e personalizado, mas são as mensagens, ordens, controles e inspirações de todos os Outros que o co-habitam em um mundo de máquinas e fluxos.

6.2 Universalidade, interface e assimetria

Deuses e homens não se misturam, pertencem a ordens de existência cuja distância os separa e torna incomunicáveis. Apolo, através da voz de Baquílides, diz a Admeto: "Tu és apenas um mortal; por isto teu espírito deve nutrir dois pensamentos ao mesmo tempo" (BLANCHOT, 2001, p. 139). O deus encarna a unidade, escapa à dualidade dos mortais, não precisa dividir-se para pensar. Admeto, por sua vez, perante a exigência divina e às limitações de sua finitude se vê forçado a transformar a dualidade do pensamento em diálogo, o uno que se desdobra em dois. O homem fala a outro homem na tentativa de dar conta deste pensamento que o transborda. É, também, sinal da *métis*, da astúcia, esse tipo de inteligência que permite ao mais fraco vencer o mais forte através da engenhosidade, do subterfúgio e da técnica. Na medida em que o humano não é uno como o deus, desdobra o pensamento e o coloca em movimento, criando uma ordem estranha à eternidade divina.

Os deuses, entretanto, não são dotados de menos *métis*. Zeus, especialmente, é o

senhor dos deuses não por ser o mais forte, cujo poder derivaria da pura violência ou coerção, mas porque é aquele possui uma *métis* inigualável. A titã Métis, figura menor no panteão grego, casa com Zeus e imediatamente é devorada por seu esposo, o que leva ao nascimento de Atena. O gesto de consumir completamente Métis, ato próprio a um deus, confere a Zeus o máximo de astúcia que lhe permite sempre estar à frente de deuses e mortais. A *métis* implica também uma *polýmetis*, pois não é nem una, nem separada, mas diversa.

> O indivíduo dotado de *métis*, seja deus ou homem, quando é confrontado com uma realidade múltipla, mutável, cujo poder ilimitado de polimorfismo torna quase inapreensível, só pode dominá-la, isto é, cercá-la no limite de uma forma única e fixa, sobre a qual ele a capturou, mostrando-se mais múltiplo, mais móvel, mais polivalente ainda que seu adversário (DÉTIENNE; VERNANT, 2008, p. 13).

Os deuses não param de intervir diretamente nos afazeres humanos, mas apenas na medida em que tomam uma forma humana, animal ou natural. É neste polimorfismo que encontramos sua capacidade de vencer as adversidades, constituindo uma multiplicidade. O poder de transformação torna o deus inapreensível e aterrorizante ao mortal, incapaz localizá-lo ou identificá-lo em meio a tantas máscaras. Por isso, o humano deve ser mais artificioso que o deus, deve vencê-lo por uma astúcia, emboscada ou engodo, obrigando-o a revelar-se como deus. A *métis* divina só pode ser vencida por outra *métis*, por essa potência do múltiplo e do ondulante.

Admeto responde à exigência divina multiplicando o pensamento. Amplifica-o e delega, faz com que circule, mude e transforme-se. À exigência da unidade responde com a mobilidade da *métis*. Deste modo, o pensamento circula entre os mortais, sem limitar-se a eles. Estendendo o pensamento aos *daemons*, o mortal pode comunicar-se com os deuses, na medida em que suas palavras, desejos e súplicas são mediadas pelos espíritos que compõem a cadeia intermediária entre as duas ordens, divina e mortal. Se o deus, em sua forma divina, é a pura alteridade que escapa completamente ao pensamento humano, como o diálogo é possível com os *daemons*, com estes seres que sendo ainda próximos já inserem um elemento de diferença e heterogeneidade na comunicação?

A comunicação se dá pelo estabelecimento de uma distinção, a criação de um espaço onde a relação entre sinal e ruído é alterada de modo a permitir que as mensagens predominem sobre a entropia. Todo ato de comunicação é similar ao trabalho do demônio de Maxwell, só é possível na medida em que vence o ruído do meio (SERRES, 2003). Mais precisamente, a comunicação ocorre entre um mínimo e um máximo de possibilidade. Uma mensagem tem de, primeiro, ser escolhida entre pelo menos duas mensagens possíveis. Na ausência de escolha, tudo o que pode ser transmitido é uma única mensagem, idêntica em

todas as circunstâncias, repetição monótona e homogênea: é a catatonia, ou o prisioneiro de guerra que pode apenas dizer em suas cartas que tudo está bem. Sem a existência de pelo menos duas possibilidades de mensagens não há informação, isto é, a operação pela qual uma distinção é realizada.

No outro extremo, encontramos um número excessivo de possibilidades, ao ponto de uma escolha significativa tornar-se impossível. O ruído é a impossibilidade de distinção, a constituição de um espaço homogêneo onde tudo ou se move rápido de mais, ou cujo movimento é imperceptível. O trabalho da comunicação consiste, então, em estabelecer fronteiras e delimitações que permitam ao sistema desdobrar uma complexidade dentro dos limites ótimos para a comunicação. Em uma situação ideal, a delimitação é constituída por dois interlocutores, um canal, a mensagem, e os meios para codificá-la e decodificá-la. O primeiro interlocutor codifica a mensagem, pela voz ou pela escrita, por exemplo, a transmite pelo canal, como o ar, microfone ou correio, que é então recebida e decodificada pelo segundo interlocutor, que escuta ou lê a mensagem.

Neste modelo, inúmeras coisas são pressupostas. Primeiro, que os interlocutores compartilham a mesma linguagem e os meios para codificar e decodificar mensagens, o que implica na identidade de uma estrutura subjetiva. A linguagem é o meio de transmissão entre um eu e um tu, isto é, um Eu desdobrado em dois sujeitos. A existência dos pronomes pessoais apontaria para uma função primordialmente comunicativa da linguagem, já que os pronomes não designam nenhum objeto pré-estabelecido, mas referem-se à posição daquele que fala. A identidade estrutural entre os sujeitos que falam estabelece um limite ao que pode ser significado naquele meio, enquanto a utilização da mesma linguagem cria um conjunto comum de referências e significados (DELEUZE; GUATTARI, 1995).

Nesta perspectiva, sem a articulação entre subjetividade e linguagem, isto é, sem uma identidade prévia, a comunicação seria inviável, aproximando-se de um comportamento aleatório, entrópico. O mundo do sinal favorece a simetria a reciprocidade, constitui o liame no qual mensagens vão e voltam, em que os sujeitos podem entender-se e produzem redundância. A redundância refere-se à capacidade de uma mensagem ser decodificada corretamente mesmo que haja alguma informação faltando (BATESON, 2000). Quanto maior a chance de ser decodificada, maior será sua redundância. Deste modo, a distinção entre sinal e ruído permite criar uma divisão entre dois meios, um redundante e outro aleatório, tal como o demônio de Maxwell separava os átomos mais rápidos dos mais lentos, tornando um recipiente mais quente e complexo, e outro mais frio e simples.

Este modo de comunicação segue o modelo do diálogo, desta relação estabelecida

entre dois sujeitos que ocupam posições equivalentes, e cuja linguagem permite que cheguem a acordos, entendimentos e referências não ambíguas. Concordância do significado, comunhão da subjetividade. Contudo, nossas sociedades não constituem espaços igualitários, bolhas sem ruído em que as mensagens circulam livremente em meio a um grande diálogo recíproco e simétrico. Quase toda comunicação é apoiada em algum tipo de assimetria ou desigualdade, o professor que fala e o aluno que escuta, o juiz e o réu, o técnico e o usuário, o policial e o cidadão, o homem e a mulher, o adulto e criança. O que encontramos aí não é a límpida transmissão de mensagens, a troca de diferentes opiniões que enriquecem o diálogo e o levam a um novo entendimento. Encontramos algo próximo da palavra do ditador, isto é, daquele que dita, que fala sem parar, sem ouvir, cujo rumor incessante e repetitivo preenche todo o espaço. "Toda palavra é comando, terror, sedução, ressentimento, adulação, empreendimento; toda palavra é violência – e querer ignorá-lo pretendendo dialogar é somar a hipocrisia liberal ao otimismo dialético, para o qual a guerra é apenas uma forma de diálogo" (BLANCHOT, 2001, p. 140).

Para Deleuze e Guattari (1995), a linguagem antes de ser significante ou subjetivante é composta por *palavras de ordens*. A linguagem ordena, não tanto no sentido de dar ordens explícitas, mas organizando, distribuindo, agenciando, comandando, seduzindo, aterrorizando. Criar uma distinção entre sinal e ruído, constituir um espaço livre da entropia é um ordenamento, cuja instauração muitas vezes implica na exclusão não apenas das fontes de ruído sonoro, mecânico ou elétrico, mas de todos aqueles que confundem, misturam e interrompem os fluxos de informação considerados desejáveis. A cor da pele, o gênero, a forma de falar, a postura, as roupas, tudo isto pode ser considerado como o ruído a ser ativamente excluído e separado do sinal.

Um espaço sem ruído, contudo, torna-se praticamente indistinguível da entropia. Todo sistema fechado, privado de um fluxo externo de informação ou energia tende a atingir um equilíbrio termodinâmico ou informacional, torna-se homogêneo. Entropia é a incapacidade de distinguir, separar, *diferir*, seja porque tudo acontece rápido demais, criando uma aparência de imobilidade onde nada pode ser diferenciado, seja porque nada se move realmente, caindo na mais completa estagnação e catatonia. A criação provém do ruído (BATESON, 2000; SERRES, 2007).

Para além da feliz concordância de significados e sujeitos encontramos a figura do Outro. A reciprocidade do diálogo baseia-se na equivalência estrutural de seus interlocutores, o "eu" que fala e o "tu" que escuta são ambos sujeitos que possuem um "eu" como modelo. Quando esta simetria é quebrada, a comunicação não se dá mais entre "eu" e "tu", mas entre

"eu" e Outro, figura que não pode ser reduzida à evidência do eu. Entre os dois interlocutores não há mais identidade, produz-se uma diferença de tal modo que não é mais possível afirmar se do outro lado há um sujeito. A mensagem não retorna. Se algo volta, não pode ser compreendido. Não há mais comunicação, existe apenas a relação com uma pura diferença que coloca em xeque todos os pressupostos da linguagem.

> Agora, o que está em jogo e pode entrar em relação, é tudo o que me separa do outro, quer dizer, o outro, na medida em que eu estou infinitamente separado dele, separação, fissura, intervalo que o deixa infinitamente fora de mim, mas também pretende fundar minha relação com ele sobre esta própria interrupção que é uma *interrupção de ser* – alteridade pela qual ele não é para mim, é preciso repeti-lo, nem um outro eu, nem uma outra existência, nem uma modalidade ou um momento de existência universal, nem uma sobre-existência, deus ou não-deus, mas o desconhecido em sua infinita distância (BLANCHOT, 2001, p. 134–135).

A relação com o outro é como uma anomalia, uma distorção espacial que produz não mais uma geometria plana da comunicação, mas um espaço curvado, não-euclidiano. O outro situa-se nos dois extremos da comunicação, no mínimo e no máximo de possibilidades que cercam a mensagem. A impossibilidade de redução a um sujeito, a um eu, deriva desta incapacidade de distinção própria ao ruído. Ainda que o outro se encontre no domínio do ruído, para além de todo sinal e significação, não devemos confundi-los. O puro ruído acaba se tornando homogêneo, enquanto a outro marca a mais completa alteridade: é o neutro, não o indiferenciado. Colocando em outros termos, com o ruído não há relação possível, somos levados por seu turbilhão sem ter a chance de parar e pensar, estabelecer fronteiras ou distinções. O outro aparece justamente quando estabelecemos uma relação com este ruído sem tentar reduzi-lo a uma forma compreensível. Esta relação de infinita estranheza é, como diz Blanchot (2001), impossível, a relação da não-relação, quando conseguimos ver que sob a aparente homogeneidade entrópica do ruído há uma complexidade inesgotável e incompreensível. O outro é a relação com o mais completo desconhecido *enquanto* desconhecido, onde nada é descoberto ou conhecido, mas onde o ruído pode ser expressado enquanto pura diferença.

Para o sujeito que se depara com o outro – que pode muito bem ser outrem, já que é impossível atribuir-lhe qualquer propriedade de gênero, ordem, grau ou número –, a relação é necessariamente paradoxal. Por um lado, o sujeito encontra-se despojado de qualquer possibilidade de ação, já que o outro não é nem objeto, nem sujeito, mas é móvel, efêmero ou incompreensível demais para se tornar alvo de qualquer ação. Por outro lado, o sujeito é compelido a agir, deve responder à interpelação desse outro que nada lhe diz. De algum modo deve agir no interior da mais extrema passividade. Mais precisamente, é o sujeito enquanto *eu* que se vê diante do paradoxo, pois o eu não pode agir sem uma intencionalidade, sem um

objeto que seja alvo de sua consciência ou ação. Todavia, algo age. Algo impessoal, tão estranho quanto o outro e que o eu não pode reconhecer como seu. O *daemon* é a resposta ao impossível.

A relação com o outro exige que o sujeito deixe de ser um sujeito, que não tome o eu como sua medida e imagem. Por mais extrema que pareça, a relação com outrem não constitui uma experiência mística, é ao invés disso uma *experiência-limite*. A experiência mística encontra-se, ainda, sob o domínio da unidade. O eu se perde, o sujeito se dilui e algo maior o ultrapassa, como uma gota de água que cai num oceano, ou como o ímpeto dionísiaco de Nietzsche. O sujeito perde sua unidade para fazer parte de uma unidade maior, como Deus, a Natureza ou o Universo. É uma experiência transformadora, da qual o sujeito retorna com um conhecimento do Todo, no qual pode ver, e fazer parte da Verdade. A experiência-limite, por sua vez, não acrescenta nada, não ilumina nem revela. O sujeito e o eu se perdem, mas não se reencontram em uma ordem maior ou perante um conhecimento absoluto, muito pelo contrário, deparam-se com a ausência de qualquer ordem. Transgredir o limite, ir para além de qualquer interdição, nos joga em um espaço onde não há nenhum limite, limiar ou delimitação. Nada pode ser apreendido ou conhecido. Para além do limite não há liberdade de toda repressão, a natureza revelada e a tranquilidade da verdade, mas puro e simples ruído. O sujeito retorna da experiência-limite sabendo *menos*. Não há iluminação ou revelação, o sujeito apenas se desconhece ainda mais, retorna incerto de seus próprios limites. A experiência-limite é, no final das contas, uma experiência *do* limite, da fragilidade de todas as bolhas de ordem que criamos para respirar e viver (BLANCHOT, 2007).

A experiência-limite e a relação com o outro não constituem necessariamente eventos extremos ou excepcionais. Para Blanchot, a diferença é constitutiva da própria linguagem, mesmo em seu uso mais banal. Assume-se que a palavra toma o lugar da coisa, seja ao nível da referência, do significado ou da imaginação. Pelo uso da linguagem podemos nos referir às coisas, que se transformam em objetos do pensamento, e passam a fazer parte de um mundo marcado pela utilidade e organização, tornam-se funcionais. Este uso, entretanto, é baseado em um efeito primário da linguagem: o objeto torna-se ausente. Antes de possuir uma referência a linguagem marca uma ausência. A palavra que designa a coisa não é a coisa, esta estando presente ou não. Toda palavra é acompanha por esta ausência fundamental, que é a própria fonte de toda ambiguidade da linguagem. Por mais que uma proposição possua um referente, podendo ser considerada como verdadeira na medida em que corresponde a um estado de coisas no mundo, nada impede que esta mesma proposição seja tomada em outros sentidos. A multiplicação do sentido provém da ausência da coisa, dado que em ultima

instância nenhuma palavra possui um referente verdadeiro ou definitivo, permitindo que deslize e seja retomada nos mais diversos usos e contextos.

A ambiguidade é o ruído, é o que aumenta a complexidade de um sistema de forma inesperada o faz agir de modo imprevisto, podendo produzir efeitos destrutivos ou criativos. Quando a comunicação é interrompida e o ruído entra no sistema a ponto de não reconhecermos mais nosso interlocutor é o momento em que o *daemon* aparece. Mais precisamente, tudo o que se situa entre os dois interlocutores, o canal, os meios de transmissão e codificação, é o que toma o primeiro plano quando o ruído invade o sistema. Pela interferência a transparência do *meio*, da *mídia*, é quebrada, e todos os pequenos demônios que lutavam incessantemente contra a entropia aparecem em toda a sua complexidade. Se não podemos mais distinguir o interlocutor na outra ponta do sistema de comunicação é porque este se misturou a uma multidão de *daemons* e não sabemos mais *quem ou o quê* fala e age. Às palavras juntam-se ordens, comandos e instruções dos *daemons*, que comunicam e ordenam entre si para garantir o bom funcionamento do sistema. Neste momento, os dois planos convergem, significação e subjetivação encontram-se em meio a palavras de ordem, o código-fonte mistura-se ao código de máquina.

O outro não é nem deus, nem não-deus, pois o deus é a unidade, e a alteridade nem afirma nem nega a unidade, pois encontra-se fora de seu domínio, positivo ou negativo, de pensamento. Vimos que os deuses são, também, figuras de multiplicidade, quando entram no domínio da *métis*. Os *daemons* se fazem necessários para estabelecer um canal de comunicação entre a unidade, o bom senso e, inversamente, entre o polimorfismo e mutação constante dos deuses. Neste trabalho os próprios *daemons* desaparecem, o mundo torna-se um lugar funcional, onde tudo tem seu lugar e faz sentido, incluindo os próprios deuses.

A máquina universal não é menos polimorfa, é a máquina mais dotada de *métis*, que pode se transformar em qualquer outra máquina. Quantas máquinas agem em uma máquina universal? A resposta depende, primeiro, de qual máquina universal estamos falando. Se considerarmos a máquina universal proposta por Turing, em sua versão monádica e não interativa, podemos afirmar que em qualquer momento existem apenas duas máquinas agindo: a máquina universal e a máquina executada. A situação logo se complexifica, pois podemos conceber uma máquina universal que executa outra máquina universal executando uma terceira máquina, criando a possibilidade de um encadeamento de máquinas infinito.

Tomando um computador e um sistema operacional modernos como uma máquina universal, a situação torna-se ainda mais complexa. Você está utilizando o computador, provavelmente com o navegador aberto, acessando algumas páginas, ouvindo música, com

um editor de texto também aberto. Em determinado momento, tudo parece ficar lento. A música chega a ser interrompida por instantes, os programas não respondem imediatamente, tudo fica mais pesado e arrastado. Ao abrir o gerenciador de tarefas ou monitor do sistema, para ver quais programas estão abertos e como os recursos estão sendo utilizados, além dos programas que você abriu é exibida uma longa lista de outros programas com nomes arcanos: são os *daemons*, os processos executados em segundo plano. Nesta lista os três programas visíveis misturam-se a dezenas de outros agentes invisíveis. O acesso a esta visão privilegiada e complexa do sistema não revela nenhum programa que esteja utilizando excessivamente o processador, ao mesmo tempo em que 100% do processador é utilizado. Uma força invisível continua a agir no sistema, consumindo seus recursos, interrompendo outros programas, interferindo, produzindo ruído.

O processo oculto pode ser um vírus, ou algum tipo de *malware*. Sua invisibilidade pode derivar tanto de sua execução conter comandos que o escondam de outros programas, quanto por ter modificado diretamente os programas que monitoram o sistema, ou ainda, por ter modificado o próprio sistema operacional em um nível tão profundo que é indistinguível de seu funcionamento normal. De três programas, passamos a um agrupamento de *daemons*, chegando uma invasão de vírus indetectáveis. A capacidade de executar programas simultaneamente, unida ao seu caráter interativo, aumenta exponencialmente a incerteza acerca de quem ou o quê age no sistema. Como mônada fechada, a máquina universal é uma multiplicidade, mas contida nos limites de sua operação, a ação provém apenas dela mesma e de seus programas. Quando a abrimos, não podemos mais ter essa certeza. O vírus age sozinho, ou está sendo executado remotamente, transformando o computador em uma máquina zumbi de uma *botnet*? A interação, através de suas interfaces, potencializa as interferências. O que é ruído, o que é sinal?

Neste momento o computador, a máquina universal, aparece em toda a sua complexidade e opacidade. A caixa-preta é aberta. Os *daemons* são revelados, o canal mistura-se à mensagem, e no meio disso tudo algo escapa, algo permanece desconhecido enquanto age. A máquina universal é a primeira a ser tornar invisível, pois nunca age sozinha, existe apenas na medida em que executa outro programa. Apenas nestes raros momentos de dúvida e perplexidade, em que nos questionamos acerca do que está realmente acontecendo nesta máquina tão misteriosa é que a máquina universal se revela enquanto tal. Percebemos que não estamos lidando apenas com uma máquina, mas com uma certa universalidade. O universal da máquina não deve ser considerado como uma abstração que aniquila a particularidade, ao contrário, como Feyerabend (2006) coloca, o universal funciona também

como um *mediador*. Ao invés de situar-se em um plano superior, como unidade de ordem maior, a universalidade constitui um meio comum, aquilo que é universal porque circula, permeia, encontra-se *entre* os particulares, os mediando e colocando em comunicação. A máquina universal é como um líquido, um gás, uma substância que preenche os espaços e os coloca em contato, é um *daemon* que agencia *daemons*.

As arquiteturas modernas dos computadores que operam com a noção de programa armazenado, isto é, de que programas e dados encontram-se igualmente na memória e são potencialmente intercambiáveis, ampliam e estendem a mediação das máquinas universais. É este nivelamento que permite a existência de vírus e programas que modificam o funcionamento de outros programas. A possibilidade desta modificação recursiva encontra-se, como já vimos, no código de máquina, no fato de que tudo se transforma em zeros e uns, em instruções. Mas se considerarmos um computador ou máquina universal apenas ao nível do código de máquina, observaríamos que o sistema tem como ponto de partida uma certa simetria ou reversibilidade fundamental.

Tendo acesso direto ao código de máquina, seja pelo *hardware* ou pela utilização de outro *software*, qualquer pedaço de código presente na memória do computador pode ser modificado. Qualquer mudança realizada pode ser, efetivamente, revertida. Basta pensar na clássica cena do *hacker* que invade um sistema e, ao sair, apaga todos os registros e vestígios de sua visita. Ainda que o sistema possua uma série de mecanismos de controle de seus fluxos, de registros de idas e vindas em seu domínio de funcionamento, a partir do momento em que o próprio sistema, ou a plataforma em que é executado encontram-se comprometidos, toda essa ordenação pode ser desfeita.

Na prática essa reversibilidade completa raramente é praticável. Quando atualizamos o sistema operacional, ou um programa aberto, é comum termos de reiniciá-lo para que as modificações sejam aplicadas. Quando utilizamos um programa ele carrega inúmeros dados na memória, como as modificações que foram realizadas em um documento e que podem ser desfeitas ou refeitas. A princípio, seria possível retraçar todas as modificações realizadas pelo programa até chegar a seu estado inicial, quando as atualizações seriam aplicadas. Na prática, contudo, isto é praticamente impossível, dada a complexidade dos dados envolvidos, e o fato de que durante o processo determinadas informações são apagadas ou sobrescritas, impedindo a sua reversibilidade. É muito mais simples fechar o programa e iniciá-lo novamente.

A possibilidade de reversibilidade provém daquilo que constitui o maior poder da máquina universal, sua recursividade. Cada ponto da máquina pode ser modificado, qualquer bit pode ser transformado, seja pela ação de outros bits, isto é, da própria máquina, seja por

uma intervenção direta no *hardware,* como a interferência de um raio cósmico. A reversibilidade nada mais é do que um encadeamento preciso de instruções ou intervenções que faça o sistema retornar a seu estado inicial, onde cada ação tomada até aquele momento é desfeita ou compensada. A reversibilidade constitui uma *simetria*, que em seu sentido matemático, deve ser entendida como uma *transformação*. Um corpo é simétrico quando, sofrendo uma modificação, como uma torção ou reflexão, ele permanece igual, ou aparentemente idêntico a seu estado interior. Imagine um quadrado que é rotacionado em 90º. Aparentemente, nada mudou. Caso a transformação se dê num ângulo de 45º, a simetria é quebrada, pois o quadrado torna-se diferente do que era anteriormente (STEWART; GOLUBITSKY, 1992). Deste modo, uma máquina reversível realizaria um determinado número de computações, retornando a seu estado inicial no término das operações, como se nada acontecera.

Uma máquina universal completamente reversível não seria muito útil. Seria, antes de mais nada, uma máquina incapaz de lembrar, pois a memória já constitui uma quebra de simetria. O acúmulo de memória faz com que, a cada ciclo, o sistema recomece a partir de um ponto diferente, de acordo com o que foi registrado. A tentativa de livrar-se da memória e retornar ao estado inicial apenas leva o sistema para um ponto ainda mais distante de seu equilíbrio. Mas o que realmente coloca em xeque a simetria é que para lembrar é necessário esquecer. Dado que o espaço de memória é sempre limitado, em determinado momento será preciso apagar ou sobrescrever uma memória para o registro de uma nova. Este apagamento impede que o sistema reconstitua com precisão o caminho tomado até aquele momento, criando bifurcações e passados possíveis para um mesmo presente. Quanto mais lembra, mais a máquina esquece, e o passado é progressivamente multiplicado.

As assimetrias da máquina universal são as suas interfaces. Toda comunicação parte de um desvio, de um *clinamen*, que nasce do caos homogêneo, do ruído do fundo, e introduz uma diferença. Pelo desvio um gradiente é criado, e com ele uma zona que segue um fluxo distinto de seu exterior, a entropia é redirecionada. O desvio é uma desordem que produz ordem. Antes que tudo retorne ao ruído, produzimos sentido, memória, mundo. A desordem que volta sobre si mesma (SERRES, 2003). As máquinas desorganizadas de Turing operam a partir desse princípio de desordem, vão se modificando aleatoriamente a partir de interferências até produzir organizações que permitam filtrar o ruído, estabelecendo interfaces. A máquina universal, por sua vez, já se encontra organizada. Basta ver a imensa tabela de instruções da máquina universal proposta por Turing, ou as suas mais variadas implementações em computadores. Cada máquina universal tem uma organização própria,

exige um código ou um modo de programação muito específico para funcionar. Como vimos, a máquina universal não executa *qualquer* código arbitrário, este deve estar de acordo com a linguagem ou dialeto que é capaz de compreender. Esta limitação já constitui uma primeira interface. A máquina universal executa qualquer máquina que seja capaz de interpretar.

Na medida em que acumula assimetrias, a máquina universal transforma-se em um labirinto. O *hardware*, junto à arquitetura que implementa, constitui o primeiro conjunto de interfaces que define o campo de possibilidades e estados operacionais do sistema. Cada arquitetura possui diferentes modos de acesso e programação, que nem sempre encontram equivalentes em outra arquitetura. Acima do *hardware*, encontramos o sistema operacional, segundo conjunto de interfaces. E assim por diante, um computador ou uma máquina universal organiza-se como encadeamentos de interfaces, criando canais e redes de comunicação. Portanto, a universalidade da máquina define-se pelo modo como as mediações são estabelecidas pelo sistema, como as assimetrias e passagens são criadas. Apesar disso, a máquina não se encontra livre de interferências. Seja pela inevitável ação da entropia, seja por ataques ou *bugs*, a máquina será acessada por outros meios que não as interfaces, afinal de contas, uma máquina existe no mundo e está sujeita a todas as suas forças e vicissitudes.

Como a universalidade de cada máquina depende do modo como é organizada, dos códigos que é capaz de interpretar, uma interferência produz os mais variados efeitos. Uma sequência de instruções aleatórias pode travar completamente a máquina, enquanto um ataque muito bem montado pode inserir instruções em pontos precisos permitindo que certas ações proibidas pela interface sejam executadas. Como no caso do vírus que produz processos ocultos, a interferência leva à produção de uma nova interface, cujo controle é inicialmente acessível apenas ao invasor e não ao usuário da máquina.

Não devemos esquecer que os *daemons* habitam as interfaces, ou mais precisamente, que trabalham incansavelmente para mantê-las operacionais, circulando, encadeando e tecendo redes nos labirintos de instruções. A comunicação na máquina, em especial na máquina universal, não está menos sujeita aos desníveis, assimetrias e estranhezas do que a linguagem humana. Não é, também, uma mera extensão de nossa linguagem, uma ferramenta simbólica, mas constitui um modo de funcionamento que não pode ser reduzido à ação humana. Um *daemon* que fala exclusivamente a linguagem binária precisa de outros, muitos outros *daemons* para traduzi-la e torná-la legível para nós.

A máquina universal constitui-se então como um meio, um espaço de mediação entre *daemons* e usuários, sendo que estes podem ser humanos, outras máquinas ou *daemons*. Admeto para dar conta do duplo pensamento, teve de se dividir em dois interlocutores em

diálogo. Mas diferente do deus que pensa a unidade, Admeto insere a diferença no pensamento, pois dialoga com outrem, com uma existência que lhe é completamente estranha e heterogênea. Pensar junto à máquina universal enquanto tal, nos coloca a mesma exigência de estranhamento, pois devemos pensar junto a uma multidão de *daemons*, dessas pequenas máquinas que mesmo quando passam no teste de Turing, nada garante que pensem como nós. Só, perante à máquina, o sujeito já se encontra em meio a uma multidão, a uma comunidade de humanos e não-humanos. Esta comunidade vai muito além da conjunção humano-máquina, pois a máquina, especialmente com a internet, é o ponto de convergência de outros humanos e máquinas. E na medida em que nos cercamos cada vez mais por estas máquinas, boa parte de nossas relações passam a ser mediadas por essa universalidade. É aí que encontraremos uma das possibilidades de constituição de uma comunidade assimétrica, pautada pela diferença e não pela identidade. Comunidade formada não por indivíduos, mas por híbridos, agenciamentos de corpos e *daemons* que constituem o que podemos chamar de uma cognição estendida, ou distribuída. Se abrirmos a máquina, cabe agora abrir a mente.

6.3 Abrir a mente

Na religião e no pensamento grego antigo muitas das propriedades e faculdades mentais que atribuímos ao indivíduo psicológico eram, também, divindades que compunham o panteão divino. A *Métis*, a astúcia, titã que Zeus devorou após seu casamento. Paixões e sentimentos, *Éros*, *Aidós* (pudor, humildade), *Phóbos* (medo), atitudes mentais, desvios e erros do espírito, *Pístis* (fé), *Áte* (engano), *Lýssa* (fúria). A própria *psyché* era um *daemon*, que posteriormente veio a ser internalizada. Esta divinização de características psicológicas permitia que se tornassem em objeto de culto, como foi o caso de *Mnemósyne*, a memória (VERNANT, 1990).

Esta possibilidade de exteriorização de características psicológicas não constituía uma projeção, pois esta noção já assume um sujeito que é primeiramente interiorizado e que só depois externaliza em outrem aquilo que é seu. Como coloca Sloterdijk (2011), a interiorização grega só começa mesmo com a figura do sábio, daquele que tem de pensar diferente dos outros e por isso deve se isolar com suas próprias ideias. O culto às propriedades psicológicas é antes uma condição de sociabilidade, um modo de pertencimento de sujeitos que não podem ser completamente definidos por aquilo que estaria na profundidade de sua alma. A vida psíquica era primeiro objeto de uma prática coletiva antes de ser propriedade

exclusiva do indivíduo.

Para Arendt (2007), um *daemon* acompanhava todo homem grego, o que formava como um segundo rosto e que só seria finalizado com sua morte. Este rosto, produto de suas ações entre os outros homens, é o que lhe sobreviveria, seria a imagem que persistiria e seria lembrada por outrem. O *daemon* é produzido como mediação entre homens, como aquilo que os ultrapassa e forma a base da tradição. Ainda que a *psyché* venha a ser internalizada, ela não se confunde com a vida psíquica do indivíduo, mas existe como um *daemon* que *reside* no interior. "A individualização desse *dáimon* unida a um ser humano particular que descobre nele o seu próprio destino não modifica o seu caráter de força misteriosa, estranha ao homem, de realidade presente no seio de toda natureza, no vento, nos animais, nas plantas, bem como no homem" (VERNANT, 1990, p. 160). Há no humano algo que vem de fora, que o habita, que não pode ser reduzido e que o ultrapassa, formando o laço social e constituindo a base de uma memória que transcende o corpo.

Enquanto a concepção grega antiga nos permite desconstruir a ideia de um sujeito completamente fechado em sua interioridade, devemos, contudo, abandonar os pressupostos que a tornam possível, isto é, a ideia de alma. Se há algo que existe antes e para além do sujeito, encarnando-se em objetos de culto, em *daemons* ou em deuses completamente formados é porque a alma existe independente do corpo, que nada mais é do que seu receptáculo temporário. A partir daí é possível estabelecer todas as dicotomias entre corpo e alma que culminarão na formulação cartesiana: material/imaterial, temporal/eterno, múltiplo/uno, passivo/ativo, espaço/tempo. O problema é como conceber a subjetividade, e a cognição, de modo que não sejam interiorizadas e nem dissociadas de sua relação com a corporeidade, para que a sua relação com a máquina universal possa ser pensada.

Ainda que a dicotomia corpo/mente seja um dos problemas centrais das diversas abordagens das ciências cognitivas, muitas vezes acaba sendo replicado em determinados modelos computacionais. Conceber a mente como uma máquina universal, na medida em que a universalidade é entendida como a generalidade independente de máquina, como vimos no capítulo 3, uma mente, enquanto *software*, pode ser implementada em qualquer máquina, isto é, em qualquer corpo. Este é o programa do que Searle (1980) chama de "Inteligência Artificial Forte", no qual uma mente implementada em um computador e em um corpo humano seriam equivalentes, ou no mínimo isomorfas, permitindo em ultima instância *uploads* e *downloads* de mentes.

Uma mente dissociada e abstraída de seu corpo acaba se reduzindo a um núcleo lógico, a uma simples máquina de resolução de problemas gerais (CLARK, 1997). O próprio

Turing mostrou o quanto é absurda a concepção da mente, e da subjetividade, reduzidas à máquina universal. Primeiro, porque um sujeito que fosse completamente uma máquina universal, cuja existência se reduzisse a resolver e computar os problemas que lhe são apresentados independentemente de qualquer contexto, viveria em um estado semi-vegetativo que dificilmente reconheceríamos como sendo "inteligente", existência completamente mecânica desprovida de intuição. Segundo, uma pessoa cujas partes fossem gradualmente substituídas (traduzidas) por equivalentes robóticos ou mecânicos acabaria se vendo privada de toda uma gama de experiências propriamente humanas, afastando-se de um pensamento que poderíamos chamar de "humano", apesar de que seria dotada de um modo próprio de pensamento e inteligência. Como Turing já havia dito, e muitos viriam afirmar depois, incluindo Searle (1980), uma inteligência propriamente humana só pode ser produzida em um corpo humano, ou um substituto que lhe seja indistinguível.

As limitações da máquina universal levam Turing (2004 [1948]) a conceber as máquinas desorganizadas, que existem em um meio de interferências que gradualmente serão transformadas em interfaces, sendo a mente uma delas. O corpo, ou mais precisamente o cérebro não é, entretanto, uma máquina completamente desorganizada, no sentido de que é produzido aleatoriamente. Há uma densidade própria ao corpo, que existe dentro de certos limites de variabilidade. A desorganização pode ser entendida então como as variações individuais dos corpos dentro destes limites de nossa fisiologia e arquitetura. Na perspectiva da cognição incorporada, nossa mente é simultaneamente específica ao nosso tipo de corpo e capaz de, em determinados momentos, agir de modo geral, aproximando-se de uma máquina universal.

Incorporar a mente é coextensivo a considerá-la como constituindo e fazendo parte de um mundo. Para Varela, Rosch e Thompson (1993) a cognição não se restringe nem à resolução de problemas nem ao processamento de informação, não é uma entidade separada do mundo que se contentaria em representá-lo com maior ou menor exatidão. A cognição constitui um modo de agir no mundo, é *enativa*, pois na medida em que age, produz (*enacts*) o mundo, assim como é produzida por ele. Nesta perspectiva não faz sentido falar em uma distinção *a priori* entre sujeito e objeto, mente e mundo. Ambos só existem em um processo de co-produção. As especifidades biológicas e ontológicas do corpo, o que constitui a sua densidade própria, existem apenas enquanto constituem e pertencem a um mundo. Por isso, os "universais" de nossa cognição ou biologia não podem ser dissociados de todas as diferenças que permeiam e constituem nosso modo ser e agir no mundo, pois estas diferenças que de fato incorporam e dão vazão aos universais, transformando-os e sendo transformadas no processo,

tais como as preensões de Whitehead (1978).

Todavia, um dos problemas que pode ser encontrado em diversas variações da perspectiva da mente incorporada, refere-se ao modo como objetos e actantes externos ao sujeito são considerados. Ainda que recoloque o sujeito em um modo de ação e existência ativo na construção do mundo, a divisão entre interior e exterior permanece, e tudo aquilo que se encontra fora dos limites do corpo não pertence nem participa da cognição, ou se o faz é como mera extensão, como uma prótese do sujeito. O caso mais exemplar seria a memória, cuja externalização constitui um dos procedimentos mais antigos da humanidade, como já vimos com os gregos. Esta externalização, contudo, não seria nada mais do que um registro passivo de informações, de sua inscrição em uma mídia externa, cuja decifração e operação caberiam completamente ao sujeito que o acessa. Os objetos são meros receptáculos passivos para as ações do sujeito, do mesmo modo que era o corpo para a perspectiva dualista. Se a cognição não for delimitada pelo corpo, corre-se o risco de recair na ideia de alma, na presença de algo místico que transcenda as barreiras da matéria e que faça toda a especificidade do indivíduo em seu corpo/mundo se perder (MALAFOURIS, 2013).

A hipótese da mente estendida propõe que a cognição deva ser entendida não como um processo intracranial, limitado ao nosso cérebro ou corpo, mas como um sistema que é recursivamente composto pela participação de agentes externos. Não é difícil nem estranho conceber, por exemplo, que a memória possa ser externalizada através do registro de seus conteúdos em determinados tipos de mídia, como o papel. Escrever ideias e pensamentos que não devem ser esquecidos, fotografar pessoas, situações, objetos ou gravar o áudio de uma conversa, são todas operações que buscam conservar, em maior ou menor grau, a memória como um *estado*, como um momento congelado que poderia ser posteriormente reativado por nosso aparato cognitivo interiorizado. Poderíamos pensar no computador humano que Turing (2004 [1936]) descreve, cujos estados mentais são substituídos por anotações em um caderno. Todavia, a hipótese da mente estendida propõe que além de estados mentais, externalizamos também *processos* mentais. Em outras palavras, o processo de pensamento não acontece apenas no interior de nossa cabeça, recolhendo informações passivas que estariam registradas em entidades para além do limite de nossa pele, o processamento é realizado *com* os agentes externos (CLARK; CHALMERS, 2008; MALAFOURIS, 2013).

Tome, por exemplo, a operação de multiplicar dois números grandes. Realizar uma operação tão complicada utilizando apenas nossa memória de curto prazo limitada seria uma tarefa extremamente difícil ou quase impossível para a maioria de nós, sujeita a uma grande quantidade de erros. A simples introdução de lápis e papel, e o conhecimento das regras

aritméticas de multiplicação numérica nos permite facilmente dominar uma tarefa que se encontraria muito além de nossas capacidades cognitivas (assim como a de outras espécies). O fato do papel permitir organizar um número sob o outro, alinhando as unidades, dezenas, centenas, etc, e, principalmente, conservando os números enquanto realizamos o conjunto de operações que constituem o cálculo, é o seu modo de *processar* tal informação, já que papel e lápis também agem sobre e organizam o conteúdo a ser trabalhado. Não são utilizados apenas como modo de armazenamento de estados, mas participam e constituem a sua organização e transformação, o processo.

Sem o acoplamento entre sujeito, lápis e papel o processo cognitivo de calcular simplesmente não seria possível. Mais precisamente, a cognição não se encontra nem do lado do corpo e do cérebro, nem do lado dos instrumentos, mas no circuito de causalidade recíproco formado por ambos. O cálculo não acontece exclusivamente como um conjunto de representações internalizadas que apenas respondem e modelam o mundo exterior, ao invés disso o cálculo acontece tanto na mente quanto no papel, nos neurônios, nos gestos, na motricidade e na materialidade própria ao papel e ao lápis, e com todas as convenções sociais e culturais que constituem nosso modo de fazer aritmética.

Nem todos os nossos processos cognitivos se dão de forma estendida, alguns são de fato internos ao limiar do corpo e do cérebro. Podemos calcular mentalmente sem recorrermos a nenhum recurso externo, como somar 15+15. Contudo, como Malafouris (2013) coloca, diversos processos cognitivos que consideramos privados, originaram-se de uma relação com a exterioridade. Primeiro aprendemos a contar e a calcular utilizando os dedos, marcas no papel ou objetos externos, para depois abstrairmos as regras básicas da aritmética. A capacidade de realizar operações aritméticas simples, como reconhecer automaticamente quantidades que variem de uma a quatro unidades, além de identificar variações em seu número, é algo comum ao homem e a outras espécies. Mas poder contar, realizar operações matemáticas complexas e derivar o conceito de número dependem, primeiro, de todo um processo que começa com o auxílio de objetos no mundo, para posteriormente se internalizar em menor ou maior medida.

Similar à mente estendida, a noção de "cognição distribuída" proposta por Hutchins (1995) concebe a cognição como constituindo um processo computacional que não precisa estar localizada necessariamente em um indivíduo. Em seu estudo, o autor busca compreender como um navio de guerra da marinha norte-americana é pilotado. Os processos cognitivos necessários para pilotá-lo não se encontram exclusivamente na mente do capitão que, sozinho, comandaria todo o navio. A cognição se distribui pelo próprio navio, incluindo seu

maquinário, instrumentos de navegação, motores, arquitetura, computadores, cartas de navegação, sendo constituída também por toda a tripulação, nos mais variados papéis que desempenham, assim como a própria hierarquia militar, que define como as funções serão distribuídas e os canais de comunicação e comando, além dos conflitos que possam ocorrer pela presença de soldados de diferentes membros do exército (marinha e aeronáutica). Pilotar um navio é um processo cognitivo que envolve atores humanos, máquinas, convenções sociais, e que surge como uma propriedade emergente do sistema como um todo, de modo ressonante com a ecologia da mente proposta por Bateson (2000).

As perspectivas da mente estendida e da cognição distribuída apresentam certas similaridades com o princípio de simetria da Teoria Ator-Rede, pois a cognição é concebida como um agenciamento de atores humanos e não-humanos que possuem, cada um a seu modo, a capacidade de agir, constituindo um "externalismo ativo" (MALAFOURIS, 2013). Um dos modos em que cognição e mundo podem ser acoplados se dá pelo isomorfismo. Clark e Chalmers (2008) contam a história de Inga e Otto, duas pessoas que tentam lembrar o endereço de um museu ao qual querem ir. Inga, que já conhece o museu, simplesmente se lembra do endereço, isto é, acessa uma memória interna que não estava consciente mas que lhe era diretamente acessível. Otto, por sua vez, é portador de Alzheimer, e por isso sempre carrega consigo um bloco de notas que contém um conjunto de informações que precisa lembrar regularmente ou que considera relevante, como o endereço do museu. Neste caso, para Otto, o ato de lembrar implica em consultar o seu bloco de notas. Em ambos os casos, os dois sujeitos lembram o endereço do museu, ao qual chegam sem problemas. Em um sentido estrito, os dois atos constituem o "mesmo" processo cognitivo: lembrar de um endereço. São isomorfos e até mesmo intercambiáveis.

A cognição não se encontra nem em Otto, nem em Inga, mas nos híbridos e sistemas que cada um constitui. No caso de Inga, o sistema é formado em um *loop* do sujeito sobre si mesmo, que acessa uma memória que lhe é interna. No caso de Otto o *loop* passa por um elemento externo. Agora considere que existe um duplo de Otto, a sua cópia exata que, contudo, possui um bloco de notas com o endereço incorreto do museu. Os dois Ottos, que possuem exatamente a mesma estrutura cerebral e corpórea, produzirão atos cognitivos diversos com resultados muito diferentes: um chegará com sucesso ao museu, enquanto outro ficará perdido. Cada Otto constituirá um sujeito e uma mente completamente distintas, de acordo com os acoplamentos e hibridizações em que se constituem. Do mesmo modo, entre Inga e Otto há diferenças consideráveis no modo como seus processos cognitivos ocorrem. Otto, por exemplo, não carrega seu bloco de notas enquanto toma banho, já Inga leva sua

memória para onde quer que vá. O bloco de notas pode ser danificado ou modificado por terceiros, mas Inga pode se embriagar, tornando temporariamente inacessíveis certas memórias, ou pode ter sua estrutura cerebral alterada por uma intervenção externa (CLARK; CHALMERS, 2008).

Estas diferenças, dentre muitas outras, revelam um dos problemas da hipótese da mente estendida. Em suas formulações iniciais, a hipótese parte de uma perspectiva computacional e em grande parte representacional. A cognição é vista como um tipo de processamento de informação, como um modo e representar o mundo, de manipular, gravar e ler informações, como no caso da memória, que não passa do acesso de um registro. A questão é como podemos conciliar, por um lado, o caráter enativo de cognição incorporada, que concebe a cognição por aquilo que *faz* não por aquilo que representa, enquanto por outro lado a compomos com todos os atores externos ao organismo que participam e constituem a cognição.

Primeiro, a composição da cognição por atores externos não pode se restringir a objetos, situações, pessoas ou agentes que desempenhem alguma função computacional. Devemos considerar qualquer acoplamento que produza uma ação cognitiva constituinte do mundo, e não apenas representativa (VARELA; ROSCH; THOMPSON, 1993). Deste modo, lembrar com ou sem um bloco de notas, constituem ações distintas, ainda que produzam efeitos computacionais similares. O processo de cada um está sujeito a afetos e possibilidades distintas. A memória de Inga não pode ser diretamente acessada por outrem, enquanto o bloco de Otto pode.

Para evitar que os atores que compõem a cognição não sejam considerados como meros instrumentos ou próteses sujeitos a uma agência completamente humana é necessário levar em conta toda a problemática da "cultura material" (MALAFOURIS, 2013; MILLER, 2013). Tal como a Teoria Ator-Rede propõe, a ação é sempre composta, é em direção a ela que um número indeterminado de atores converge (LATOUR, 2012). Os objetos, coisas, animais, bactérias, forças e divindades que compõem nosso mundo material constituem os agentes que tornam a cognição estendida possível ao agirem com e para além do sujeito. Como o navio descrito por Hutchins, a cognição distribuída sequer tem um sujeito humano como centro, encontra-se dispersa igualmente entre humanos e não-humanos. A cultura material implica em que "grande parte do que nos torna o que somos existe não por meio de nossa consciência ou de nosso corpo, mas como um ambiente exterior que nos habitua e incita" (MILLER, 2013, p. 79).

O que a materialidade coloca em jogo é a heterogeneidade dos atores e actantes que

participam dos agenciamentos cognitivos. Se tanto humanos e não-humanos, de acordo com o princípio de simetria da Teoria Ator-Rede, são agentes válidos, é porque agem de modo diferente. Sem essa diferença não haveria necessidade de tradução nem de estabelecimento de alianças. Esta diferença é o que Latour (1988) chama tanto de força quanto de fraqueza. Os actantes precisam aliar-se por serem fracos, ao mesmo tempo em que sendo fortes podem estabelecer mais alianças. A noção de "substituto" [*surrogate*] ou "situação substituta" [*surrogate situation*] proposta por Clark (2005, 2010) permite pensar esta relação de força e fraqueza entre atores.

> Por situação substituta [*surrogate situation*] entendo qualquer estrutura do mundo real que é utilizada para substituir, ou assumir o lugar, de algum aspecto ou situação alvo. Por situação alvo, entendo um evento ou estrutura do mundo real que seja atual, possível, ou pelo menos superficialmente possível, objeto último de meu empenho cognitivo (CLARK, 2005, p. 236). [cxv]

O modelo ou maquete de um prédio funciona como seu substituto. A tradução de um prédio para um objeto ordens de magnitude menor, não é uma representação fiel, constitui necessariamente uma simplificação e redução, mas é justamente isto que o torna útil e capaz de estabelecer alianças. Nem sempre o original é seu melhor modelo, tal como o mapa da província chinesa que, na narrativa de Borges (1978), possuiria o mesmo tamanho da província. Um mapa pode ser utilizado apenas na medida em que *modifica* e *desloca* o território ao qual se refere, criando um objeto que apenas por uma extrapolação pode corresponder a algo no mundo.

O modelo de um prédio pode ser diretamente manipulado, rotacionado. Podemos abri-lo e olhar no seu interior, ou podemos deslocá-lo, colocando-o em outros locais mais propícios à sua construção. Nada disto poderia ser feito com o prédio original, especialmente se não foi construído ainda. Inversamente, não podemos entrar nem habitar o modelo de um prédio. Todos esses afetos e possibilidades de conexão que constituem estes objetos dependem diretamente de sua materialidade. A cognição se produz ai como um jogo de alianças e hibridizações que opera substituições e mesclas. Para Serres (2015), toda substituição possui um ponto intermediário que constitui a junção paradoxal dos elementos que combina, cruzamento que nunca pode ser completamente elidido, e por isso nunca há uma representação pura que aconteça independente dos gestos e multiplicidade de combinações que a produzem como ação em meio a outras ações no mundo.

Assumindo a perspectiva da ecologia da mente de Bateson (2000), se a mente e a cognição funcionam como um ecossistema, torna-se evidente, primeiro, a heterogeneidade dos elementos que a compõem. Um ecossistema existe apenas pela interação entre diversas

espécies de organismos com o clima, a geologia, os fluxos hídricos e energéticos, assim como os canais de comunicação e transmissão de informação. A cognição se dá do mesmo modo. Assumir a sua heterogeneidade nos recoloca perante o problema da comunicação com Outrem. Se a cognição não pode ser mais restrita a um isomorfismo em que algo externo simplesmente replica um processo interno, devemos pensar como pode ser estabelecida uma relação nesta diferença. Mesmo no caso de Otto, em que o bloco de notas deve reparar uma função que lhe seria natural, encontramos a possibilidade de outros processos cognitivos que não seriam possíveis apenas com um corpo e um cérebro, como o acesso direto à memória por um terceiro.

Deste modo, quando um elemento da ecologia cognitiva é substituído, todo o sistema pode se alterar, do mesmo modo em que a inserção de uma nova espécie em um ecossistema altera toda a sua dinâmica de funcionamento. É neste processo de substituição que os *daemons* tornam-se visíveis. Mais precisamente, a visibilidade se dá no momento da substituição, pois uma vez que esta se encontre estabelecida os *daemons* voltam a se tornar invisíveis. O substituto é uma interface, uma mediação entre um ator e aquilo que é substituído e cuja efetividade depende, em parte, do desaparecimento daquilo que lhe é específico, deixando que por um determinado tempo o modelo assuma completamente o lugar daquilo que é modelado.

> Os objetos são importantes não porque sejam evidentes e fisicamente restrinjam ou habilitem, mas justo o contrário. Muitas vezes, é precisamente porque nós não *os* vemos. Quanto menos tivermos consciência deles, mais conseguem determinar nossas expectativas, estabelecendo o cenário e assegurando o comportamento apropriado, sem se submeter a questionamentos. Eles determinam o que ocorre à medida que estamos inconscientes da capacidade que têm de fazê-lo (MILLER, 2013, p. 78–79).

O modelo do prédio em construção nos permite falar e pensar sobre o prédio como se ele existisse de fato, servindo também como substrato de nossas ações e decisões. O efeito produzido seria bem diferente se falássemos do modelo como uma simples maquete e não como um substituto.

Talvez seja na religião que esta operação de substituição se torne mais evidente. Para Clark (2010) e Day (2004), grande parte de nossos modos de expressão religiosa constituem uma forma de cognição estendida. Nas mais variadas concepções religiosas, o mundo fala, seja através de espíritos, anjos, demônios, deuses, forças, mas cuja fala encarna-se em algum tipo de materialidade: em ídolos, estátuas, rituais, nos elementos, nas estrelas. Ou como Miller (2013) coloca, "o imaterial só pode se expressar pelo material" (p. 111). A materialidade não constitui uma redução, é antes o meio pelo qual as forças que nos excedem e que seriam

completamente incompreensíveis podem fazer parte do nosso mundo, com as quais possamos estabelecer relações de trocas e comunicação, ainda que a heterogeneidade e diferença constitutivas de tal relação nunca despareçam completamente. “Como esses deuses e reis conquistaram o mundo senão desviando o som e a fúria do múltiplo em direção a outro corpo que não o seu próprio? Com tudo isso sendo consolidado em objetos trabalhados?” (SERRES, 2015, p. 162).

Quando os gregos antigos tornam faculdades e processos mentais em divindades e objetos de culto, é este mesmo processo de externalização e substituição que observamos. A exigência imposta a Admeto, de possuir um duplo pensamento, leva ao diálogo e à externalização, à quebra da unidade divina e à produção da diferença por substituições e delegações. Quando uma estátua é animada por um deus e este fala por sua boca, mesmo que a voz provenha de um sacerdote escondido, a estátua substitui o deus, enquanto simultaneamente *é* o deus. A substituição, em um primeiro momento, implica uma troca ou uma equivalência: a palavra no lugar da coisa, a proposição e o referente, a mão e a ferramenta. São dois polos que permanecem distintos, pois se temos um é justamente porque não podemos ter o outro. Se sacrificamos o bode é para não termos de sacrificar a vítima humana. Serres (2015) propõe, ao invés disso, uma *lógica da substituição*, que toma a substituição não por seu começo ou fim, mas a captura em seu processo, quando as partes envolvidas encontram-se ainda misturadas. A estátua é uma mescla de objeto construído e divindade eterna e seu poder advém justamente dessa duplicidade. “A palavra 'substituição', tal como a palavra 'substância', literalmente significa o que fica sob a estátua, o que está se escondendo em seu vazio oco ou sob suas aparências acidentais” (SERRES, 2015, p. 160).[cxvi] Em certo sentido, toda estátua é uma esfinge, essa criatura quimérica que não é nem homem, nem animal, e que desafia aquele que passa.

A substituição aproxima-se do que Latour (2002) chama de fatiche, aquilo que é ao mesmo tempo construído e que existiu desde sempre, artefato cuja verdade provém justamente de seu caráter artificial. O problema do fatiche é de como podemos construir algo que adquire uma autonomia para além de seu criador, como a estátua que é tanto uma construção quanto uma divindade que sempre existiu, ou um fato científico que só se manifesta em condições artificiais muito precisas em um laboratório, mas que existe independentemente de tal ocasião. O que importa na substituição ou no fatiche não é que os sujeitos acreditem, erroneamente ou não, que uma coisa é outra. O importante é sua atitude, o que é efetivamente *feito* ao substituírem.

> Não se trata de pedra-fetiche, mas de fatiches [fe(i)tiches], esses seres deslocados,

> que nos permitem viver, isto é, passar continuamente da construção à autonomia sem jamais acreditar em uma outra. Graças aos fatiches [fe(i)tiches], construção e verdade permanecem sinônimos. Uma vez quebrados, tornam-se antônimos (LATOUR, 2002, p. 55).

É na trama subterrânea formada por fatiches e híbridos, por aquilo que se encontra sob a estátua, enfim, pelos *daemons*, que a vida se torna possível, nessa troca e mescla incessante que obedece a uma lógica de substituições.

Com a noção de substituição podemos estabelecer como se dá a relação entre o sujeito, cognição e máquina universal. A operação fundamental da máquina universal é a substituição. Na Introdução, falamos do video "Evolution of the Desk", que mostrava como nos últimos trinta anos todo um conjunto de objetos encontrados sobre uma mesa de escritório foram substituídos por aplicativos em um computador. Um editor de texto pode tomar o lugar de lápis e papel, ou uma máquina de escrever, chegando a simular vários detalhes de seu funcionamento e iconografia. Mas o editor de texto não é uma máquina de escrever, assim como nenhum dos aplicativos corresponde exatamente aos objetos de que toma o lugar.

A grande transformação produzida pela sua tradução ao domínio de uma máquina universal é a possibilidade de interconexão, seu pertencimento a um meio tornado comum pelo código de máquina, pelo sistema operacional ou por outros protocolos. Qualquer programa que seja executado em um sistema operacional não pode ser mais uma mônada, seu funcionamento depende de uma negociação constante com o sistema operacional e com outros programas. E são literalmente os *daemons* que estabelecem e mantêm esses canais de comunicação, as interfaces. Qualquer programa que precise do horário atual tem de consultar o *daemon* relógio, que trabalha incessantemente no *background* contando cada segundo desde 1970[36]. Os *daemons* operam a substituições e são eles próprios substituições, na medida em que agregam e combinam-se com outros *daemons*.

A interferência constitui outro modo de substituição. Quando um programa é colocado em meio a outros, é difícil ter certeza absoluta do que vai acontecer. Dependendo de quais forem os programas, um pode corromper ou modificar a memória de outro. Este mesmo problema encontra-se na base da Tese de Church-Turing, pois uma máquina universal não tem como saber se qualquer programa arbitrário terminará ou não, pois não pode eliminar *a priori* as interferências. Geralmente a interferência constitui *bugs* que tornam evidentes as substituições e os *daemons*. Mas nem toda interferência é um *bug* ou uma quebra do sistema, a grande maioria nos passa despercebida (basta olhar o arquivo de *log* de qualquer sistema

36 Na computação o "tempo Unix" é definido como o número de segundos que passaram desde 1 de janeiro de 1970.

operacional para ver a quantidade de erros que são registrados e ignorados).

O que realmente faz com que a máquina universal, em suas mais diversas encarnações, estabeleça uma relação singular com a cognição não se refere tanto a suas interfaces ou interferências, pois a substituição é um traço presente e essencial de praticamente qualquer objeto e situação que permeie nossa vida, é a própria possibilidade de polissemia e produção de diferença. A máquina universal nos faz viver em mundo habitado por um novo tipo de *daemon*. A cognição estendida, como vimos, não se apoia em objetos técnicos necessariamente. Acopla-se com pessoas, situações, fenômenos naturais e divindades. Um objeto externo ao corpo de um sujeito pode participar do processo de sua cognição sem ter de ser uma máquina de processamento de informação. Enquanto que, historicamente, grande parte de nossas mídias são estáticas ou fornecem modos limitados de ação, enquanto as mais ativas, como organismos e pessoas, facilmente escapam ao nosso controle. Já o *daemon*, define-se tanto por seu caráter ativo, quanto pelo controle.

Na interface, tal como o demônio de Maxwell, o *daemon* cumpre as tarefas necessárias para estabilizar e manter o sistema funcionando. É também nosso servo, ou até mesmo escravo, que está sempre pronto a executar as ordens que recebe. Esse caráter servil deriva, a princípio, de sua constituição e participação no código que, como vimos no capítulo 5, distingue-se de outras linguagens por ser performático em toda sua extensão. O código é uma sequência de ordens, é a palavra de ordem por excelência. Mas é justamente nessa servilidade que o código e *daemon* encontram sua autonomia. O código não espera. Seus efeitos são imediatos. Enquanto que para um sujeito fazer com que outro execute aquilo que ordena ou pede é necessário que seja estabelecida uma série de condições anteriores, que podem ser extremamente complexas, ao nível do código todo símbolo opera diretamente uma transformação.

Esta independência e imediaticidade do código permite que os *daemons* ajam de modo independente dos usuários. Claro que o código pode ser feito para esperar, pois a princípio a grande maioria do código escrito que existe foi criado por humanos para servir aos seus interesses. Mas um programa que a cada instante espere a intervenção de um usuário para continuar funcionando não traria muitas vantagens. As esperas constituem os momentos de interação com o sistema, são as interfaces que existem sobre um fundo de *daemons* que prestam pouca atenção a nossos desejos e anseios. Do mesmo modo, a existência dos *daemons* existe apenas em meio a uma rede sociotécnica extremamente complexa e por muitas vezes frágil. A imediaticidade e efetividade do código deve ser considerada nesse contexto.

Como fatiches, construímos *daemons* que ao nos obedecerem, adquirem sua autonomia. Progressivamente esses *daemons* executam funções cognitivas cada vez mais complexas, ao ponto de lhes delegarmos completamente certas funções. Se a cognição é uma propriedade das relações emergentes formadas pelo sistema cognitivo, os *daemons*, muitas vezes, pensam *para nós*, *sem nós*. Basta pensar em uma notificação no celular nos avisando de um evento que ocorrerá nas próximas horas e que pode ser de interesse, sendo que nunca ordenamos o *daemon* a executar tal operação, efeito de seu algoritmo trabalhando sobre nossas ações e interesses passados. Os *daemons* constituem um novo inconsciente, que não se encontra mais interiorizado, mas espalhado por aí e obedecendo a processos e regras decididas por seus acoplamentos. Os *daemons* operam um descentramento do sujeito. Ainda que a mente seja o produto de uma relação com outrem, poderia ser centralizada em um sujeito, sendo um raio ou aura que o ultrapassaria mas que se encontraria ancorada ainda. Com a máquina universal e seus *daemons*, a cognição e a mente descentralizam-se, não há mais um núcleo, e sim inúmeros centros, mônadas que pensam em seus mundos imediatos e que não precisam mais se agregar em uma figura total. O verdadeiro poder da associação entre humano e máquina universal não se encontra neste ou naquele *daemon* específico, mas na possibilidade de que o extremo polimorfismo da máquina universal seja conjugado de tal modo a não produzir um sujeito robótico e mecânico, mas um híbrido, uma *outra universalidade*. Encontramos aí a produção de um novo *corpo sem órgãos*.

6.4 Da máquina universal ao corpo sem órgãos

O problema colocado por este trabalho pode ser enunciado pela questão "o que pode uma máquina universal?" Vimos que, desde sua concepção com Turing, a relação com o limite já se colocava como fundamental. Tanto em uma perspectiva histórica quanto filosófica, vimos as inúmeras formas pelas quais a máquina universal foi construída e desconstruída, problematizada e confirmada. Surge como algo híbrido entre ideia lógica e máquina conceitual, transforma-se em autômatos, neurônios, linguagens. É retomada como tradução, como independência de máquinas, como generalidade ou especifidade, como liberdade. A máquina universal é, também, problematizada como modelo da mente, é recontextualizada em meio a máquinas desorganizadas, e no domínio do código, deixa de ser uma mônada fechada, abrindo-se a interações, interfaces e interferências. Por fim, torna-se a

morada de uma nova estirpe de *daemons*.

Em cada um destes domínios exploramos o que pode uma máquina universal. Encontramos inúmeros pontos de contato e distanciamento, mas de nenhum modo esgotamos todas a possibilidades. Claro, tal esgotamento é impossível, mas esta pesquisa nos permitiu ver que não existe A Máquina Universal, apenas máquinas, cada uma participando de um determinado conjunto de práticas, de agenciamentos. Mesmo na lógica, com suas pretensões e anseios por verdades universais platônicas, a máquina universal só existe como resultado de uma prática muito cuidadosa e específica. Colocar a questão do que pode uma máquina universal é indissociável de questionar também *como fazer uma máquina universal*. Colocando de outra maneira, a máquina universal é uma prática. É também um meio, uma universalidade como mediação. Mas onde, em que momento preciso, se houver um, podemos dizer que uma máquina é universal? Como distinguir a ação que produz a universalidade de qualquer outra ação?

Um ator pode se associar a outros, criando agregados que aumentam a capacidade de ação de ambos, como alguém que lê um livro. O livro não será lido a não ser que seja tomado por uma pessoa, do mesmo modo que esta não poderá ler sem associar-se ao livro. As associações constituem séries, onde se encadeiam fluxos de ações, em que ator age com outro, que por sua vez age com um terceiro e assim por diante (LATOUR, 1992). Este jogo de associações é o modo próprio de funcionamento das máquinas desejantes de Deleuze e Guattari (2010). Uma máquina desejante deve ser tomada no sentido mais geral possível, pois é qualquer ação, humana ou não-humana, que crie uma associação entre elementos heterogêneos, produzindo uma série de fluxos e cortes.

Em um certo sentido, as associações são como uma linha de montagem, onde uma peça é trabalhada por um sujeito, sendo levada a outro que a modifica novamente e assim por diante. A série de associações, por mais heterogênea que seja, acaba nos colocando em um caminho linear, limitando-nos a um conjunto restrito de estratégias. Como duas séries podem se cruzar? A substituição constitui a possibilidade do encontro, pois faz com que diversas séries coexistam sobre o mesmo plano (LATOUR, 1992). Ao substituirmos o livro por um *ebook*, a leitura se dá em meio a séries que podem nos oferecer *updates* ao livro, funções que aumentam, diminuem ou mudam a fonte do texto ou que nos permitem compartilhar as anotações feitas. O traçado deste plano é o momento em que a máquina universal se produz, é o modo como *fazemos um corpo sem órgãos*.

A coexistência de duas séries, o que chamamos de "disjunção inclusiva", não garante a sua conjunção. A existência de dois programas em uma mesma máquina não implica

necessariamente em uma interação entre ambos. Para que um corpo sem órgãos (CsO) possa ser produzido, "não definiremos algo por sua forma, nem por seus órgãos nem por suas funções, nem como substância, nem como sujeito" (DELEUZE, 2002, p. 132). Não definiremos a máquina universal por sua implementação, nem por sua arquitetura, funções, objetivos, usuários ou princípios lógico-matemáticos. Compreendermos a máquina universal a partir de seus *afetos*, de sua capacidade de afetar e ser afetada, de criar disjunções e possibilitar conjunções.

O CsO não é um conceito, é uma prática, e só existe na medida em que é feito. Para tal é necessário que desfaça os estratos que o transformam em corpo organizado. São três os estratos: organismo, significação e subjetividade. Há CsO para cada estrato: quando escapamos às determinações biológicas, quando quebramos a significação e quando desfazemos a subjetividade. A produção do CsO se dá em duas etapas. A primeira desfaz os estratos, enquanto a segunda faz passar intensidades. Em outras palavras, não basta apenas destruir o estrato, escapar à todas amarras. Isto pode muito bem nos destruir. Um corpo que tenha acabado com seus estratos cai em um vazio sem intensidade, nada mais acontece e não há mais para onde retornar. Devemos ter prudência, experimentar quais são nossos limites, onde podemos transpô-los, mas sempre mantendo um mínimo de organismo, de significação e subjetividade. Um ponto de retorno a partir do qual possamos nos lançar novamente (DELEUZE; GUATTARI, 1996).

A substituição é, também, um modo de experimentação (CLARK, 2005). Quando colocamos um rascunho, um esboço, na posição do produto final, podemos modificá-lo, testar novas possibilidades, experimentar sem o ônus ou os perigos de construir um produto final que será completamente imprevisível, puro ruído. A substituição, em um primeiro momento, desfaz o estrato. No lugar de um aplicativo, colocamos uma versão *crackeada* que permite utilizá-lo sem precisar ser um usuário legítimo. Pela substituição vamos trocando partes, vendo o que muda e o que se mantém. É uma operação topológica, de dobragem. A experimentação é a prática fundamental do CsO.

A máquina foi desmontada, e agora? Podemos tentar tanto reconstruí-la, quanto podemos utilizar as peças para compor uma nova máquina. É o problema de como produzir um devir, de como capturar elementos de outrem, criando um híbrido que não é nem homem nem não-homem. A hibridização que ocorre sobre o CsO é a passagem de uma intensidade, é a conjunção que produz um terceiro tipo de máquina: o sujeito descentrado. Tal como o quase-objeto/quase-sujeito de Serres (2007), o sujeito descentrado existe apenas enquanto circula, enquanto não se encontra nos polos sujeito/objeto de uma relação. Nesta passagem,

não podemos atribuir uma ideia, um pensamento ou uma ação a um determinado sujeito como autor. O pensamento circula, a subjetividade atravessa corpos, mentes, coisas, máquinas, animais e divindades. As substituições misturam e combinam afetos, entrelaçam os fluxos de máquinas desejantes produzindo novos fluxos que não são mais redutíveis à sua origem. Na máquina universal não encontramos o sujeito como usuário ou programador, como servo ou mestre, mas como essa agência distribuída que circula entre forças, que se mistura a *daemons*, ou torna-se indistinguível e imperceptível de processos, códigos, circuitos e instruções. Assim como as máquinas desorganizadas que imitam o humano para poderem ser *qualquer um* sob esta máscara, o sujeito que se compõe com a máquina universal transforma-se em uma multidão, nesse *todo mundo* que habita a máquina que é *qualquer máquina*.

Quando utilizamos um programa, o computador se resume a esta função, ou a um conjunto bem limitado de funções que a acompanham. Não há universalidade aí. Produzimo-la quando escapamos ao domínio da função, quando a subvertemos a novos usos, combinados com outros programas, quando criamos uma mediação, seja por interfaces ou por interferências. Todavia, transformar a conjunção em algo viável, fazer passar uma intensidade, não é algo simples. Por isso os vírus de computador são tão fascinantes. Têm de, primeiro, desconstruir o organismo, criando modos de serem executados, modificando o código de seu hospedeiro. É uma operação tão delicada que, em certos casos, se não for executada de forma precisa, a máquina poderá ser danificada, impedindo que o próprio vírus se propague e funcione.

O maior perigo do CsO é a produção de um corpo canceroso, o microfascismo. Criar um CsO implica em uma despersonalização, em abandonar uma subjetividade centrada no eu. Isto, contudo, ao invés de dar margem à produção de intensidades e diferenças, produz uma propagação do mesmo. Perde-se o eu individual para fazer parte de um grande coletivo homogêneo, do mesmo modo que o câncer se propaga pelo organismo substituindo as células por suas cópias (DELEUZE; GUATTARI, 1996). Um vírus de computador pode invadir o sistema e começar a se replicar incessantemente, ocupando toda a memória ou sobrescrevendo todos os arquivos com versões suas. Ou um erro muito mais simples: um processo ou função pode cair em uma recursão infinita, criando um ciclo vicioso que reproduz apenas a si mesmo.

O que pode uma máquina universal? Essa questão poderá ser respondida a cada nova experimentação, em cada agenciamento produzido, com suas novas formas de subjetivar, pensar e constituir um mundo. O problema ético colocado pela existência de máquinas universais é de como poderemos conviver, de como poderemos nos subjetivar, lutar, resistir e criar nesse mundo habitado por *daemons* que fogem tão rapidamente à nossa compreensão,

em que nunca sabemos com certeza quem ou o quê está no controle. Ainda que não possamos responder a estas questões, pelo menos podemos ter uma ideia, após este longo de trabalho, de alguns modos pelos quais podemos constituir universalidade como espaços de liberdade, experimentação e criação.

Referências

AGAMBEN, G. Genius. In: **Profanações**. São Paulo: Boitempo, 2007.

AGAR, J. **The Government Machine: A Revolutionary History of the Computer**. Cambridge: MIT Press, 2003.

AKERA, A. Voluntarism and the Fruits of Collaboration: The IBM User Group, Share. **Technology and Culture**, v. 42, n. 4, p. 710–736, 2001.

ARENDT, H. **A Condição Humana**. Rio de Janeiro: Forense Universitaria, 2007.

ARNOLD, V. **Catastrophe Theory**. Berlin: Springer-Verlag, 1992.

BACKUS, J. Programming in America in the 1950s: Some Personal Impressions. In: METROPOLIS, N.; HOWLETT, J.; ROTA, G.-C. (Eds.). . **A History of Computing in the Twentieth Century.** New York: Academic Press, 1980. p. 125–135.

BARLOW, J. P. **A Declaration of the Independence of Cyberspace**, 1996. Disponível em: <https://www.eff.org/cyberspace-independence>. Acesso em: 15 fev. 2016

BARLOW, J. P. **The first serious infowar is now engaged. The field of battle is WikiLeaks. You are the troops. #WikiLeaks**, 2010. Disponível em: <https://twitter.com/jpbarlow/status/10627544017534976>. Acesso em: 2 out. 2016

BARR, S. **Experiments in Topology**. New York: Thomas Y. Crowell Company, 1964.

BATESON, G. **Steps to an Ecology of Mind**. Chicago: University of Chicago Press, 2000.

BLANCHOT, M. **A Conversa Infinita: a palavra plural**. São Paulo: Escuta, 2001. v. 1

BLANCHOT, M. **A Conversa Infinita: a experiência limite**. São Paulo: Escuta, 2007. v. 2

BOOTH, A. S. Opening Address. In: GOODMAN, R. (Ed.). . **Annual Review in Automatic Programming I: Papers read at the Working Conference on Automatic Programming of Digital Computers held at Brighton, 1-3 April 1959**. [s.l.] Pergamon Press, 1960.

BORGES, J. L. Do rigor da ciência. In: **História universal da infâmia**. Porto Alegre: Globo, 1978.

BROOKS, F. P. **The Mythical Man-Month: essays on software engineering**. New York: Addison-Wesley, 1975.

BROOKS, F. P. No Silver Bullet — Essence and Accident in Software Engineering. In: KUGLER, H.-J. (Ed.). . **Proceedings of the IFIP Tenth World Computing Conference**. Amsterdam: Elsevier Science, 1986.

BULLYNCK, M.; DAYLIGHT, E.; DE MOL, L. Why Did Computer Science Make a Hero Out of Turing? **Communications of the ACM**, v. 58, n. 3, p. 37–39, 2015.

BUTLER, S. **Erewhon and Erewhon Revisited**. New York: Random House, 1955.

CALLON, M. Some Elements of a Sociology of Translation. In: **Power, Action and Belief: A New Sociology of Knowledge?** London: Routledge, 1986. p. 196–233.

CAMPBELL-KELLY, M. **From Airline Reservations to Sonic the Hedgehog: a history of the software industry**. Boston: MIT Press, 2003.

CARR III, J. W. Languages, Logic, Learning, and Computers. **Computers and Automation**, v. 7, p. 21–22, 25–26, 1958.

CASTELLS, M. **Redes de Indignação e Esperança**. Rio de Janeiro: Zahar, 2013.

CHOMSKY, N. On Certain Formal Properties of Grammars. **Information and Control**, v. 2, n. 2, p. 137–167, 1959.

CHUN, W. H. K. On "Sourcery", or Code as Fetish. **Configurations**, v. 16, n. 3, p. 299–324, 2008.

CHURCH, A. An unsolvable problem of elementary number theory. **American Journal of Mathematics**, v. 58, p. 345–363, 1936.

CHURCH, A. An unsolvable problem of elementary number theory. In: DAVIS, M. (Ed.). . **The Undecidable**. Mineola, NY: Dover, 1965a.

CHURCH, A. A note on the Entcheidungsproblem. In: DAVIS, M. (Ed.). . **The Undecidable**. Mineola, NY: Dover, 1965b.

CLARK, A. **Being There**. Cambridge: MIT Press, 1997.

CLARK, A. **Natural-Born Cyborgs**. Oxford: Oxford University Press, 2003.

CLARK, A. Beyond the Flesh: some lessons from a Mole Cricket. **Artificial Life**, v. 11, p. 233–244, 2005.

CLARK, A. Material Surrogacy and the Supernatural. In: MALAFOURIS, L.; RENFREW, C. (Eds.). . **The Cognitive Life of Things**. Cambridge: University of Cambridge, 2010.

CLARK, A.; CHALMERS, D. The Extended Mind. In: CLARK, A. (Ed.). . **Supersizing the Mind**. Oxford: Oxford University Press, 2008.

CLELAND, C. E. The concept of computability. **Theoretical Computer Science**, v. 317, n. 1–3, p. 209–225, 2004.

CLELAND, C. E. The Church–Turing Thesis. A Last Vestige of a Failed Mathematical Program. In: OLSZEWSKI, A.; WOLÉNSKI, J.; JANUSZ, R. (Eds.). . **Church's Thesis After 70 Years**. Frankfurt: Ontos Verlag, 2007.

COLEMAN, G. **Coding Freedom**. Princeton: Princeton University Press, 2012.

COLEMAN, G. **Hacker, Hoaxer, Whistleblower, Spy**. London: Verso, 2014.

COPELAND, J. (ED.). **The Essential Turing**. Oxford: Clarendon Press, 2004.

COPELAND, J. Turing's Thesis. In: OLSZEWSKI, A.; WOLÉNSKI, J.; JANUSZ, R. (Eds.). . **Church's Thesis After 70 Years**. Frankfurt: Ontos Verlag, 2007.

CORBATO, F. **The Origin of the Word Daemon**, 2002. Disponível em: <http://ei.cs.vt.edu/~history/Daemon.html>. Acesso em: 23 jan. 2016

DAVIS, M. Why Gödel Didn't Have Church's Thesis. **Information and Control**, v. 54, p. 3–24, 1982a.

DAVIS, M. **Computability and Unsolvability**. New York: Dover, 1982b.

DAVIS, M. **The Universal Computer**. New York: Norton, 2000.

DAY, M. Religion, off-line cognition and the extended mind. **Journal of Cognition and Culture**, v. 4, n. 1, p. 101–121, 2004.

DAYLIGHT, E. **The Dawn of Software Engineering: from Turing to Djikstra**. Belgium: Lonely Scholar, 2012.

DE MILLO, R. A.; LIPTON, R. J.; PERLIS, A. Social Processes and Proofs of Theorems and Programs. **Communications of the ACM**, v. 22, n. 5, p. 271–280, 1979.

DELEUZE, G. **A Dobra: Leibniz e o barroco**. Campinas: Papirus, 1991.

DELEUZE, G. Carta a um crítico severo. In: **Conversações**. São Paulo: Ed. 34, 1992.

DELEUZE, G. Bartebly, ou a fórmula. In: **Crítica e Clínica**. São Paulo: Ed. 34, 1997. p. 80–103.

DELEUZE, G. **Bergsonismo**. São Paulo: Ed. 34, 1999.

DELEUZE, G. **Espinosa: filosofia prática**. São Paulo: Escuta, 2002.

DELEUZE, G. **Diferença e Repetição**. Rio de Janeiro: Graal, 2006.

DELEUZE, G.; GUATTARI, F. **Mil Platôs: capitalismo e esquizofrenia**. Rio de Janeiro: Ed. 34, 1995. v. 2

DELEUZE, G.; GUATTARI, F. **Mil Platôs: capitalismo e esquizofrenia**. Rio de Janeiro: Ed. 34, 1996. v. 3

DELEUZE, G.; GUATTARI, F. **Mil Platôs: capitalismo e esquizofrenia**. São Paulo: Ed. 34, 1997. v. 4

DELEUZE, G.; GUATTARI, F. **O Anti-Édipo: capitalismo e esquizofrenia**. São Paulo: Ed. 34, 2010.

DELEUZE, G.; PARNET, C. **O Abecedário**, 1988. Disponível em: <http://stoa.usp.br/prodsubjeduc/files/262/1015/Abecedario+G.+Deleuze.pdf>

DELONG, H. **A Profile of Mathematical Logic**. Mineola, NY: Dover, 2004.

DENNET, D. Can Machines Think? In: **Brainchildren**. New York: Penguin Books, 1998.

DÉTIENNE, M.; VERNANT, J.-P. **Métis: as astúcias da inteligência**. São Paulo: Odysseus Editora, 2008.

DIJKSTRA, E. **ALGOL-60 Translation**: Mathematisch Centrum. Amsterdam: [s.n.].

DIJKSTRA, E. The Humble Programmer. **Communications of the ACM**, v. 15, n. 10, p. 859–866, 1972a.

DIJKSTRA, E. Notes on Structured Programming. In: DAHL, O.-J.; DIJKSTRA, E.; HOARE, C. A. R. (Eds.). . **Structured Programming**. London: Academic Press, 1972b.

DOCTOROW, C. Lockdown: The coming war on general-purpose computing. **Boing Boing**, 2011.

DORAN, R. Computer architecture and the ACE computers. In: COPELAND, J. (Ed.). . **Alan Turing's Electronic Brain**. Oxford: Oxford University Press, 2005.

DYSON, G. **Turing's Cathedral**. New York: Pantheon Books, 2012.

ENSMENGER, N. **The Computer Boys Take Over**. Cambridge: The MIT Press, 2010.

FETZER, J. Program verification: The very idea. **Communications of the ACM**, v. 31, n. 9, p. 1048–1063, 1988.

FEYERABEND, P. Os universais como tiranos e como mediadores. In: **A conquista da abundância**. São Leopoldo: Editora Unisinos, 2006.

FLECK, L. **Gênese e Desenvolvimento de um Fato Científico**. Belo Horizonte: Fabrefactum, 2010.

FREGE, G. O Pensamento. Uma investigação Lógica. In: **Investigações Lógicas**. Porto Alegre: EDI-PUCRS, 2002.

GALLOWAY, A. **Protocol**. Cambridge: The MIT Press, 2004.

GALLOWAY, A.; THACKER, E. **The Exploit**. Minneapolis: University of Minnesota Press, 2007.

GILL, S. The Philosophy of Programming. In: GOODMAN, R. (Ed.). . **Annual Review in Automatic Programming I: Papers read at the Working Conference on Automatic Programming of Digital Computers held at Brighton, 1-3 April 1959**. [s.l.] Pergamon Press, 1960.

GÖDEL, K. Remarks before the princeton bicentennial conference on problems in mathematics. In: DAVIS, M. (Ed.). . **The Undecidable**. Mineola, NY: Dover, 1965 [1946].

GOLDIN, D.; WEGNER, P. The Interactive Nature of Computing: Refuting the Strong Church–Turing Thesis. **Minds & Machines**, v. 18, p. 17–38, 2008.

GOLDSMITH, J.; WU, T. **Who Controls the Internet?** Oxford: Oxford University Press, 2006.

GORN, S. Standardized Programming Methods and Universal Coding. **Journal of the Association for Computing Machinery**, v. 4, n. 3, p. 254–273, 1957.

GREIMAS, A. J.; COURTÉS, J. **Dicionário de Semiótica**. São Paulo: Contexto, 2012.

HAIGH, T. Actually, Turing did not invent the computer. **Communications of the ACM**, v. 57, n. 1, p. 36–41, 2014.

HARDT, M. Prefácio - Junho Maldito. In: CAVA, B.; COCCO, G. (Eds.). . **Amanhã vai ser Maior**. São Paulo: Annablume, 2014.

HARDT, M.; NEGRI, A. **Multitude**. New York: Penguin Books, 2004.

HILBERT, D. Mathematical Problems. **Bulletin (New Series) of the American Mathematical Society**, v. 37, n. 4, p. 407–436, [1900] 2000.

HOARE, C. A. R. An axiomatic basis for computer programming. **Communications of the ACM**, v. 12, n. 10, p. 576–580, 1969.

HODGES, A. Did Church and Turing Have a Thesis about Machines? In: OLSZEWSKI, A.; WOLENSKI, J.; JANUSZ, R. (Eds.). . **Church's Thesis After 70 Years**. Frankfurt: Ontos Verlag, 2007.

HOFSTADTER, D. **Gödel, Escher, Bach**. New York: Basic Books, 1999.

HOOKWAY, B. **Interface**. Cambridge: The MIT Press, 2014.

HOPCROFT, J. E.; ULLMAN, J. D. **Formal Languages and their Relation to Automata**. London: Addison-Wesley, 1969.

HUTCHINS, E. **Cognition in the Wild**. Cambridge: MIT Press, 1995.

JASANOFF, S. The idiom of co-production. In: **States of Knowledge**. London: Routledge, 2004.

JORDAN, T. **Hacktivism and Cyberwars**. London: Routledge, 2004.

KLEENE, S. C. Recursive Predicates and Quantifiers. In: DAVIS, M. (Ed.). . **The Undecidable**. Mineola, NY: Dover, 1965 [1943].

KLEENE, S. C. **Introduction to Metamathematics**. New York: Elsevier, 1971 [1952].

KLEENE, S. C. General recursive functions of natural numbers. **Mathematische Annalen**, v. 112, p. 727–742, 1936.

KLEENE, S. C. Representation of Events in Nerve Nets and Finite Automata. In: SHANNON, C.; MCCARTHY, J. (Eds.). . **Automata Studies**. Princeton:

Princeton University Press, 1956.

KLEENE, S. C. **Mathematical Logic**. Mineola, NY: Dover, 1967.

KLEENE, S. C. Origins of recursive function theory. **Annals of the History of Computing**, v. 3, n. 1, p. 52–67, 1981.

KLINE, M. **Mathematics: the loss of certainity**. Oxford: Oxford University Press, 1982.

KLINE, R. R. Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence. **IEEE Annals of the History of Computing**, v. 33, n. 4, p. 5–16, 2011.

KNUTH, D. E.; TRABB PRADO, L. The early development of programming languages. In: METROPOLIS, N.; HOWLETT, J.; ROTA, G. C. (Eds.). . **A History of Computing in the Twentieth Century.** San Diego: Academic Press, 1980 [1976].

KRAJEWSKI, M. **Paper Machines**. Cambridge: The MIT Press, 2011.

LATOUR, B. **The Pasteurization of France**. Cambridge: Harvard University Press, 1988.

LATOUR, B. Where Are the Missing Masses? The Sociology of a Few Mundane Artifacts. In: BJIKER, W.; LAW, J. (Eds.). . **Shaping Technology/ Building Society**. Cambridge: The MIT Press, 1992.

LATOUR, B. **Jamais Fomos Modernos**. Rio de Janeiro: Ed. 34, 1994.

LATOUR, B. **Ciência em ação**. São Paulo: Editora Unesp, 2000.

LATOUR, B. **A Esperança de Pandora**. Bauru, SP: EDUSC, 2001.

LATOUR, B. **Reflexão sobre o culto moderno dos deuses fe(i)tiches**. Bauru, SP: EDUSC, 2002.

LATOUR, B. **Reagregando o Social**. Salvador: Edufba, Bauru:Edusc, 2012.

LATOUR, B. How Better to Register the Agency of Things. In: MATHESON, M. (Ed.). . **The Tanner Lectures on Human Values**. Salt Lake City: University of Utah Press, 2015. v. 34.

LAW, J. Technology and Heterogeneous Engineering: the case of Portuguese expansion. In: BIJKER, W. E.; HUGHES, T.; PINCH, T. (Eds.). . **The Social Construction of Technological Systems**. Cambridge: MIT Press, 1987.

LAW, J. **After Method**. New York: Routledge, 2004.

LEIBNIZ, G. W. Os princípios da filosofia ou A Monadologia. In: **Discurso de metafísica e outros textos**. São Paulo: Martins Fontes, 2004a. p. 129–150.

LEIBNIZ, G. W. Discurso de metafísica. In: **Discurso de metafísica e outros textos**. São Paulo: Martins Fontes, 2004b.

LEIBNIZ, G. W. **Ensaios de Teodiceia**. São Paulo: Estação Liberdade, 2013.

LÉVY, P. **As Tecnologias da Inteligência**. Rio de Janeiro: Ed. 34, 1993.

LEVY, S. **Hackers**. Cambridge: O'Reilly, 2010 [1984].

MACKENZIE, D. Negotiating Arithmetic, Constructing Proof: The Sociology of Mathematics and Information Technology. **Social Studies of Science**, v. 23, n. 1, p. 37–65, 1993.

MACKENZIE, D. **Computing, Risk, and Trust**. Cambridge: MIT Press, 2004.

MAHONEY, M. **Histories of Computing**. Cambridge: Harvard University Press, 2011.

MALAFOURIS, L. **How Things Shape the Mind**. Cambridge: The MIT Press, 2013.

MANNA, Z.; PNUELI, A. **The Temporal Logic of Reactive and Concurrent Systems**. New York: Springer-Verlag, 1991.

MCCARTHY, J. Towards a Mathematical Science of Computation. In: COLBURN, T.; FETZER, J.; RANKIN, T. (Eds.). . **Program Verification**. Dordrecht: Springer Netherlands, 1993 [1962].

MCCULLOCH, W. S.; PITTS, W. H. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, p. 115–133, 1943.

MILLER, D. **Trecos, Troços e Coisas**. Rio de Janeiro: Zahar, 2013.

MINSKY, M. **Computation: finite and infinite machines**. Englewood Cliffs: Prentice-Hall, 1967.

NELSON, V. **The Secret Life of Puppets**. Cambridge: Harvard University Press, 2001.

PERLIS, A.; SAMELSON, K. Preliminary report: international algebraic language. **Communications of the ACM**, v. 1, n. 12, p. 8–22, 1958.

PETZOLD, C. **The Annotated Turing**. Indianapolis: Wiley Publishing, 2008.

PLATÃO. O Banquete. In: **Diálogos**. Os Pensadores. São Paulo: Abril Cultural, 1983.

PLATÃO. **Fedro**. Lisboa: Guimarães Editores, 2000.

POST, E. Finite combinatory processes. Formulation I. **The Journal of Symbolic Logic**, v. 1, p. 103–105, 1936.

POST, E. Finite combinatory processes. Formulation I. In: **The Undecidable**. Mineola, NY: Dover, 1965a.

POST, E. Recursive Unsolvability of a Problem of Thue. In: DAVIS, M. (Ed.). . **The Undecidable**. Mineola, NY: Dover, 1965b.

PRIESTLEY, M. **A Science of Operations**. London: Springer, 2011.

PRIGOGINE, I. **O Fim das Certezas**. São Paulo: Editora Unesp, 2011.

PROCLO. **On The Theology of Plato**. London: A. J. Valpy, 1816. v. 2

RABIN, M. O. Probabilistic automata. **Information and Control**, v. 6, n. 3, p. 230–245,

1963.

RABIN, M. O.; SCOTT, D. Finite automata and their decision problems. **IBM Journal**, v. 3, p. 114–125, 1959.

RAYMOND, E. S. A brief history of hackerdom. In: **The Cathedral and the Bazaar**. Sebastopol: O'Reilly, 2001 [1992].

RAYMOND, E. S. The Cathedral and the Bazaar. In: **The Cathedral and the Bazaar**. Sebastopol: O'Reilly, 2001 [1998].

ROTMAN, B. **Ad Infinitum: the ghost in Turing's Machine**. Stanford: Stanford University Press, 1993.

ROTMAN, B. **Mathematics as Sign**. Stanford: Stanford University Press, 2000.

SALUS, P. H. **The Daemon, the GNU, and the Penguin: A History of Free and Open Source**. 2008.

SEARLE, J. Minds, Brains and Programs. **Behavioral and Brain Sciences**, v. 3, n. 3, p. 417–457, 1980.

SERRES, M. **Luzes: cinco entrevistas com Bruno Latour**. São Paulo: Unimarco Editora, 1999.

SERRES, M. **Hermes II. La Interferencia**. Buenos Aires: Editorial Almagesto, 2000.

SERRES, M. **O nascimento da física no texto de Lucrécio**. São Paulo: Editora Unesp, 2003.

SERRES, M. **The Parasite**. Minneapolis: University of Minnesota Press, 2007.

SERRES, M. **Statues**. London: Bloombury Academic, 2015.

SHAGRIR, O. Gödel on Turing on Computability. In: OLSZEWSKI, A.; WOLÉNSKI, J.; JANUSZ, R. (Eds.). . **Church's Thesis After 70 Years**. Frankfurt: Ontos Verlag, 2007.

SHANNON, C.; MCCARTHY, J. (EDS.). **Automata Studies**. Princeton: Princeton University Press, 1956.

SIMONDON, G. **El modo de existencia de los objetos técnicos**. Buenos Aires: Prometeo Libros, 2007.

SIMONDON, G. **La Individuación**. Buenos Aires: La Cebra y Editorial Cactus, 2009.

SLOTERDIJK, P. **Spheres, Volume I: Bubbles**. Los Angeles: Semiotext(e), 2011.

SLOTERDIJK, P. **Controversial Philosopher Says Man And Machine Will Fuse Into One Being**, 2 set. 2015. Disponível em: <http://www.huffingtonpost.com/entry/peter-sloterdijk-man-machine-interview_55e37927e4b0aec9f3539a06>. Acesso em: 19 out. 2015

SMITH, B. C. Limits of correctness in computers. **ACM SIGCAS Computers and Society**,

v. 14,15, n. 1,2,3,4, p. 18–26, 1985.

SMITH, B. C. Introduction. In: **Age of Significance**. [s.l: s.n.]. v. 1.

SMITH, D. **Essays on Deleuze**. Edinburgh: Edinburgh University Press, 2012.

SMITH, R. E. A Historical Overview of Computer Architecture. **Annals of the History of Computing**, v. 10, n. 4, p. 277–303, 1989.

STALLMAN, R. The GNU Manifesto. In: **Free Software, Free Society**. Boston: Free Software Foundation, 2010 [1985].

STALLMAN, R. Free Software definition. In: **Free Software, Free Society**. Boston: Free Software Foundation, 2010 [1996].

STEWART, I.; GOLUBITSKY, M. **Fearful Symmetry**. London: Penguin Books, 1992.

TANENBAUM, A. S. In defense of program testing or correctness proofs considered harmful. **ACM SIGPLAN Notices**, v. 11, n. 5, p. 64–68, 1976.

TARDE, G. A variação universal. In: **Monadologia e Sociologia**. São Paulo: Cosac Naify, 2007a.

TARDE, G. Monadologia e Sociologia. In: **Monadologia e Sociologia**. São Paulo: Cosac Naify, 2007b.

TARDE, G. Que és una sociedad. In: **Creencias, Deseos, Sociedades**. Buenos Aires: Cactus, 2011a. p. 35–67.

TARDE, G. La invención considerada motor de la evolución social. In: **Creencias, Deseos, Sociedades**. Buenos Aires: Cactus, 2011b.

TEDRE, M. **The Science of Computing: shaping a discipline**. New York: CRC Press, 2015.

THACKER, E. **In The Dust of This Planet**. Winchester, UK: Zero Books, 2011. v. 1

THOMSEN, D.; GEORGIEV, A. **Evolution of the Desk**, 2016. Disponível em: <http://bestreviews.com/best-home-office-desks#evolution-of-the-desk>. Acesso em: 2 dez. 2016

TRUDEAU, R. J. **Introduction to Graph Theory**. New York: Dover, 1993.

TURING, A. On Computable Numbers, with an Application to the Entscheidungsproblem. In: COPELAND, J. (Ed.). . **The Essential Turing**. Oxford: Clarendon Press, 2004 [1936].

TURING, A. Computing Machinery and Intelligence. In: COPELAND, J. (Ed.). . **The Essential Turing**. Oxford: Clarendon Press, 2004 [1950].

TURING, A. Systems of Logic Based on Ordinals. In: COPELAND, J. (Ed.). . **The Essential Turing**. Oxford: Clarendon Press, 2004 [1938].

TURING, A. Can Digital Computers Think? In: COPELAND, J. (Ed.). . **The Essential**

Turing. Oxford: Clarendon Press, 2004 [1951].

TURING, A. Intelligent Machinery. In: COPELAND, J. (Ed.). . **The Essential Turing**. Oxford: Clarendon Press, 2004 [1948].

TURING, A. et al. Can Automatic Calculating Machine be said to Think? In: COPELAND, J. (Ed.). . **The Essential Turing**. Oxford: Clarendon Press, 2004 [1952].

TURING, A. Lecture on the Automatic Computing Engine. In: COPELAND, J. (Ed.). . **The Essential Turing**. Oxford: Clarendon Press, 2004 [1947].

TURING, A. The Chemical Basis of Morphogenesis. In: COPELAND, J. (Ed.). . **The Essential Turing**. Oxford: Clarendon Press, 2004 [1952].

TURING, A. On computable numbers, with an application to the Entscheidungsproblem. **Proceedings of the London Mathematical Society**, v. 42, p. 230–265, 1936.

TURING, A. Computability and λ-definability. **Journal of Symbolic Logic**, v. 2, n. 4, p. 153–163, 1937.

TURKLE, S. **Life on the Screen**. New York: Simon & Schuster, 1995.

ULLMAN, E. The dumbing-down of programming Part Two: Returning to the source. Once knowledge disappears into code, how do we retrieve it? **Salon**, 1998.

VARELA, F.; ROSCH, E.; THOMPSON, E. **The Embodied Mind**. Cambridge: MIT Press, 1993.

VERNANT, J.-P. **Mito e Pensamento entre os Gregos**. Rio de Janeiro: Paz e Terra, 1990.

VON NEUMANN, J. **First draft of a report on the EDVAC**. [s.l.] Moore School of Engineering, University of Pennsylvania, 1945.

VON NEUMANN, J. Probabilistic logics and the synthesis of reliable organisms form unreliable components. In: SHANNON, C.; MCCARTHY, J. (Eds.). . **Automata Studies**. Princeton: Princeton University Press, 1956.

VON NEUMANN, J. The General and Logical Theory of Automata. In: TAUB, A. H. (Ed.). . **John Von Neumann: Collected Works**. Oxford: Pergamon Press, 1963. v. 5.

VON NEUMANN, J.; GOLDSTINE, H. H. On the Principles of Large Scale Computing Machines. In: BRÓDY, F.; VÁMOS, T. (Eds.). . **The Neumann Compendium**. Singapore: World Scientific Publishing, 1955.

WANG, H. A Variant to Turing's Theory of Computing Machines. **Journal of the Association for Computing Machinery**, v. 4, p. 63–92, 1957.

WEGNER, P. Why interaction is more powerful than algorithms. **Communications of the ACM**, v. 40, n. 5, p. 80–91, 1997.

WEINBERG, G. **The Psychology of Computer Programming**. New York: Van Nostrand

Reinhold Company, 1971.

WHITEHEAD, A. N. **Process and Reality**. New York: Free Press, 1978.

Anexos

Anexo A: tabela de instruções completa da máquina universal

Tabela abreviada [skeleton table] (TURING, 2004 [1936], p. 63–70)

Configuração		*Comportamento*	
config-m.	*símbolo*	*operações*	*config-m. final*
$\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$	**e**	*L*	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$
	não **e**	*L*	$\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$
$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	α		$\mathfrak{C}$
	não α	*R*	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$
	Nenhum	*R*	$\mathfrak{f}_2(\mathfrak{C}, \mathfrak{B}, \alpha)$
$\mathfrak{f}_2(\mathfrak{C}, \mathfrak{B}, \alpha)$	α		$\mathfrak{C}$
	não α	*R*	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$
	Nenhum	*R*	$\mathfrak{B}$
$\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{f}(\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$
$\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$		*E*	$\mathfrak{C}$
$\mathfrak{e}(\mathfrak{B}, \alpha)$			$\mathfrak{e}(\mathfrak{e}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$
$\mathfrak{pe}(\mathfrak{C}, \mathfrak{B})$			$\mathfrak{f}(\mathfrak{pe}_1(\mathfrak{C}, \beta), \mathfrak{C}, \mathbf{e})$
$\mathfrak{pe}_1(\mathfrak{C}, \mathfrak{B})$	Qualquer	*R, R*	$\mathfrak{pe}_1(\mathfrak{C}, \beta)$
	Nenhum	*P*β	$\mathfrak{C}$
$\mathfrak{l}(\mathfrak{C})$		*L*	$\mathfrak{C}$
$\mathfrak{r}(\mathfrak{C})$		*R*	$\mathfrak{C}$
$\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{f}(\mathfrak{l}(\mathfrak{C}), \mathfrak{B}, \alpha)$
$\mathfrak{f}''(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{f}(\mathfrak{r}(\mathfrak{C}), \mathfrak{B}, \alpha)$
$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{f}'(\mathfrak{c}_1(\mathfrak{C}), \mathfrak{B}, \alpha)$
$\mathfrak{c}_1(\mathfrak{C})$	β		$\mathfrak{pe}(\mathfrak{C}, \beta)$
$\mathfrak{ce}(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{c}(\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$
$\mathfrak{ce}(\mathfrak{B}, \alpha)$			$\mathfrak{ce}(\mathfrak{ce}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$
$\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, \beta)$			$\mathfrak{f}(\mathfrak{re}_1(\mathfrak{C}, \mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha)$
$\mathfrak{re}_1(\mathfrak{C}, \mathfrak{B}, \alpha, \beta)$		*E, P*β	$\mathfrak{C}$
$\mathfrak{re}(\mathfrak{B}, \alpha, \beta)$			$\mathfrak{re}(\mathfrak{re}(\mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha)$
$\mathfrak{cr}(\mathfrak{C}, \mathfrak{B}, \alpha)$			$\mathfrak{c}(\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, a), \mathfrak{B}, \alpha)$
$\mathfrak{cr}(\mathfrak{B}, \alpha)$			$\mathfrak{cr}(\mathfrak{cr}(\mathfrak{B}, \alpha), \mathfrak{re}(\mathfrak{B}, a, \alpha), \alpha)$

$\mathfrak{cp}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$			$\mathfrak{f}'(\mathfrak{cp}_1(\mathfrak{C}, \mathfrak{A}, \beta), \mathfrak{f}(\mathfrak{A}, \mathfrak{C}, \beta), \alpha)$
$\mathfrak{cp}_1(\mathfrak{C}, \mathfrak{A}, \beta)$	γ		$\mathfrak{f}'(\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma), \mathfrak{A}, \beta)$
$\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma)$	Γ não γ		$\mathfrak{C}$ $\mathfrak{A}$
$\mathfrak{q}(\mathfrak{C})$	Qualquer Nenhum	*R* *R*	$\mathfrak{q}(\mathfrak{C})$ $\mathfrak{q}_1(\mathfrak{C})$
$\mathfrak{q}_1(\mathfrak{C})$	Qualquer Nenhum	*R*	$\mathfrak{q}(\mathfrak{C})$ $\mathfrak{C}$
$\mathfrak{q}(\mathfrak{C}, \alpha)$			$\mathfrak{q}(\mathfrak{q}_1(\mathfrak{E}, \alpha))$
$\mathfrak{q}_1(\mathfrak{C}, \alpha)$	α não α	 *L*	$\mathfrak{C}$ $\mathfrak{q}(\mathfrak{E}, \alpha)$
$\mathfrak{pe}_2(\mathfrak{C}, \alpha, \beta)$			$\mathfrak{pe}(\mathfrak{pe}(\mathfrak{C}, \beta), \alpha)$
$\mathfrak{ce}_2(\mathfrak{B}, \alpha, \beta)$			$\mathfrak{ce}(\mathfrak{ce}(\mathfrak{B}, \beta), \alpha)$
$\mathfrak{ce}_3(\mathfrak{B}, \alpha, \beta, \gamma)$			$\mathfrak{ce}(\mathfrak{ce}_2(\mathfrak{B}, \beta, \gamma), \alpha)$
$\mathfrak{e}(\mathfrak{C})$	**e** não **e**	*R* *L*	$\mathfrak{e}_1(\mathfrak{C})$ $\mathfrak{e}(\mathfrak{C})$
$\mathfrak{e}_1(\mathfrak{C})$	Qualquer Nenhum	*R, E, R*	$\mathfrak{e}_1(\mathfrak{C})$ $\mathfrak{C}$
$\mathfrak{con}(\mathfrak{C}, \alpha)$	não A A	*R, R* *L, Pα, R*	$\mathfrak{con}(\mathfrak{C}, \alpha)$ $\mathfrak{con}_1(\mathfrak{C}, \alpha)$
$\mathfrak{con}_1(\mathfrak{C}, \alpha)$	A D	*R, Pα, R* *R, Pα, R*	$\mathfrak{con}_1(\mathfrak{C}, \alpha)$ $\mathfrak{con}_2(\mathfrak{C}, \alpha)$
$\mathfrak{con}_2(\mathfrak{C}, \alpha)$		*R, Pα, R* *R, R*	$\mathfrak{con}_2(\mathfrak{C}, \alpha)$ $\mathfrak{C}$

Tabela da Máquina Universal (TURING, 2004 [1936], p. 70–72)

Configuração		*Comportamento*	
config-m.	*símbolo*	*operações*	*config-m. final*
$\mathfrak{b}$			$\mathfrak{f}_1(\mathfrak{b}_1, \mathfrak{b}_1, ::)$
$\mathfrak{b}_1$		*R, R, P:, R, R, PD, R, R, PA*	$\mathfrak{anf}$
$\mathfrak{anf}$			$\mathfrak{g}(\mathfrak{anf}_1, :)$
$\mathfrak{anf}_1$			$\mathfrak{con}(\mathfrak{fom}, y)$
$\mathfrak{fom}$	; *z* não *z* não ;	*R, Pz, L* *L, L* *L*	$\mathfrak{con}(\mathfrak{fmp}, x)$ $\mathfrak{fom}$ $\mathfrak{fom}$

$\mathfrak{fmp}$			$\mathfrak{cpe}(\mathfrak{e}(\mathfrak{fom}, x,y), \mathfrak{sim}, x,y)$
$\mathfrak{sim}$			$\mathfrak{f}'(\mathfrak{sim}_{\mathrm{I}},\mathfrak{sim}_{\mathrm{I}}, z)$
$\mathfrak{sim}_{\mathrm{I}}$			$\mathfrak{con}(\mathfrak{sim}_{2},)$
$\mathfrak{sim}_{2}$	A		$\mathfrak{sim}_{3}$
	não A	*R, Pu, R, R, R*	$\mathfrak{sim}_{2}$
$\mathfrak{sim}_{3}$	não A	*L, Py*	$\mathfrak{e}(\mathfrak{mf}, z)$
	A	*L, Py, R, R, R*	$\mathfrak{sim}_{3}$
$\mathfrak{mf}$			$\mathfrak{g}(\mathfrak{mf}, :)$
$\mathfrak{mf}_{\mathrm{I}}$	não A	*R, R*	$\mathfrak{mf}_{\mathrm{I}}$
	A	*L, L, L, L*	$\mathfrak{mf}_{2}$
$\mathfrak{mf}_{2}$	C	*R, Px, L, L, L*	$\mathfrak{mf}_{2}$
	:		$\mathfrak{mf}_{4}$
	D	*R, Px, L, L, L*	$\mathfrak{mf}_{3}$
$\mathfrak{mf}_{3}$	não :	*R, Pv, L, L, L*	$\mathfrak{mf}_{3}$
	:		$\mathfrak{mf}_{4}$
$\mathfrak{mf}_{4}$			$\mathfrak{con}(\mathfrak{l}(\mathfrak{l}(\mathfrak{mf}_{5})),)$
$\mathfrak{mf}_{5}$	Qualquer	*R, Pw, R*	$\mathfrak{mf}_{4}$
	Nenhum	*P:*	$\mathfrak{sh}$
$\mathfrak{sh}$			$\mathfrak{f}(\mathfrak{sh}_{\mathrm{I}}, \mathfrak{inst}, u)$
$\mathfrak{sh}_{\mathrm{I}}$		*L, L, L*	$\mathfrak{sh}_{2}$
$\mathfrak{sh}_{2}$	D	*R, R, R, R*	$\mathfrak{sh}_{2}$
	não D		$\mathfrak{inst}$
$\mathfrak{Sh}_{3}$	C	*R, R*	$\mathfrak{sh}_{4}$
	não C		$\mathfrak{inst}$
$\mathfrak{sh}_{4}$	C	*R, R*	$\mathfrak{sh}_{5}$
	não C		$\mathfrak{pe}_{2}(\mathfrak{inst}, 0, :)$
$\mathfrak{sh}_{5}$	C		$\mathfrak{inst}$
	não C		$\mathfrak{pe}(\mathfrak{inst}, 1, :)$
$\mathfrak{inst}$			$\mathfrak{g}(\mathfrak{l}(\mathfrak{inst}_{\mathrm{I}}), u)$
$\mathfrak{inst}_{\mathrm{I}}$	α	*R, E*	$\mathfrak{inst}_{\mathrm{I}}(\alpha)$
$\mathfrak{inst}_{\mathrm{I}}$ *(L)*			$\mathfrak{ce}_{5}(\mathfrak{ob}, v, y, x, u, w)$
$\mathfrak{inst}_{\mathrm{I}}$ *(R)*			$\mathfrak{ce}_{5}(\mathfrak{ob}, v, y, x, u, w)$
$\mathfrak{inst}_{\mathrm{I}}$ *(N)*			$\mathfrak{ec}_{5}(\mathfrak{ob}, v, y, x, u, w)$
$\mathfrak{ob}$			$\mathfrak{e}(\mathfrak{anf})$

Anexo B: código-fonte e código de máquina

*Código-fonte, na linguagem **C**, que escreve a frase "Hello World"*

```
#include<stdio.h>

main()
{
    printf("Hello World");
}
```

Código-fonte em notação hexadecimal

```
0000000 6923 636e 756c 6564 733c 6474 6f69 682e
0000020 0a3e 6d0a 6961 286e 0a29 0a7b 2020 2020
0000040 7270 6e69 6674 2228 6548 6c6c 206f 6f57
0000060 6c72 2264 3b29 7d0a 000a
0000071
```

Código de máquina, em base hexadecimal, produzido ao compilar o código-fonte

```
0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000020 0001 003e 0001 0000 0000 0000 0000 0000
0000040 0000 0000 0000 0000 0130 0000 0000 0000
0000060 0000 0000 0040 0000 0000 0040 000d 000a
0000100 4855 e589 00bf 0000 b800 0000 0000 00e8
0000120 0000 5d00 48c3 6c65 6f6c 5720 726f 646c
0000140 0000 4347 3a43 2820 6255 6e75 7574 3420
0000160 382e 342e 322d 6275 6e75 7574 7e31 3431
0000200 302e 2934 3420 382e 342e 0000 0000 0000
0000220 0014 0000 0000 0000 7a01 0052 7801 0110
0000240 0c1b 0807 0190 0000 001c 0000 001c 0000
0000260 0000 0000 0015 0000 4100 100e 0286 0d43
0000300 5006 070c 0008 0000 2e00 7973 746d 6261
0000320 2e00 7473 7472 6261 2e00 6873 7473 7472
0000340 6261 2e00 6572 616c 742e 7865 0074 642e
0000360 7461 0061 622e 7373 2e00 6f72 6164 6174
0000400 2e00 6f63 6d6d 6e65 0074 6e2e 746f 2e65
0000420 4e47 2d55 7473 6361 006b 722e 6c65 2e61
0000440 6865 665f 6172 656d 0000 0000 0000 0000
0000460 0000 0000 0000 0000 0000 0000 0000 0000
*
0000560 0020 0000 0001 0000 0006 0000 0000 0000
0000600 0000 0000 0000 0000 0040 0000 0000 0000
0000620 0015 0000 0000 0000 0000 0000 0000 0000
0000640 0001 0000 0000 0000 0000 0000 0000 0000
0000660 001b 0000 0004 0000 0000 0000 0000 0000
0000700 0000 0000 0000 0000 0590 0000 0000 0000
0000720 0030 0000 0000 0000 000b 0000 0001 0000
0000740 0008 0000 0000 0000 0018 0000 0000 0000
0000760 0026 0000 0001 0000 0003 0000 0000 0000
0001000 0000 0000 0000 0000 0055 0000 0000 0000
0001020 0000 0000 0000 0000 0000 0000 0000 0000
0001040 0001 0000 0000 0000 0000 0000 0000 0000
0001060 002c 0000 0008 0000 0003 0000 0000 0000
0001100 0000 0000 0000 0000 0055 0000 0000 0000
0001120 0000 0000 0000 0000 0000 0000 0000 0000
0001140 0001 0000 0000 0000 0000 0000 0000 0000
0001160 0031 0000 0001 0000 0002 0000 0000 0000
0001200 0000 0000 0000 0000 0055 0000 0000 0000
0001220 000c 0000 0000 0000 0000 0000 0000 0000
0001240 0001 0000 0000 0000 0000 0000 0000 0000
0001260 0039 0000 0001 0000 0030 0000 0000 0000
0001300 0000 0000 0000 0000 0061 0000 0000 0000
0001320 002a 0000 0000 0000 0000 0000 0000 0000
0001340 0001 0000 0000 0000 0001 0000 0000 0000
0001360 0042 0000 0001 0000 0000 0000 0000 0000
0001400 0000 0000 0000 0000 008b 0000 0000 0000
0001420 0000 0000 0000 0000 0000 0000 0000 0000
0001440 0001 0000 0000 0000 0000 0000 0000 0000
0001460 0057 0000 0001 0000 0002 0000 0000 0000
0001500 0000 0000 0000 0000 0090 0000 0000 0000
0001520 0038 0000 0000 0000 0000 0000 0000 0000
0001540 0008 0000 0000 0000 0000 0000 0000 0000
0001560 0052 0000 0004 0000 0000 0000 0000 0000
0001600 0000 0000 0000 0000 05c0 0000 0000 0000
0001620 0018 0000 0000 0000 000b 0000 0008 0000
0001640 0008 0000 0000 0000 0018 0000 0000 0000
0001660 0011 0000 0003 0000 0000 0000 0000 0000
0001700 0000 0000 0000 0000 00c8 0000 0000 0000
0001720 0061 0000 0000 0000 0000 0000 0000 0000
0001740 0001 0000 0000 0000 0000 0000 0000 0000
0001760 0001 0000 0002 0000 0000 0000 0000 0000
0002000 0000 0000 0000 0000 0470 0000 0000 0000
0002020 0108 0000 0000 0000 000c 0000 0009 0000
0002040 0008 0000 0000 0000 0018 0000 0000 0000
0002060 0009 0000 0003 0000 0000 0000 0000 0000
0002100 0000 0000 0000 0000 0578 0000 0000 0000
0002120 0015 0000 0000 0000 0000 0000 0000 0000
0002140 0001 0000 0000 0000 0000 0000 0000 0000
0002160 0000 0000 0000 0000 0000 0000 0000 0000
0002200 0000 0000 0000 0000 0001 0000 0004 fff1
0002220 0000 0000 0000 0000 0000 0000 0000 0000
0002240 0000 0000 0003 0001 0000 0000 0000 0000
0002260 0000 0000 0000 0000 0000 0000 0003 0003
0002300 0000 0000 0000 0000 0000 0000 0000 0000
0002320 0000 0000 0003 0004 0000 0000 0000 0000
0002340 0000 0000 0000 0000 0000 0000 0003 0005
0002360 0000 0000 0000 0000 0000 0000 0000 0000
0002400 0000 0000 0003 0007 0000 0000 0000 0000
0002420 0000 0000 0000 0000 0000 0000 0003 0008
0002440 0000 0000 0000 0000 0000 0000 0000 0000
0002460 0000 0000 0003 0006 0000 0000 0000 0000
0002500 0000 0000 0000 0000 0009 0000 0012 0001
0002520 0000 0000 0000 0000 0015 0000 0000 0000
0002540 000e 0000 0010 0000 0000 0000 0000 0000
0002560 0000 0000 0000 0000 6800 6c65 6f6c 632e
0002600 6d00 6961 006e 7270 6e69 6674 0000 0000
0002620 0005 0000 0000 0000 000a 0000 0005 0000
0002640 0000 0000 0000 0000 000f 0000 0000 0000
0002660 0002 0000 000a 0000 fffc ffff ffff ffff
0002700 0020 0000 0000 0000 0002 0000 0002 0000
0002720 0000 0000 0000 0000
0002730
```

Notas

i The first serious infowar is now engaged. The field of battle is WikiLeaks. You are the troops. #WikiLeaks

ii Governments of the Industrial World, you weary giants of flesh and steel, I come from Cyberspace, the new home of Mind. On behalf of the future, I ask you of the past to leave us alone. You are not welcome among us. You have no sovereignty where we gather.

We have no elected government, nor are we likely to have one, so I address you with no greater authority than that with which liberty itself always speaks. I declare the global social space we are building to be naturally independent of the tyrannies you seek to impose on us. You have no moral right to rule us nor do you possess any methods of enforcement we have true reason to fear.

iii To translate is to displace: [...] But to translate is also to express in one's own language what others say and want, why they act in the way they do and how they associate with each other: it is to establish oneself as a spokesman.

iv Turing was not, in any literal sense, one of the builders of the new discipline. He was not involved with ACM or other early professional groups, did not found or edit any journal, and did not direct the dissertations of a large cohort of future computer scientists. He never built up a laboratory, set up a degree program, or won a major grant to develop research in the area. His name does not appear as the organizer of any of the early symposia for computing researchers, and by the time of his death his interests had already drifted away from the central concerns of the nascent discipline.

v Therefore, the good historical question to ask is not "Are stored-program computers universal Turing machines?" but "Why have electronic stored-program computers been cast as universal, as general-purpose machines?"

vi Consider a given countably infinite class of mathematical or logical questions, each of which calls for a "yes" or "no" answer.

Is there a method or procedure by which e can answer any question of the class in a finite number of steps?

In more detail, we inquire whether for the given class of questions a procedure can be described, or a set of rules or instructions listed, once and for all to serve as follows. If (*after* the procedure has been described) we select *any* questionmof the class, the procedure will then tell us how to perform sucessive steps, after a finite number of which we will have the answer to the question we selected. In performing the steps, we have only to follow the instructions mehcanically, like robots; no insight or ingenuity or invention is required of us. After any step, if we don't have de answer yet, the instructions together with te existing situation will tell us what to do next. The instructions will enable us to recognize when the steps come to an end, and to read off from the resulting situation the answer to the question, "yes" or "no".

vii The decision problem is solved when we know a procedure with a finite number of operations that

determines the validity or satisfiability of any given expression.... The decision problem must be considered the main problem of mathematical logic.

viii Thesis I. Every effectively calculable function (effectively decidable predicate) is general recursive.

ix According to my definition, a number is computable if its deciamal can be written down by a machine.

x the justification lies in the fact that the human memory is necessarily limited.

xi We may compare a man in the process of computing a real number to a machine which is only capable of a Wnite number of conditions q1 , q2 , . . . , qR which will be called ‘‘m-configurations’’. The machine is supplied with a ‘‘tape’’ (the analogue of paper) running through it, and divided into sections (called ‘‘squares’’) each capable of bearing a ‘‘symbol’’. At any moment there is just one square, say the r-th, bearing the symbol S(r) which is ‘‘in the machine’’. We may call this square the ‘‘scanned square’’. The symbol on the scanned square may be called the ‘‘scanned symbol’’. The ‘‘scanned symbol’’ is the only one of which the machine is, so to speak, ‘‘directly aware’’. However, by altering its m-configuration the machine can eVectively remember some of the symbols which it has ‘‘seen’’ (scanned) previously. The possible behaviour of the machine at any moment is determined by the m-configuration qn and the scanned symbol S(r). This pair qn , S(r) will be called the ‘‘configuration’’: thus the coniguration determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (i.e. bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left. In addition to any of these operations the m-configuration may be changed. Some of the symbols written down will form the sequence of Wgures which is the decimal of the real number which is being computed. The others are just rough notes to ‘‘assist the memory’’. It will only be these rough notes which will be liable to erasure.

xii copying down sequences of symbols, comparing sequences, erasing all symbols of a given form, etc.

xiii It is possible to invent a single machine which can be used to compute any computable sequence. If this machine U is supplied with a tape on the beginning of which is written the S.D of some computing machine M, then U will compute the same sequence as M. In this section I explain in outline the behaviour of the machine. The next section is devoted to giving the complete table for U.

xiv Let us suppose that there is such a process; that is to say, that we can invent a machine D which, when supplied with the S.D of any computing machine M will test this S.D and if M is circular will mark the S.D with the symbol ‘‘u’’ and if it is circle-free will mark it with ‘‘s’’.

Notas

xv "What are the possible processes which can be carried out in computing a number?''

xvi Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e. on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. […]

The behaviour of the computer at any moment is determined by the symbols which he is observing, and his ''state of mind'' at that moment. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite.

[…]

Let us imagine the operations performed by the computer to be split up into "simple operations" which are so elementary that it is not easy to imagine them further divided. Every such operation consists of some change of the physical system consisting of the computer and his tape.

[...]

The simple operations must therefore include:

(a) Changes of the symbol on one of the observed squares.

(b) Changes of one of the squares observed to another square within L squares of one of the previously observed squares.

It may be that some of these changes necessarily involve a change of state of mind. The most general single operation must therefore be taken to be one of the following:

(A) A possible change (a) of symbol together with a possible change of state of mind.

(B) A possible change (b) of observed squares, together with a possible change of state of mind.

xvii We suppose, as in I, that the computation is carried out on a tape; but we avoid introducing the "state of mind" by considering a more physical and definite counterpart of it. It is always possible for the computer to break oV from his work, to go away and forget all about it, and later to come back and go on with it. If he does this he must leave a note of instructions (written in some standard form) explaining how the work is to be continued. This note is the counterpart of the "state of mind". We will suppose that the computer works in such a desultory manner that he never does more than one step at a sitting. The note of instructions must enable him to carry out one step and write the next note. Thus the state of progress of the computation at any stage is completely determined by the note of instructions and the symbols on the tape.

xviii In the present formulation the symbol space is to consist of a two way infinite sequence of spaces or boxes, i. e., ordinally similar to the series of integers …, -3, -2, -1, 0, 1, 2, 3, … . The problem solver or worker is to

move and work in this symbol space, being capable of being in, and operating in but one box at time. And apart from the presence of the worker, a box is to admit of but two possible conditions, i. e., being empty or unmarked, and having a single mark in it, say a vertical stroke.

[…]

The worker is assumed to be capable of performing the following primitive acts:

(a) *Marking the box he is in (assumed empty),*

(b) *Erasing the mark in the box he is in (assumed marked),*

(c) *Moving to the box ons his right,*

(d) *Moving to the box ons his left,*

(e) *Determining wheter the box he is in, is or is not marked*

xix The model identifies mathematical reasoning in its entirety – proofs, justifications, validation, demonstrations, verification – with the carrying out of chains of imagined actions that detail the step-by-step realization of a certain kind of symbolically instituted, mentally experiencied narrative. Thus, unlike physical science where thought experiments are contrasted with real experiments and are seen as one ratiocinative and persuasional device among many, mathematics, as presented here, will be singularly and exclusively founded on them.

xx is made up of all those *written* texts (book,s papers, research announcements, blackboard lectures) which are presented according to very precise, completely unambiguous rules, - conventions and protocols controlling the permitted patterns of sign use with respect to the content of assertions and their logical validation, the status of conjectures and hypothesis, the giving of definitions, specifying of notations, exhbiting of objects, and so on. In the extreme form of laconism permitted by mathematics such texts consist of sequences of definition, theorem, proof, and nothing eles.

xxi drawing illustrative figures and diagrams; giving motivations; supplying cognate ideas; rendering intuitions, guiding principles, and underlying stories; suggesting applications; fixing the intended interpretations of formal and notational systems; making extra-mathematical conncetions – generally using natural language as well as a variety of explicational moves, tropes, and iconographical devices to convey all manner of mathematical senes, ranging from the level of technical content to the “point” of a proof or the “triviality” of a result or the “fruitfulness” of a ounter-example.

xxii a timeless voice from no one and from nowhere.

xxiiithe Agent, unlike the Subject, has no ability to imagine and can only respond to signs in their truncated, skeletonized form as signifiers devoid of intentional meaning. In other words, the Agent is conceived as an automaton, a wholly mechanical and formal proxy for the Subject.

Notas

xxivThis conviction of the solvability of every mathematical problem is a powerful incentive to the worker. We hear within us the perpetual call: There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no *ignorabimus*.

xxv A function is said to be "effectively calculable'" if its values can be found by some purely mechanical process. Although it is fairly easy to get an intuitive grasp of this idea, it is nevertheless desirable to have some more deWnite, mathematically expressible definition. Such a definition was first given by Gödel at Princeton in 1934 (Gödel [2], 26), following in part an unpublished suggestion of Herbrand, and has since been developed by Kleene [2]). These functions were described as "general recursive" by Gödel. We shall not be much concerned here with this particular definition. Another definition of effective calculability has been given by Church (Church [3], 356–358), who identifies it with λ-definability. The author has recently suggested a definition corresponding more closely to the intuitive idea (Turing [1], see also Post [1]). It was stated above that "a function is effectively calculable if its values can be found by some purely mechanical process". We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine. It is possible to give a mathematical description, in a certain normal form, of the structures of these machines. The development of these ideas leads to the author's definition of a computable function, and to an identification of computability with effective calculability.4 It is not difficult, though somewhat laborious, to prove that these three definitions are equivalent (Kleene [3], Turing [2]).

xxvi Tarski has stressed in his lecture (and I think justly) the great importance of the concept of general recursiveness (or Turing's computability). It seems to me that this importance is largely due to the fact that with this concept one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e., one not depending on the formalism chosen. 1 In all other cases treated previously, such as demonstrability or definability, one has been able to define them only relative to a given language, and for each individual language it is clear that the one thus obtained is not the one looked for. For the concept of computability, however, although it is merely a special kind of demonstrability or decidability, the situation is different.

xxvii The greatest improvement was made possible through the precise definition of the concept of finite procedure, which plays a decisive role in these results. There are several different ways of arriving at such a definition, which, however, all lead to exactly the same concept. The most satisfactory way, in my opinion, is that of reducing the concept of finite procedure to that of a machine with a finite number of parts, as has been done by the British mathematician Turing.

xxviii Perhaps the strongest argument in favor of Turing's thesis is the fact that, over the years, all other noteworthy attempts to give precise yet intuitively satisfactory definitions of "effective procedure" have turned out to be equivalent.

Notas

xxixEngineers avoided Turing's paper because it appeared entirely theoretical, and theoreticians avoided it because of the references to paper and tape machines.

xxx These will be defined by analogy with physical computers of a certain kind.

xxxi Turing's theory of computable functions antedated but has not much influenced the extensive actual construction of digital computers. These two aspects of theory and practice have been developed almost entirely independently of each other. The main reason is undoubtedly that logicians are interested in questions radically different from those with which the applied mathematicians and electrical engineers are primarily concerned. It cannot, however, fail to strike one as rather strange that often the same concepts are expressed by very different terms in the two developments. One is even inclined to ask whether a rapprochement might not produce some good effect. This paper will, it is believed, be of use to those who wish to compare and connect the two approaches.

xxxii That every net, if furnished with a tape, scanners connected to afferents, and suitable efferents to perform the necessary motor-operations, can compute only such numbers as can a Turing machine.

xxxiii A “black box” with a finite number of inputs and a finite nuber of outputs. […] The internal functioning of such a “black box” is equivalent to a prescription that specifies which outputs will be stimulated in response to the stimulation of any given combination of the inputs, and also the time of stimulation of these outputs.

xxxiv You can build an organ which can do anything that can be done, but you cannot build an organ which tell you whether it can be done.

xxxv Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an a priori upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing’s concept unrealistic.

In the last few years the idea of a finite automaton has appeared in the literature. These are machines having only a finite number of internal states that can be used for memory and computation. The restriction of finiteness appears to give a better approximation to the idea of a physical machine. Of course, such machines cannot do as much as Turing machines, but the advantage of being able to compute an arbitrary general recursive function is questionable, since very few of these functions come up in practical applications.

xxxvi A grammar can be regarded as a device that enumerates the sentences of a language. We study a sequence of restrictions that limit grammars first to Turing machines, [...] and finally to finite state IV[arkov sources (finite automata).

Notas

xxxvii Whatever generalization of the finite automaton concept we may want to consider, it ought to be based on some intuitive notion of machines. This is probably the best guide to fruitful generalizations and problems, and our intuition about the behavior of machines will then be helpful in conjecturing and proving theorems.

xxxviii Because that was a unique machine, he knew only too well that his programs had only local significance, and also because it was patently obvious that this machine would have a limited lifetime, he knew that very little of his work would have a lasting value.

xxxixTo perform a computation, the computer must be programmed; the instruction set is the only language in which this can be accomplished. Initially, each computer provided a unique set of instructions. For example, the first prototype of the Manchester Mark I provided subtraction as its only arithmetic operation. Typical computers provided three or four of the basic arithmetic operations. Programmers working with a computer lacking a needed operation would have to substitute a sequence of instructions to achieve the effect of the absent' instruction. Thus, a large, well designed instruction set often simplified the work of early computer programmers.

xl The lesson which digital computer users have been learning during the past ten years is that, oddly enough, the main use for these new information machines may not be as computers at all but rather as "symbol manipulators." It is as a symbol manipulator - evaluating and interpreting symbols from its own environment - that the digital computer may be able to perform its most spectacular jobs.

Before the machine user can make use: of this lesson, however, he must adopt an "outside-in view" of the digital computer. He must become much more problem oriented and much less equipment oriented. He must realize that it is the method of description of the problem that is important and not the actual physical hardware or even the logical structure into which the hardware is interconnected.

How does a user describe a problem? Generally he develops a correspondence or mapping between the elements and relationships of the real world and a set of symbols, that he, and later the machine, is to manipulate or act upon.

xli There is no one computer, but only computers. Even in the most general form of a finite Turing machine, the computer is a schema rather than a single device. What it can do depends on the finite state table and on the contents of the tape. In its most general, or universal, form it can do anything for which we can provide suitable instructions, that is, the table for a specific Turing machine. That is the source of its power: it is a protean machine. However, precisely because it can do anything, it can do nothing in and of itself. It does things only when we provide the programs that cause the universal machine to emulate particular machines of our design.

xlii No actant is so weak that it cannot enlist another. Then the two join together and become one for a third

actant, which they can therefore move more easily.

xliii it states that any computing machine which has the minimum proper number of instructions can simulate any other computing machine, however large the instruction repertoire of the latter. All forms of automatic programming are merely embodiments of this rather simple theorem and, although from time to time we may be in some doubt as to how FORTRAN, for example, differs from MATH-MATIC or the Ferranti AUTOCODE from FLOW-MATIC, it will perhaps make things rather easier to bear in mind that they are simple consequences of Turing's theorem.

xliv … together with a suitably designed program, can be equivalent to another computer which may not actually exist as a separate entity.

xlv I. The new language should be as close as possible to standard mathematical notation and be readable with little further explanation.
II. It should be possible to use it for the description of computing processes in publications.
III. The new language should be mechanically translatable into machine programs.

xlvi Specialization refers to language restrictions, static (compile time) solutions, and the exploitation of machine-specific facilities – in the inrest of efficiency. Generalization refers to general language constructs, dynamic (runtime) solutions, and machine-independent languagen design – in the interest of correctness and reliability.

xlvii Our solution is for “good computing machines”, where by “good” we want to mean that we are completely free to determine how teh computer should be used, this in contrast with machines for which considerations of efficiency force us in practice to a special manner of use, that is, force us to take account of specific properties and peculiarities of the machine.

xlviii COBOL cripples the mind.

xlix The programmer had to be a resourceful inventor to adapt his problem to the idiosyncrasies of the computer: He had to fit his program and data into a tiny store, and overcome bizarre difficulties in getting information in and out of it, all while using a limited and often peculiar set of instructions. He had to employ every trick he could think of to make a program run at a speed that would justify the large cost of running it. And he had to do all of this by his own ingenuity, for the only information he had was a problem and a machine manual. [...]
Programming in the early 1950s was a black art, a private arcane matter involving only a programmer, a problem, a computer, and perhaps a small library of subroutines and a primitive assembly program

Notas

l The principles behind the approach were essentially those that had proven so successful in traditional manufacturing: replaceable parts, simple and repetitive tasks, and a strict division of labor.

li Defined the program specifications, designed the program, coded it, tested it, and wrote the documentation.

lii That programming was an essentially creative undertaking; that individual programmers varied enormously in terms of style and productivity; and that current programming practices resembled craft more than they did science.

liii It just had not occurred to me that there was going to be any difficulty about getting programs working. And it was with somewhat of a shock that I realized that for the rest of my life I was going to spend a good deal of my time finding mistakes that I had made myself in programs.

liv As tong as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

lv Like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination.
[...]
The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be.

lvi Human beings are not accustomed to being perfect, and few areas of human activity demand it. Adjusting to the requirement for perfection is, I think, the most difficult part of learning to program.

lvii In a mathematical science, it is possible to deduce from the basic assumptions, the important properties of the entities treated by the science.

lviii Must study the various ways elements of data spaces are represented in the memory of the computer and how procedures are represented by computer programs.

lix Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the exact text of the program itself by means or purely deductive reasoning. Deductive reasoning involves the application of valid rules of inference to sets of valid axioms. It is therefore desirable and interesting to elucidate the axioms and rules of inference which underlie our reasoning about computer programs.

Notas

lx Program testing can be used to show the presence of bugs, but never to show their absence!

lxi When the correctness of a program, its compiler, and the hardware of the computer have all been established with mathematical certainty, it will be possible to place great reliance on the results of the program, and predict their properties with a confidence limited only by the reliability of the electronics.

lxii That spoken stage is the first filter for a proof. If it generates no excitement or belief among his friends, the wise mathematician reconsiders it. But if they fred it tolerably interesting and believable, he writes it up. After it has circulated in draft for a while, if it still seems plausible, he does a polished version and submits it for publication. If the referees also fred it attractive and convincing, it gets published so that it can be read by a wider audience. If enough members of that larger audience believe it and like it, then after a suitable cooling-off period the reviewing publications take a more leisurely look, to see whether the proof is really as pleasing as it first appeared and whether, on calm consideration, they really believe it.

And what happens to a proof when it is believed? The most immediate process is probably an internalization of the result. That is, the mathematician who reads and believes a proof will attempt to paraphrase it, to put it in his own terms, to fit it into his own personal view of mathematical knowledge. No two mathematicians are likely to internalize a mathematical concept in exactly the same way, so this process leads usually to multiple versions of the same theorem, each reinforcing belief, each adding to the feeling of the mathematical community that the original statement is likely to be true. The most compelling transformation that can take place is generalization. If, by the same social process that works on the original theorem, the generalized theorem comes to be believed, then the original statement gains greatly in plausibility.

lxiii When you show that a program meets its specifications, all you have done is to show that two formal descriptions, slightly different in character, are compatible . This is why I think it is somewhere between misleading and immoral for computer scientists to call this "correctness". What is called a proof of correctness is really a proof of the compatibility or consistency between two formal objects of an extremely similar sort: program and specification .

lxiv There is no such thing as an incorrect program.

lxv Despite the many differences in professional goals and theoretical orientation that existed between the vocational programmers and the academic computer scientists, the strength of their shared aesthetic values and craft traditions provided a basis for community solidarity

lxvi There were 17 customers with 701s and 18 different assembly programs.

lxvii And an anarchic horde of early microcomputer enthusiasts bent on taking computer power to the people.

Notas

lxviii Over the years, over nearly two decades, John Lions' Code and Commentary became the most copied work in computing. They carry the appropriate copyright notices and the restriction to licensees, but there was no way that Western Electric could stem their circulation.[...]

Why care? Because here we are in the mid-1970s with the users taking control and determining what to distribute where information was concerned. Luckily, Western Electric was no more successful at controlling information than Popes Paul V and Urban VIII were when Galileo wrote of heliocentricity. But note again: In the 1970s, you received Lions' work in hard copy, via airmail from Sydney, Australia.

lxix The Golden Rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. […]

So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free.

lxx By working on and using GNU rather than proprietary programs, we can be hospitable to everyone and obey the law. In addition, GNU serves as an example to inspire and a banner to rally others to join us in sharing. This can give us a feeling of harmony which is impossible if we use software that is not free. For about half the programmers I talk to, this is an important happiness that money cannot replace.

lxxi The GPL is a license that while built on top of copyright law, reverses traditional copyright principles. Instead of granting the owner the right to restrict copies, the owner of a copyright grants the users the right to copy and share programs. And the GPL goes further, projecting into future versions: it behaves like a legal firewall against the threat of future private enclosure. Future versions of a distributed software program licensed under the GNU GPL must remain under the same license, and hence can also be used, shared, modified, and distributed by other users. (This differs from putting software in the public domain, since material released this way can be subsequently incorporated into a new piece of work, which in turn can be copyrighted).

lxxii Individual wizards or small bands of mages working in splendid isolation.

lxxiii A great babbling bazaar of differing agendas and approaches [...] out of which a coherent and stable system could seemingly emerge only by a succession of miracles.

lxxiv Given enough eyeballs, all bugs are shallow.

lxxv As the association of unhelpful elements into self-sustaining networks that are, accordingly, able to resist dissociation.

Notas

lxxvi The proposition that the ways in which we know and represent the world (both nature and society) are inseparable from the ways in which we choose to live in it.

lxxvii As tong as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

lxxviii What will happen when a machine takes the part of A in this game?

lxxix We suppose, as in I, that the computation is carried out on a tape; but we avoid introducing the "state of mind" by considering a more physical and definite counterpart of it. It is always possible for the computer to break oV from his work, to go away and forget all about it, and later to come back and go on with it. If he does this he must leave a note of instructions (written in some standard form) explaining how the work is to be continued. This note is the counterpart of the "state of mind". We will suppose that the computer works in such a desultory manner that he never does more than one step at a sitting. The note of instructions must enable him to carry out one step and write the next note. Thus the state of progress of the computation at any stage is completely determined by the note of instructions and the symbols on the tape.

lxxx When we have decided what machine we wish to imitate we punch a description of it on the tape of the universal machine.

lxxxi Is probably Continuous Controlling, but is very similar to much discrete machinery.

lxxxii We might say that the clock enables us to introduce a discreteness into time, só that time can for some purposes be regarded as a succession of instants instead of as a continuous flow.

lxxxiii All of this suggests that the cortex of the infant is an unorganised machine, which can be organised by suitable interfering training. The organising might result in the modification of the machine into a universal machine or something like it. This would mean that the adult will obey orders given in appropriate language, even if they were very complicated; he would have no common sense, and would obey the most ridiculous orders unflinchingly. When all his orders had been fulfilled he would sink into a comatose state or perhaps obey some standing order, such as eating.

lxxxiv A large remnant of the random behaviour of infancy remains in the adult.

lxxxv Consists in making spontaneous judgments which are not the result of conscious trains of reasoning.

Notas

lxxxvi To convert a brain or machine into a universal machine is the extremest form of discipline. [...] But discipline is certainly not enough in itself to produce intelligence. That which is required in addition we call initiative. [...] Our task is to discover the nature of this residue as it occurs in man, and to try and copy it in machines.

lxxxvii Todo quiere la *cosa social*, al igual que la *cosa vital*, es propagarse, no organizarse. La organización solo es un medio cuyo fin es la propagación, la repetición *generativa* o *imitativa.*

lxxxviii It would be quite unfair to expect a machine straight from the factory to compete on equal terms with a university graduate. The graduate has had contact with human beings for twenty years or more. This contact has throughout that period been modifying his behaviour pattern. His teachers have been intentionally trying to modify it. At the end of the period a large number of standard routines will have been superimposed on the original pattern of his brain. These routines will be known to the community as a whole. He is then in a position to try out new combinations of these routines, to make slight variations on them, and to apply them in new ways.

lxxxixCertainly the machine can only do what we do order it to perform, anything else would be a mechanical fault. But there is no need to suppose that, when we give it its orders we know what we are doing, what the consequences of these orders are going to be.

xc I don't want to give a definition of thinking, but if I had to I should probably be unable to say anything more about it than that it was a sort of buzzing that went on inside my head. But I don't really see that we need to agree on a definition at all.

xci From this point of view one might be tempted to deWne thinking as consisting of 'those mental processes that we don't understand'. If this is right then to make a thinking machine is to make one which does interesting things without our really understanding quite how it is done.

xcii One way of setting about our task of building a 'thinking machine' would be to take a man as a whole and to try to replace all the parts of him by machinery. He would include television cameras, microphones, loudspeakers, wheels and 'handling servo-mechanisms' as well as some sort of 'electronic brain'. This would of course be a tremendous undertaking. [...] In order that the machine should have a chance of Wnding things out for itself it should be allowed to roam the countryside, and the danger to the ordinary citizen would be serious. Moreover [...], the creature would still have no contact with food, sex, sport and many other things of interest to the human being. Thus although this method is probably the 'sure' way of producing a thinking machine it seems to be altogether too slow and impracticable.

xciii Over time, the only representation of the original knowledge becomes the code itself, which by now is

something we can run but not exactly understand. It has become a process, something we can operate but no longer rethink deeply. Even if you have the source code in front of you, there are limits to what a human reader can absorb from thousands of lines of text designed primarily to function, not to convey meaning. When knowledge passes into code, it changes state; like water turned to ice, it becomes a new thing, with new properties. We use it; but in a human sense we no longer know it.

xcivIs a set of rules which tell us, from moment to moment, precisely how to behave.

xcv There is no such thing as an incorrect program. All programs perform some computation correctly.

xcvi Every effective computation can be carried out by a TM [Turing Machine].

xcvii An execution of a transformational program can be viewed as consisting of three consecutive activities. First, the environment prepares an input. Then, the program performs its computation until it terminates, with no intervention from the environment. Lastly, the environment uses the output generated by the program. In the reactive case, we do not enjoy such an orderly sequence of the program and the environment, each acting in its turn. The environment may present new inputs and attempt to use outputs at the same time the program tries to read and write them. One of the main concerns of reactive programming is that, if the program is not fast enough, it may miss some deadlines or fail to respond to or sense important events. Thus, an important characterization of the reactive case is that it describes the situation in which the program and its environment act concurrently, rather than sequentially.

xcviii Interactive models have multiple pragmatic modes of use, while algorithms have a single intended pragmatic interpretation determined by the syntax. The goal of expressing semantics by syntax is replaced by the interactive goal of expressing semantics by multiple pragmatic modes of use. The goal of complete behavior specification is replaced by the goal of harnessing useful forms of partial behavior through interfaces.

xcixEverything, universally, must be paid for. Why should this law change here? In other words, nothing comes from intself, by iteration of tautologia.

c The parasite has placed itself in the most profitable positions, at the intersection of relations.

ci Noise destroys and horrifies. But order and flat repetition ar in the vicinity of death. Noise nourishes a new order. Organization, life, and intelligent thought live between order and noise, between disorder and perfect harmony.

cii Living and inventive path follows the fringed, capricious curve where the simple beach of sand meets the

noisy rolling in of the waves.

ciii Uncompiled source code is logically equivalent to the same code compiled into assembly language and/or linked into machine code. For example, it is absurd to claim that a certain value expressed as a hexadecimal (base 16) number is more or less fundamental than that same value expressed as [a] binary (base 2) number. They are simply two expressions of the same value.

civ So, source code thus only becomes source after the fact. Source code is more accurately a re-source, rather than a source. Source code becomes a source when it becomes integrated with logic gates (and at an even lower level, with the transistors that comprise these gates); when it expands to include software libraries, when it merges with code burned into silicon chips; and when all these signals are carefully monitored, timed, and rectified.

cv Is the only language that is executable.

cvi Real-world computing—what, as I have said, I call "computation in the wild"—turns out not to be formal, on any discernible reading of that recalcitrant and contentious predicate. In particular, the common assumption that computers involve, or are constituted by, the manipulation of symbols "independent of their semantics" proves to be too strong, empirically. To reduce a complex investigation to a single sentence, I can put it this way: the formal-symbol manipulation construal ultimately fails because, in many real-world cases, semantic domains are constitutively implicated in the computational processes defined in, over, around, and through them. Real-world computers are actively involved in their subject matters, that is; they are consequential players in the very worlds they represent. In their comings and goings, they make essential effective use of, and essentially affect, the very states of affairs that their symbol structures are (semantically) about. The concomitant "co-constitution" of symbols and referents, in the assembly of a computational process, defeats the sorts of symbol-reference independence that formality presumes.

cvii And yet, sometimes, there is agreement. The most amazing thing in the world is that agreement, understanding, harmony, sometimes exist.

cviiiThis quasi-object is not an object, but it is one nevertheless, since it is not a subject, since it is in the world ; it is also a quasi-subject, since it marks or designates a subject who, without it, would not be a subject. […] This quasi-object, when being passed, makes the collective , if it stops, it makes the individual.

cix The "we" is not a sum of the "I"'s, but a novelty produced by legacies, concessions, withdrawals, resignations, of the "I".

cx Is the pilot of the whole of our life.

cxi I am thought of, therefore I am.

cxii Initially, living in house-like containers always has a dual character: it means both the coexistence of humans with humans and he community of humans with their invisible companions.

cxiii The angels that pass, be they gods or demons, occupy the crossroads: knots of exchange, changes, cuts, bifurcations of decision, spindles, bundles, where the many come in one single hand. The beginnings of politics.

cxivOur use of the word daemon was inspired by the Maxwell's daemon of physics and thermodynamics. (My background is Physics.) Maxwell's daemon was an imaginary agent which helped sort molecules of different speeds and worked tirelessly in the background. We fancifully began to use the word daemon to describe background processes which worked tirelessly to perform system chores.

cxv By a surrogate situation I mean any kind of real-world structure that is used to stand in for, or take the place of, some aspect of some target situation. By a target situation I mean an actual, possible, or at least superficially possible real-world event or structure that is the ultimate object of my cognitive endeavor.

cxvi The word “substitution”, just like the word “substance”, literally says what stands below the statue, what is hiding in its hollow void or beneath its accidental appearances.